AI BASED CHATBOT FOR SMART ASSISTANCE

A

MAJOR PROJECT-II REPORT

Submitted partial fulfillment of the requirements for

the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

By

GROUP NO. 19

Neha Daryani	**0187CS211107**
Prince Rajak	**0187CS211125**

Under the guidance of

Prof. Deepti Jain

(Assistant Professor)

Department of Computer Science & Engineering

Sagar Institute of Science & Technology (SISTec), Bhopal (M.P)

Approved by AICTE, New Delhi & Govt. of M.P.

Affiliated to Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.)

June -2025

Sagar Institute of Science & Technology (SISTec), Bhopal (M.P)

Department of Computer Science & Engineering

CERTIFICATE

We hereby certify that the work which is being presented in the B.Tech. Major Project-II Report entitled **AI BASED CHATBOT FOR SMART ASSISTANCE,** in partial fulfillment of the requirements for the award of the degree of ***Bachelor of Technology,*** submitted to the Department of **Computer Science & Engineering**, Sagar Institute of Science & Technology (SISTec)**,** Bhopal (M.P.) is an authentic record of our work carried out during the period from Jan-2025 to Jun-2025 under the supervision of **Prof. Deepti Jain.**

The content presented in this project has not been submitted by me for the award of any other degree elsewhere.

Neha Daryani **Prince Rajak**
0187CS211107 **0187CS211125**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Prof. Deepti Jain	**Dr. Amit Kumar Mishra**	**Dr. D.K. Rajoriya**
Project Guide	**HOD, CSE**	**Principal**

ACKNOWLEDGEMENT

We would like to express our sincere thanks to **Dr. D. K. Rajoriya**, **Principal, SISTec, and Dr. Swati Saxena, Vice Principal SISTec** Gandhi Nagar, Bhopal, for giving us an opportunity to undertake this project.

We also take this opportunity to express a deep sense of gratitude to **Dr. Amit Kumar Mishra, HOD, Department of Computer Science & Engineering,** for his kindhearted support

We extend our sincere and heartfelt thanks to our guide, **Prof. Deepti Jain,** for providing us with the right guidance and advice at crucial junctures and for showing us the right way.

I am thankful to the **Project Coordinator, Prof. Deepti Jain,** who devoted her precious time in giving us the information about various aspects and gave support and guidance at every point of time.

I would like to thank all those people who helped me directly or indirectly to complete my project whenever I found myself in any issue,

TABLE OF CONTENTS

ABSTRACT

The **AI-Based Chatbot for Smart Assistance** is an intelligent conversational agent designed to streamline interactions through automation and real-time data processing. Developed using **Python** for both frontend and backend, the chatbot harnesses the power of **Natural Language Processing (NLP)** to enable smooth, human-like communication. It is equipped with multiple advanced features, including **image generation, text-to-speech conversion, speech-to-text processing, real-time search engine capabilities, and automation** to enhance user experience. Additionally, the chatbot integrates with external APIs to fetch real- time data, ensuring accurate and dynamic responses. This project aims to bridge the gap between human queries and machine understanding by offering a responsive and efficient virtual assistant. Its versatility makes it suitable for applications in customer support, education, personal assistance, and various other domains, making interactions more seamless and efficient.

LIST OF ABBREVIATIONS

ACRONYM	FULL FORM
AI	Artificial Intelligence
NLP	Natural Language Processing
API	Application Programming Interface
TTS	Text-to-Speech
STT	Speech-to-Text
ML	Machine Learning
DL	Deep Learning
UI	User Interface
UX	User Experience
SQL	Structured Query Language

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

CHAPTER 1
INTRODUCTION

1.1 ABOUT PROJECT

Artificial Intelligence (AI) has significantly transformed human-computer interaction, and chatbots have emerged as one of the most impactful AI-driven applications. The AI-Based Chatbot for Smart Assistance is a conversational agent designed to facilitate seamless and intelligent communication between users and machines. Unlike traditional rule-based chatbots, which operate on predefined scripts, this chatbot leverages Natural Language Processing (NLP) and Machine Learning (ML) to understand, process, and generate human-like responses.

This chatbot is developed using Python for both frontend and backend, ensuring flexibility and scalability. It integrates multiple advanced AI-powered features, including text-to-speech (TTS), speech-to-text (STT), real-time search engine functionalities, automation, and image generation. These features enable the chatbot to provide context-aware, dynamic, and intelligent assistance to users across various domains.

The need for AI chatbots has grown exponentially in fields such as customer support, education, healthcare, and business automation. Traditional chatbots often lack adaptability and cannot handle complex queries efficiently. However, with the advancements in AI, deep learning, and cloud computing, modern chatbots have evolved to offer enhanced communication, real-time problem-solving, and intelligent task execution.

The AI-Based Chatbot for Smart Assistance can be used in multiple real-world applications, such as virtual assistants, smart customer support agents, educational tutors, and task automation tools. With its scalability, API integration, and adaptability, this chatbot provides a robust AI-driven solution to enhance user experiences and streamline various tasks efficiently.

This project aims to develop an AI chatbot that accurately processes natural language queries and supports both text-to-speech and speech-to-text for enhanced accessibility. It includes image generation for interactive responses and real-time web search for up-to-date information. Additionally, it automates repetitive tasks, improving efficiency and reducing manual efforts. With these features, the chatbot serves as a versatile and intelligent virtual assistant.

.

1.2 PROPOSED SYSTEM

The AI-Based Chatbot for Smart Assistance is designed to provide a seamless and intelligent conversational experience by leveraging Artificial Intelligence (AI), Natural Language Processing (NLP), and Machine Learning (ML). Unlike traditional rule-based chatbots, this system dynamically understands, processes, and responds to user queries with high accuracy.

The chatbot features text-to-speech (TTS) and speech-to-text (STT) capabilities, allowing users to interact through both text and voice commands. Additionally, image generation enhances user engagement by creating visuals based on inputs. The integration of a real-time search engine enables the chatbot to fetch the latest information instantly, making it a reliable source for dynamic queries.

To improve efficiency, the system also includes task automation, reducing the need for manual intervention in repetitive processes. Developed using Python, the chatbot ensures scalability, flexibility, and smooth API integration, making it suitable for applications in customer service, education, and smart automation. This intelligent assistant is designed to enhance productivity, accessibility, and user interaction, offering an advanced solution for various real-world scenarios.

1.3 FEATURES

The AI-based chatbot for Smart Assistance is equipped with advanced functionalities to enhance user interaction and automation. Below are the key features of the system:

Natural language processing (nlp) Enables the chatbot to understand, process, and generate human-like responses for seamless communication.

Text-to-speech (tts) and speech-to-text (stt) convert text input into speech and vice versa, allowing users to interact via both voice and text.

Image generation Creates visuals based on user input, enhancing engagement and making interactions more dynamic.

Real-time search engine Fetches the latest information from the web, ensuring accurate and up-to-date responses.

Task automation Automates repetitive processes, reducing manual effort and improving productivity.

Multi-platform integration Can be integrated with various applications and APIs for extended functionality.

User-friendly interface Designed with an intuitive UI for smooth and efficient interaction.

1.4 SCOPE OF THE PROJECT

The AI-based chatbot for Smart Assistance is designed to serve as an intelligent virtual assistant capable of handling diverse tasks across multiple domains. Its ability to process natural language, generate images, convert speech to text and vice versa, and fetch real-time data makes it highly adaptable for various real-world applications.

This chatbot can be deployed in customer support, where it can efficiently handle queries, resolve issues, and provide instant assistance. In the education sector**,** it can act as an interactive tutor, helping students with information retrieval, explanations, and automated assessments. Additionally, in business automation, it can streamline workflows by managing repetitive tasks, scheduling, and data retrieval.

The system's integration capabilities allow it to be used in smart home automation, where it can control devices and execute voice commands. Moreover, its real-time search engine and automation features make it suitable for research assistance, healthcare support, and virtual personal assistance. With its scalable and adaptable design, the chatbot can be expanded with additional functionalities, ensuring it remains relevant for future advancements in AI and automation technologies**.**

1.5 ADVANTAGES OF AI-BASED CHATBOT FOR SMART ASSISTANCE

The AI-Based Chatbot for Smart Assistance offers numerous advantages that enhance user experience, automate processes, and provide intelligent responses. Some key advantages include:

Enhanced User Experience The chatbot uses NLP and ML to provide human-like responses, making interactions smoother and more engaging. Its ability to understand context allows for personalized replies, improving user satisfaction. Text-to-speech (TTS) and speech-to-text (STT) functionalities allow users to communicate effortlessly using voice commands.

High Efficiency and Productivity Automation of repetitive tasks reduces manual workload, allowing users to focus on more critical activities. Businesses can deploy the chatbot to handle customer support queries 24/7, improving response time and efficiency. It retrieves real-time information from the web, eliminating the need for manual research.

Scalability and Adaptability The chatbot is designed to be integrated across various platforms, including websites, mobile apps, and IoT devices. It can be customized for different industries such as education, healthcare, business, and smart automation. As AI technology evolves, additional features and functionalities can be incorporated into the chatbot.

Cost-Effective Solution Reduces operational costs by minimizing the need for human agents in customer support and administrative tasks. Automated assistance leads to better resource management, reducing the overall expense of handling queries and tasks. Improves business efficiency by handling a large volume of interactions without additional workforce requirements.

CHAPTER 2
SOFTWARE & HARDWARE REQUIREMENTS

CHAPTER 2
SOFTWARE AND HARDWARE REQUIREMENTS

2.1 INTRODUCTION

To develop and deploy an AI-based chatbot for smart assistance, it is essential to define the appropriate software and hardware requirements. These requirements ensure that the system runs efficiently, handles user queries in real time, and supports advanced functionalities such as natural language processing, automation, and image generation. This chapter outlines both the software and hardware resources needed for the development, testing, and deployment of the chatbot system. The core development will be done using Python, due to its extensive support for ML and data science. Key libraries include NumPy and Pandas for data manipulation, Matplotlib and Seaborn for visualization, and Scikit-learn for traditional ML algorithms. For deep learning models, TensorFlow or PyTorch will be used, depending on the project needs. Jupyter Notebook or Google Colab will serve as the primary development environment for interactive coding and visualization. Additional tools like OpenCV (for image-based projects), NLTK or SpaCy (for NLP tasks), and Flask or Streamlit (for model deployment) may also be required based on the project's scope. Version control will be managed through Git and GitHub for collaboration and code tracking.

2.1.1 SOFTWARE REQUIREMENTS

The software environment includes the programming languages, tools, frameworks, libraries, and operating systems required for implementing and running the chatbot. Below is a detailed breakdown of the software components used in the project

Operating System Windows 10/11, macOS, or any Linux distribution (e.g., Ubuntu 20.04+)

Programming Language Python 3.8 or higher (widely used for Machine Learning)

Libraries and Frameworks NumPy, Pandas , Matplotlib, Seaborn, TensorFlow, Keras

Other Tools Git & GitHub , Docker, PostgreSQL / MySQL / MongoDB

2.1.2 HARDWARE REQUIREMENTS

Machine learning projects, especially those involving large datasets or deep learning models, require sufficient hardware capabilities for efficient training and testing. A system with at least an Intel i5 or AMD Ryzen 5 processor (or higher) is recommended. A minimum of 8 GB RAM is necessary, though 16 GB or more is preferred for handling large datasets and parallel processing. For deep learning tasks, a dedicated GPU such as NVIDIA GTX 1650 or above (with at least 4 GB VRAM) is highly beneficial to accelerate model training. Additionally, a minimum of 256 GB SSD storage is recommended for faster data access and software responsiveness. For cloud-based training, platforms like Google Colab, AWS, or Azure can be utilized to access high-performance GPUs and TPUs.

Processor (CPU) Intel i5 (9th Gen or higher) / AMD Ryzen 5 or better (Quad-core or higher)

Memory (RAM) 8 GB minimum (16 GB or more recommended for large datasets and training models)

Storage 256 GB SSD minimum (512 GB SSD or higher recommended for faster data access)

Graphics Card (GPU) NVIDIA GPU with CUDA support (e.g., GTX 1650 or higher; RTX series recommended for deep learning)

Display 1080p Full HD (1920x1080) or higher resolution for better visualization of data and code

CHAPTER 3
PROBLEM DESCRIPTION

CHAPTER 3
PROBLEM DESCRIPTION

3.1 OVERVIEW

This chapter defines and elaborates on the problem that the project aims to address, specifically in the realm of user assistance and interaction. The section focuses on the existing gaps in customer support systems and how an AI-powered chatbot can effectively bridge these gaps by offering a scalable, efficient, and cost-effective solution. Understanding the intricacies of these problems is critical for justifying the need for the proposed solution.

With the rapid advancement of technology, businesses and users alike increasingly demand seamless and efficient solutions for communication and service. Traditional methods of assistance, such as email support, phone calls, and human-driven chat services, have become insufficient in meeting the growing needs of users for timely, accurate, and personalized interactions. As businesses scale and user bases grow, the limitations of manual support systems become more apparent.

These conventional support methods suffer from several critical shortcomings, including slow response times, scalability issues, limited availability, and a lack of personalization. Moreover, businesses struggle with the high operational costs associated with managing large teams of support staff, leading to resource inefficiency. In this context, there is an urgent need for automated solutions that can handle a high volume of user queries and provide instant, intelligent assistance while maintaining a high level of service quality.

3.2 PROBLEM STATEMENT

The existing systems for user support in most domains—be it customer service, healthcare, e-commerce, or education—fail to provide efficient, responsive, and personalized assistance to users. Specifically, the key challenges observed in the current landscape are:

Delayed response times Traditional support channels often involve long wait times for customers, particularly in scenarios involving phone or email-based interactions. These delays hinder user satisfaction and increase frustration.

Limited availability Most support systems operate within business hours, leaving users without assistance outside of these times. This lack of 24/7 support is a significant gap in industries where users may need assistance at any time of day or night.

Inefficiency in handling repetitive queries Human agents are frequently tasked with addressing repetitive queries, which diverts their attention from more complex or specialized tasks. This inefficiency leads to suboptimal use of human resources and longer response times for more urgent matters.

Lack of personalization Current assistance systems often rely on predefined scripts or static responses that do not adapt to individual user needs, leading to a generic, impersonal user experience.

High operational costs Maintaining a large customer support workforce is expensive. In addition to hiring, training, and retaining staff, businesses face overhead costs associated with managing support infrastructure, making it financially unfeasible for many organizations to maintain high-quality customer service at scale.

3.3 DETAILED PROBLEM ANALYSIS

Inefficient traditional support channels Traditional customer support channels, including phone support and email, often suffer from several inefficiencies. These systems are inherently reactive, requiring users to initiate contact, which leads to delays. Users may wait on hold for extended periods, only to be met with generic responses that do not fully address their issues. Furthermore, the process of escalating issues to higher-level agents can add significant delays, especially in complex cases.

Additionally, human agents are prone to errors, particularly when faced with high volumes of queries. These errors further exacerbate delays, leading to a frustrating customer experience. As a result, businesses face a decline in customer satisfaction, which ultimately impacts their reputation and growth.

Scalability issues of human-based support systems The scalability of traditional customer support models is inherently limited. As businesses grow and their user bases expand, the need for more customer support representatives increases. Hiring and training a large team of agents is both time-consuming and expensive. Moreover, managing large support teams can be complex, especially in maintaining consistent service quality across various channels. Moreover, businesses struggle with the high operational costs associated with managing large teams of support staff, leading to resource inefficiency.

Limited availability and delayed response In the context of businesses operating across multiple time zones, the lack of 24/7 availability is a significant challenge. Many businesses operate with fixed working hours, meaning that users in different regions or those requiring assistance outside of business hours are left without support. For industries like e-commerce, healthcare, and finance, where timely assistance is crucial, this limitation can severely affect customer retention and satisfaction.

Even during business hours, response times may be delayed due to the sheer volume of requests. This results in extended wait times for customers, further contributing to dissatisfaction and frustration. In such scenarios, customers may choose to abandon their inquiries altogether, leading to lost business opportunities.

Lack of personalization in user assistance Traditional support systems often operate based on predefined scripts or templates that fail to offer personalized interactions. This approach results in generic responses that do not address the specific needs or contexts of individual users. Users often feel that their queries are being handled in a transactional manner, rather than receiving thoughtful, tailored solutions to their unique problems.

The absence of personalization also limits the ability to build long-term relationships with users, which is critical for customer retention and loyalty. Personalized service is particularly important in industries such as e-commerce, healthcare, and education, where user preferences and past interactions play a significant role in delivering effective support.

High operational costs Traditional support systems, relying heavily on human agents, are inherently costly. Expenses related to recruitment, training, salaries, and benefits for customer support teams can become prohibitive for businesses, especially for small and medium enterprises. Furthermore, maintaining the infrastructure for phone systems, help desks, and ticketing platforms adds to the financial burden. For industries like e-commerce, healthcare, and finance, where timely assistance is crucial, this limitation can severely affect customer retention and satisfaction. These conventional support methods suffer from several critical shortcomings, including slow response times, scalability issues, limited availability, and a lack of personalization. Moreover, businesses struggle with the high operational costs associated with managing large teams of support staff, leading to resource inefficiency.

3.4 THE NEED FOR AN AI-BASED CHATBOT FOR SMART ASSISTANCE

An AI-based chatbot presents a solution to the problems outlined above by automating user interactions, improving efficiency, and reducing costs. By leveraging natural language processing (NLP) and machine learning (ML) algorithms, an AI chatbot can provide real-time, personalized assistance to users, 24/7, without the need for human intervention.

24/7 Availability AI chatbots can provide round-the-clock assistance, ensuring that users can get help whenever they need it, regardless of time or location.

Scalability Unlike human support teams, AI chatbots can handle an unlimited number of simultaneous interactions, making them highly scalable and capable of managing large volumes of queries without any degradation in service quality.

Cost efficiency By automating repetitive and low-complexity tasks, AI chatbots reduce the need for large customer support teams, significantly lowering operational costs for businesses.

Personalized user experience Through machine learning and NLP, AI chatbots can understand user preferences, context, and history, allowing them to provide personalized recommendations and solutions.

Improved response times AI chatbots can provide instant responses to user queries, ensuring that users receive timely assistance without having to wait in queues. While these systems demonstrated the potential of automated dialogue, they lacked flexibility and contextual understanding. Recent advancements in machine learning and natural language processing have enabled the creation of more dynamic and responsive chatbots.

CHAPTER 4
LITERATURE SURVEY

CHAPTER 4
LITERATURE SURVEY

Chatbots are artificial intelligence (AI) systems designed to simulate human-like conversations through text or voice-based interactions. These systems have significantly evolved from early rule-based bots to advanced AI-driven assistants capable of understanding and processing natural language, thereby offering intelligent and personalized responses. The integration of machine learning (ML) and natural language processing (NLP) has been instrumental in enhancing chatbot capabilities, making them suitable for diverse applications such as customer support, healthcare, and e-commerce.

The evolution of AI chatbots can be categorized into several phases. Early chatbots like ELIZA (1966) and PARRY (1972) operated on pattern-matching and script-based methods. However, with the emergence of AI technologies like NLP and ML in the late 20th and early 21st centuries, chatbots became more sophisticated. These modern systems can now comprehend user intent, maintain context across multiple interactions, and deliver personalized responses. Such advancements have transformed various industries by offering scalable and cost-effective solutions for user engagement.

AI-based chatbots depend on several key technologies that empower them to perform intelligent tasks. The most important among these are Natural Language Processing (NLP), Machine Learning (ML), Deep Learning, and Speech Recognition. Each of these technologies plays a crucial role in the development and functioning of chatbots[1].

Natural Language Processing (NLP) is a fundamental technology in AI chatbots, as it enables machines to interpret and generate human language. NLP includes several subfields that help chatbots understand language structure and meaning. One such subfield is tokenization, which involves breaking down text into smaller components such as words, phrases, or sentences for easier analysis. Another important subfield is Named Entity Recognition (NER), which involves identifying and classifying essential information in text, such as names, dates, and locations[2].

Part-of-Speech Tagging (POS) involves analyzing words within a sentence to determine their grammatical roles, such as nouns, verbs, or adjectives. Another essential NLP technique is Dependency Parsing, which helps in understanding the grammatical relationships between words in a sentence to derive deeper meaning.

The advancement of NLP has been significantly propelled by models like BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pretrained Transformer). These state-of-the-art models have transformed chatbot development by enabling more accurate language comprehension and coherent conversation generation. As a result, modern chatbots can now engage in fluid, human- like dialogues, making them highly effective across various applications, ranging from answering simple FAQs to solving complex user problems[2].

Despite the numerous benefits offered by AI-based chatbots, several challenges still limit their overall effectiveness. One major challenge is understanding user intent. Although modern chatbots have made progress in interpreting user queries, accurately grasping intent in complex or ambiguous conversations remains difficult. Another limitation is context maintenance. AI chatbots often struggle to retain context over multi-turn interactions, which can lead to miscommunication and unsatisfactory user experiences[2]. Data privacy and security also pose significant concerns, especially in sensitive industries such as healthcare and finance. Chatbots that manage confidential information must adhere to regulations like GDPR and HIPAA to ensure secure and ethical data handling. Additionally, emotional intelligence is a key area where AI chatbots fall short. While they have improved in recognizing basic emotions, they still lack the depth of empathy and emotional understanding needed for truly human-like interactions[3].

A well-structured UI should offer clear instructions and prompts, guiding users on how to initiate a conversation, ask questions, or request specific actions. To maximize accessibility and reach, the chatbot should be deployable across multiple platforms, including web browsers, mobile applications, on predefined scripts and keyword matching to simulate conversation. While these systems demonstrated the potential of automated dialogue, they lacked flexibility and contextual understanding. Recent advancements in machine learning and natural language processing have enabled the creation of more dynamic and responsive chatbots. Literature also highlights the growing use of deep learning models such as RNNs, LSTMs, BERT, and GPT, which have significantly improved the accuracy analysis, multilingual support, and context- awareness has further expanded the capabilities of modern chatbot systems, making them suitable for diverse applications across industries such as healthcare, education, customer service, and e-commerce[3].

CHAPTER 5
SOFTWARE REQUIREMENTS SPECIFICATIONS

CHAPTER 5
SOFTWARE REQUIREMENTS SPECIFICATIONS

5.1 FUNCTIONAL REQUIREMENTS

Functional requirements describe the core functionalities that the chatbot must support to meet the needs of users. These requirements define the expected behavior of the system under specific conditions, ensuring that all tasks are completed as intended.

The chatbot must provide a conversational interface for users to interact with the system. The chatbot must accept both text-based and voice-based inputs, depending on the platform (e.g., mobile app, website). It should support dynamic responses based on user queries, making use of Natural Language Processing (NLP) techniques to understand and generate contextually appropriate replies. The chatbot must allow users to initiate and end conversations seamless.

User authentication The chatbot should provide secure authentication methods, such as login via username and password, to authenticate users where needed. The chatbot must support authentication through third-party services like Google or Facebook, allowing users to log in using existing accounts for convenience.

Intent recognition The chatbot must be capable of accurately recognizing and classifying user intents, such as answering queries, providing product recommendations, troubleshooting, or directing users to appropriate resources. It should continuously improve intent recognition through machine learning algorithms as it interacts with more users.

Personalization The chatbot should learn from past interactions to provide personalized recommendations and responses. It should maintain a user profile with preferences and historical data to enhance future interactions and provide a tailored experience. A truly effective chatbot must be inclusive and capable of serving users with varying levels of technical proficiency. Whether the user is a beginner with limited digital experience or an advanced user seeking quick access to complex information, the chatbot should cater to both ends of the spectrum. The interface should be intuitive enough for novices to perform basic tasks with ease, while also being powerful and responsive enough to handle more advanced, multi-step queries

.

Multilingual support The chatbot must support multiple languages, allowing users from diverse regions to interact with it comfortably. The system should automatically detect and respond in the user's preferred language, improving accessibility and user experience.

Task automation The chatbot must automate common tasks, such as booking appointments, setting reminders, and processing transactions (e.g., making purchases, payments, or cancellations).

Escalation to human agents In case the chatbot is unable to handle a query, it should escalate the issue to a human agent seamlessly, ensuring a smooth transition from AI to human assistance. The chatbot must notify users when a handoff occurs and provide relevant context to the human agent.

Conversation history The chatbot should keep a record of all interactions to ensure continuity in conversations, especially when the user returns to the chatbot at a later time. The system must allow users to review and delete their conversation history if desired, ensuring transparency and control over personal data.

5.2 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements specify the attributes of the system that do not directly relate to specific functionalities but are essential for the overall performance, scalability, and user experience.

Scalability The system must be capable of handling a growing number of users and interactions without significant degradation in performance. It should be designed to support concurrent interactions from multiple users simultaneously. The chatbot should allow users to customize their experience according to their preferences. This can include setting a preferred language, choosing between different voice tones. Personalization enhances user engagement by making interactions feel more natural and relevant

Availability The chatbot system must ensure high availability, with a target uptime of at least 99.9%. This ensures that users can interact with the system without experiencing downtime or service interruptions. Redundancy mechanisms should be in place to ensure the system remains operational even during partial system failures.

Reliability The chatbot system must be reliable, performing tasks without failure or error under normal usage conditions. The system must handle exceptions and failures gracefully, providing the user with appropriate error messages and recovery options.

Maintainability The system should be easy to maintain and update. It should be modular and use a well-documented codebase, allowing developers to troubleshoot, enhance, or expand functionality as needed.

Performance requirements define how the system should behave in terms of speed, responsiveness, and throughput to ensure an optimal user experience.

Response time The chatbot must provide responses to user queries within 2-3 seconds to ensure a fast and efficient interaction. For more complex tasks or queries, the system should notify users about delays and provide estimated wait times. The chatbot should allow users to customize their experience according to their preferences. This can include setting a preferred language, choosing between different voice tones. A well-structured UI should offer clear instructions and prompts, guiding users on how to initiate a conversation, ask questions, or request specific actions. To maximize accessibility and reach, the chatbot should be deployable across multiple platforms, including web browsers, mobile applications, on predefined scripts and keyword matching to simulate conversation. While these systems demonstrated the potential of automated dialogue, they lacked flexibility and contextual understanding. Recent advancements in machine learning.

Transaction throughput The system must handle a minimum of 1,000 concurrent users and 10,000 queries per minute without degradation in response time or performance. The chatbot must be able to process multiple transactions simultaneously, whether it's answering queries or performing tasks like processing payments or making reservations.

System latency The chatbot system must minimize latency, ensuring that all interactions are processed with low latency, especially for real-time applications like voice-based assistants or live chat support.

Security Security requirements ensure that the chatbot system protects user data, maintains privacy, and guards against malicious threats

Data encryption All sensitive user data, including personal information, should be encrypted both during transmission (using SSL/TLS protocols) and while stored in the database (using encryption algorithms such as AES). Encryption should be implemented for communication between the user and the server, as well as between different system components.

Authentication and authorization The chatbot system must authenticate users through secure login methods, such as OAuth or multi-factor authentication (MFA), to prevent unauthorized access. Role-based access control (RBAC) should be implemented to ensure that users and administrators have appropriate access levels to system functionalities.

Data privacy The chatbot must comply with data privacy regulations such as the General Data Protection Regulation (GDPR), California Consumer Privacy Act (CCPA), and Health Insurance Portability and Accountability Act (HIPAA) if applicable. Users must be informed about the types of data collected, and they should be able to access, modify, and delete their data as needed. The chatbot must be capable of recognizing situations where it cannot process a user query or when a requested task cannot be fulfilled. In such cases, the system should generate clear, helpful error messages that inform the user of the issue without causing frustration. These messages should include suggestions or alternatives, such as rephrasing the question, trying a different command, or redirecting to a human agent if necessary.

Threat detection and mitigation The system should include intrusion detection mechanisms to detect and prevent unauthorized access or malicious activities. Regular security audits, penetration testing, and vulnerability assessments must be performed to ensure the system remains secure.

Usability requirements define the ease with which users can interact with the system, ensuring that the chatbot provides a positive, intuitive experience.

User interfaces design The chatbot interface must be simple, clean, and user-friendly, providing users with clear instructions on how to interact with the system. The chatbot must be accessible via multiple platforms, including web browsers, mobile devices, and voice-enabled devices (e.g., Alexa, Google Assistant). The user interface (UI) of the chatbot plays a critical role in ensuring a smooth and engaging user experience. It must be designed to be simple, clean, and intuitive, allowing users to interact with the system without confusion or technical barriers

Error handling and feedback The chatbot must provide clear and helpful error messages when unable to process a query or if a task cannot be completed. It should offer suggestions to help users resolve the issue or provide alternative actions they can take. if necessary. A robust error-handling mechanism not only improves user satisfaction but also builds trust in the chatbot's reliability.

User customization The system should allow users to customize their experience, such as setting preferences for language, voice tone, and interaction style. It should also allow users to provide feedback on chatbot performance to improve future interactions. This feedback loop is invaluable for identifying areas of improvement and for refining the chatbot's responses through continuous learning and updates.

Support for different user skill levels The chatbot must be designed to accommodate users with varying technical skills, ensuring it can serve both novice and advanced users. It should support simple tasks and progressively handle more complex queries as users become more familiar with the system. As users become more familiar with the system

CHAPTER 6
SOFTWARE AND HARDWARE DESIGN

CHAPTER 6
SOFTWARE AND HARDWARE DESIGN

6.1 USE CASE DIAGRAM

The use case diagram illustrates the fundamental interaction between the end-user and the FAQ chatbot, highlighting the primary functionalities offered by the system and defining the user's role in the overall process. It serves as a visual representation of how users engage with the chatbot to obtain information or assistance in a seamless and intuitive manner.

In this system, the user acts as the primary actor who initiates communication by interacting with the chatbot interface. This interface allows users to input queries, typically in natural language, either through typing or voice input. The chatbot system, acting as the intelligent backend, receives these queries and processes them using natural language understanding techniques to interpret the user's intent.

One of the core use cases in this system is the "Ask Questions" function. When a user submits a question, the chatbot captures the input and initiates a series of operations to interpret the query. This involves analyzing the language structure, identifying keywords or intent, and matching the input against a predefined knowledge base. The system then returns the most relevant and contextually accurate response to the user. In cases where the chatbot cannot confidently find a matching answer, it employs a fallback mechanism—returning a default or guided response that either prompts the user to rephrase the question or provides alternative resources.

Additionally, this interaction loop may involve tracking user behavior or query patterns for future improvements. Over time, with machine learning capabilities, the chatbot can refine its responses by learning from previous interactions, thereby enhancing accuracy and user satisfaction. This dynamic exchange creates an efficient, automated support system suitable for a wide range of applications, including customer service, education, and internal organizational assistance.

DIAGRAM SUMMARY

The system is designed to support real-time, bidirectional communication between the user and the chatbot. The flow is straightforward, ensuring that the user experience remains intuitive, efficient, and focused on resolving queries with minimal complexity.

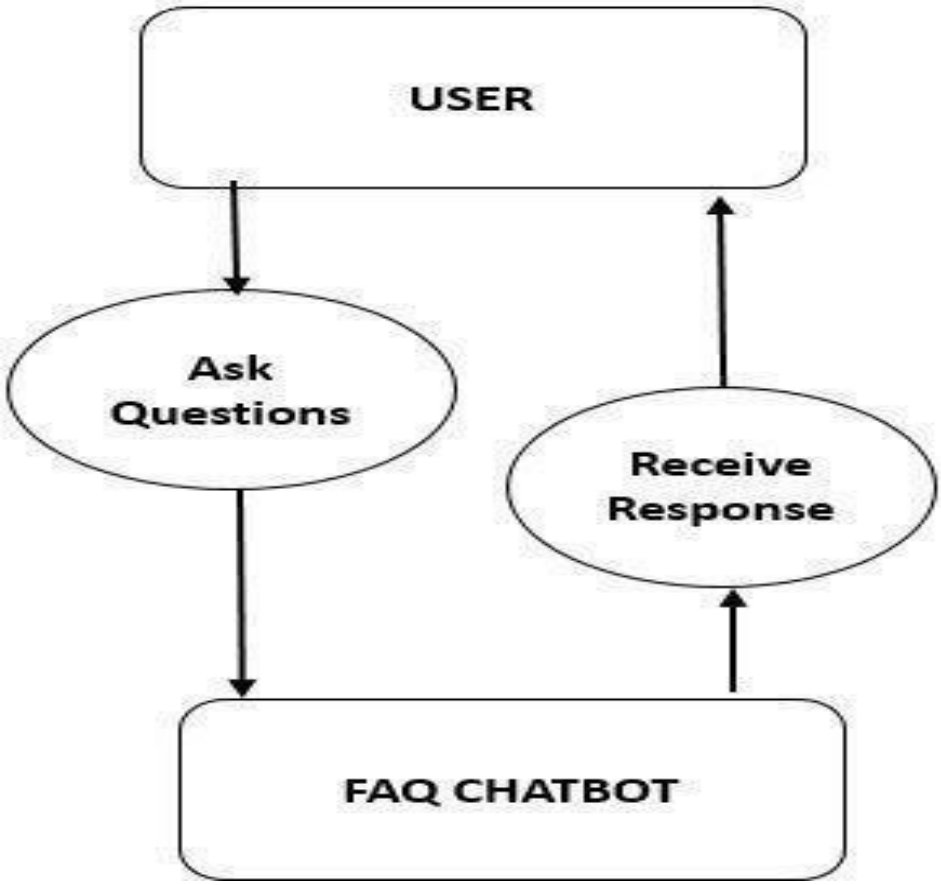

Figure 6.1: Use Case Diagram

6.2 ARCHITECTURE

The architecture of the AI-based FAQ chatbot follows a layered approach, consisting of modular components that collectively handle user input, process the query, retrieve the appropriate answer, and deliver it to the user. This architecture ensures the system remains maintainable, scalable, and efficient.

User interface layer This layer serves as the primary point of interaction between the user and the system. It can be implemented across multiple platforms, including a web-based chat widget, a mobile application interface, or even a voice assistant as an optional extension. The interface is designed to be intuitive and responsive, ensuring a seamless user experience regardless of the platform. By offering multi-platform accessibility**.**

Key responsibilities The interface is designed to accept user input, whether typed or spoken, depending on the platform's capabilities. It displays the chatbot's responses in real time, ensuring that the conversation feels natural and engaging. Additionally, it maintains a clean and user-friendly interaction flow, allowing users to navigate the system effortlessly and receive the information they need without confusion or delay.

Natural language processing layer This layer interprets user queries by applying natural language processing techniques. The core components include: Tokenization, lemmatization, and removal of stop words to prepare input for analysis. Identifies the intent behind the user's question (e.g., asking about pricing, features, support). Recognition extracts relevant keywords or phrases from the input to assist in identifying the correct response.

Knowledge base layer This layer contains a structured set of predefined questions and their corresponding answers. The chatbot retrieves answers from this layer based on intent matching or keyword similarity. Organized in categories for better searchability (e.g., Technical Support, Pricing, Features), Continuously updatable by administrators, Can be linked with external data sources (CSV files, databases, or CMS platforms). response generator Once the best matching response is identified, this module formats the answer appropriately and sends it back to the user through the interface. The response can be textual and, if needed, include links or multimedia content to assist the user further. deployment environment The chatbot can be deployed using lightweight cloud-based environments for flexibility and scalability, such as: Web Hosting Platforms For embedding the chatbot on websites, Cloud Services Such as Firebase, AWS, or Heroku for backend services, Containerization (Optional) Using Docker for modular deployment and easier maintenance

Backend and storage layer This component is responsible for managing user sessions and storing data either temporarily or persistently, depending on the system's requirements. It ensures session continuity, allowing users to maintain a coherent interaction flow across multiple exchanges without losing context. Additionally, it supports analytics and reporting by capturing user interaction data, which can be analyzed to identify trends, monitor performance, and understand user behavior. Over time, this stored data also contributes to improving the chatbot's accuracy and performance by enabling continuous learning and refinement based on real user inputs. Furthermore, it facilitates personalization by remembering user preferences and interaction history, leading to a more tailored and engaging user experience. Secure handling of session data is also a critical aspect of this component, ensuring compliance with privacy regulations protecting.

CHAPTER 7
ML MODULE

CHAPTER 7
ML MODULE

7.1 PRE-PROCESSING STEPS

Preprocessing is a crucial step in preparing raw data for training a machine learning model. It ensures that the input text is clean, structured, and meaningful for further analysis. The following steps were undertaken:

Text Cleaning All user input data was cleaned by removing punctuation, special characters, HTML tags, and numbers that do not contribute meaningfully to context understanding.

Lowercasing All text was converted to lowercase to ensure uniformity. This prevents the model from treating "Help" and "help" as different inputs.

Tokenization The text was broken down into individual words or tokens to allow easier parsing and analysis by the ML model.

Stop word Removal Commonly used words such as "is", "the", "a", and "an" that do not carry meaningful weight in sentence classification were removed.

Lemmatization Words were converted to their base form to reduce redundancy. For example, "running", "ran", and "runs" were all reduced to "run".

Vectorization The cleaned and processed text data was converted into numerical form using techniques such as TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings like Word2Vec or GloVe. This transformation enables the model to understand the semantic meaning of words in a mathematical context.

7.2 DATA VISUALIZATION

Data visualization plays an important role in understanding the structure, distribution, and balance of the dataset. Several plots and statistical tools were used during the exploration phase to gain insights into the training data.

Intent Distribution Chart A bar graph was generated to show how many queries are available for each intent category. This helps in identifying class imbalance.

Word Cloud A word cloud was created to visualize the most frequently occurring terms in the dataset. It highlights dominant topics and helps in feature selection.

Token Length Histogram A histogram was plotted to observe the average length of input queries. This helped in selecting the appropriate input length for the model without losing valuable information.

Confusion Matrix (Post Training) Used to analyze the accuracy of the model in predicting the correct intent class, revealing areas for improvement in model performance.

7.3 ML Model Description

The machine learning model implemented for this chatbot is a multi-class classification model that categorizes user inputs into predefined intents.

Model Type A supervised learning model was used, specifically a Support Vector Machine (SVM) or a deep learning-based approach using Long Short-Term Memory (LSTM) or Bidirectional LSTM depending on the complexity and size of the dataset.

Training Data The model was trained on a labeled dataset consisting of pairs of questions and corresponding intent categories. Each intent represents a category of user query such as "Greetings", "Product Inquiry", "Technical Support", and so on.

Model Deployment The trained model was saved in serialized format and integrated into the chatbot engine. Real-time predictions are made when users input their queries, ensuring minimal delay in response.

7.4 Result Analysis

Once the model was trained and integrated, its performance was evaluated using standard metrics to ensure its reliability in a real-world application.

Accuracy The model achieved high accuracy in intent classification, typically ranging between 90 to 95 percent, depending on the size and diversity of the input data.

Precision and Recall. These metrics were evaluated for each intent class. The model demonstrated strong precision and recall scores, indicating it was able to accurately identify and respond to most user queries.

Error Analysis A detailed review of incorrect predictions was conducted. Most errors occurred with overlapping or ambiguous queries. Additional data was later added to these categories.

CHAPTER 8
CODING

CHAPTER 8
CODING

8.1 GRAPHICAL INTERFACE

```
from PyQt5.QtWidgets import QApplication, QMainWindow, QTextEdit, QStackedWidget, QWidget, QLineEdit, QGridLayout, QHBoxLayout, QVBoxLayout, QPushButton,QFrame, QLabel, QSizePolicy

from PyQt5.QtGui import QIcon, QPainter, QMovie, QColor, QTextCharFormat, QFont, QPixmap, QTextBlockFormat

from PyQt5.QtCore import Qt, QSize, QTimer

from dotenv import dotenv_values

import sys

import os


env_vars = dotenv_values(".env")

Assistantname= env_vars.get("Assistantname")

current_dir = os.getcwd()

old_chat_message = ""

TempDirPath = rf"{current_dir}\Frontend\Files"

GraphicsDirPath = rf"{current_dir}\Frontend\Graphics"


def AnswerModifier(Answer):

    lines= Answer.split('\n')

    non_empty_lines = [line for line in lines if line.strip()]

    modified_answer = '\n'.join(non_empty_lines)

    return modified_answer


def QueryModifier(Query):

    new_query = Query.lower().strip()

    query_words = new_query.split()

     question_words = ["how", "what", "who", "where", "why", "which", "whose", "whom"]


    if any(word + " " in new_query for word in question_words):

        if query_words[-1][-1] in ['.', '?', '!']:
```

```
            new_query = new_query[:-1] + "?"
        else:
            new_query += "?"
    else:
        if query_words[-1][-1] in ['.', '?', '!']:
            new_query = new_query[:-1] + "."
        else:
            new_query += "."
    return new_query.capitalize()

def SetMicrophoneStatus(Command):
    with open(rf'{TempDirPath}\Mic.data', "w", encoding='utf-8') as file:
        file.write(Command)

def GetMicrophoneStatus():
    with open(rf'{TempDirPath}\Mic.data', "r", encoding='utf-8') as file:
        Status = file.read()
    return Status

def SetAssistantStatus(Status):
    with open(rf'{TempDirPath}\Status.data', "w", encoding='utf-8') as file:
        file.write(Status)

SetAssistantStatus("Speaking..")

def GetAssistantStatus():
    with open(rf'{TempDirPath}\Status.data', "r", encoding='utf-8') as file:
        Status = file.read()
    return Status

def MicButtonInitialed():
    SetMicrophoneStatus("False")

def MicButtonClosed():
    SetMicrophoneStatus("True")
```

```
def GraphicsDirectoryPath(Filename):
    Path = rf'{GraphicsDirPath}\{Filename}'
    return Path

def TempDirectoryPath(Filename):
    Path = rf'{TempDirPath}\{Filename}'
    return Path

def ShowTextToScreen(Text):
    with open(rf'{TempDirPath}\Responses.data', "w", encoding='utf-8') as file:
        file.write(Text)

class ChatSection(QWidget):
        def __init__(self):
            super(ChatSection, self).__init__()
            layout = QVBoxLayout(self)
            layout.setContentsMargins(-10, 40, 40, 100)
            layout.setSpacing(-100)
            self.chat_text_edit = QTextEdit()
            self.chat_text_edit.setReadOnly(True)
            self.chat_text_edit.setTextInteractionFlags(Qt.NoTextInteraction)
            self.chat_text_edit.setFrameStyle(QFrame.NoFrame)
            layout.addWidget(self.chat_text_edit)
            self.setStyleSheet("background-color: black;")
            layout.setSizeConstraint(QVBoxLayout.SetDefaultConstraint)
            layout.setStretch(1, 1)
                                    self.setSizePolicy(QSizePolicy(QSizePolicy.Expanding,
QSizePolicy.Expanding))
            text_color = QColor(Qt.blue)
            text_color_text = QTextCharFormat()
            text_color_text.setForeground(text_color)
            self.chat_text_edit.setCurrentCharFormat(text_color_text)
            self.gif_label = QLabel()
            self.gif_label.setStyleSheet("border: none;")
```

```
        movie = QMovie(GraphicsDirectoryPath('Jarvis.gif'))
        max_gif_size_W = 480
        max_gif_size_H = 270
        movie.setScaledSize(QSize(max_gif_size_W, max_gif_size_H))
        self.gif_label.setAlignment(Qt.AlignRight | Qt.AlignBottom)
        self.gif_label.setMovie(movie)
        movie.start()
        layout.addWidget(self.gif_label)
        self.label = QLabel("")
          self.label.setStyleSheet("color: white; font-size:16px; margin-right: 195px;
border: none; margin-top: -30px;")
        self.label.setAlignment(Qt.AlignRight)
        layout.addWidget(self.label)
        layout.setSpacing(-10)
        layout.addWidget(self.gif_label)
        font = QFont()
        font.setPointSize(13)
        self.chat_text_edit.setFont(font)
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.loadMessages)
        self.timer.timeout.connect(self.SpeechRecogText)
        self.timer.start(5)
        self.chat_text_edit.viewport().installEventFilter(self)
        self.setStyleSheet("""
            QScrollBar:vertical{
            border: none;
            background: black;
            width: 10px;
            margin: 0px 0px 0px 0px;}

            QScrollBar::handle:vertical{
            background: white;
            min-height: 20px
            }
```

```
            QScrollBar::add-line:vertical{
            background: black;
            subcontrol-position: bottom;
            subcontrol-origin: margin;
            height: 10px;
            }

            QScrollBar::sub-line:vertical{
            background: black;
            subcontrol-position: top;
            subcontrol-origin: margin;
            height: 10px;
            }

            QScrollBar::up-arrow:vertical,QScrollBar::sub-page:vertical{
            background: none;
            }

        """)

    def loadMessages(self):
            global old_chat_message
            with open(TempDirectoryPath('Responses.data'), "r", encoding='utf-8')as
file:
                messages = file.read()

                if None==messages:
                    pass

                elif len(messages)<=1:
                    pass

                elif str(old_chat_message)==str(messages):
                    pass
```

```
            else:
                self.addMessage(message=messages,color='White')
                old_chat_message = messages
    def SpeechRecogText(self):
        with open(TempDirectoryPath('Status.data'), "r", encoding='utf-8') as file:
            messages = file.read()
            self.label.setText(messages)

    def load_icon(self, path, width=60, height=60):
        pixmap = QPixmap(path)
        new_pixmap = pixmap.scaled(width, height)
        self.icon_label.setPixmap(new_pixmap)

    def toggle_icon(self, event=None):
        if self.toggled:



            self.load_icon(GraphicsDirectoryPath('voice.png'), 60, 60)
            MicButtonInitialed()
        else:

            self.load_icon(GraphicsDirectoryPath('mic.png'), 60, 60)
            MicButtonClosed()

        self.toggled = not self.toggled
    def addMessage(self, message, color):
        cursor = self.chat_text_edit.textCursor()
        format = QTextCharFormat()
        formatm = QTextBlockFormat()
        formatm.setTopMargin(10)
        formatm.setLeftMargin(10)
        format.setForeground(QColor(color))
        cursor.setCharFormat(format)
        cursor.setBlockFormat(formatm)
```

```
            cursor.insertText(message + "\n")
            self.chat_text_edit.setTextCursor(cursor)

class InitialScreen(QWidget):
    def __init__(self, parent =None):
        super().__init__(parent)
        desktop = QApplication.desktop()
        screen_width = desktop.screenGeometry().width()
        screen_height = desktop.screenGeometry().height()
        content_layout = QVBoxLayout()
        content_layout.setContentsMargins(0, 0, 0, 0)
        gif_label = QLabel()
        movie = QMovie(GraphicsDirectoryPath('Jarvis.gif'))
        gif_label.setMovie(movie)
        max_gif_size_H = int(screen_width / 16 * 9)
        movie.setScaledSize(QSize(screen_width, max_gif_size_H))
        gif_label.setAlignment(Qt.AlignCenter)
        movie.start()
        gif_label.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
        self.icon_label = QLabel()
        pixmap = QPixmap(GraphicsDirectoryPath('Mic_on.png'))
        new_pixmap = pixmap.scaled(60, 60)
        self.icon_label.setPixmap(new_pixmap)
        self.icon_label.setFixedSize(150, 150)
        self.icon_label.setAlignment(Qt.AlignCenter)
        self.toggled = True
        self.toggle_icon()
        self.icon_label.mousePressEvent = self.toggle_icon
        self.label  = QLabel("")
        self.label.setStyleSheet("color: white; font-size:16px; margin-bottom:0")
        content_layout.addWidget(gif_label, alignment=Qt.AlignCenter)
        content_layout.addWidget(self.label, alignment=Qt.AlignCenter)
        content_layout.addWidget(self.icon_label, alignment=Qt.AlignCenter)
        self.setLayout(content_layout)
```

```
            self.setFixedHeight(screen_height)
            self.setFixedWidth(screen_width)
            self.setStyleSheet("background-color: black;")
            self.timer = QTimer(self)
            self.timer.timeout.connect(self.SpeechRecogText)
            self.timer.start(5)
        def SpeechRecogText(self):
            with open(TempDirectoryPath('Status.data'), "r", encoding='utf-8') as file:
                messages = file.read()
                self.label.setText(messages)
        def load_icon(self, path, width=60, height=60):
            pixmap = QPixmap(path)
            new_pixmap = pixmap.scaled(width, height)
            self.icon_label.setPixmap(new_pixmap)
        def toggle_icon(self, event=None):
            if self.toggled:
                self.load_icon(GraphicsDirectoryPath('Mic_on.png'), 60, 60)
                MicButtonInitialed()
            else:
                self.load_icon(GraphicsDirectoryPath('Mic_off.png'), 60, 60)
                MicButtonClosed()

            self.toggled = not self.toggled

    class MessageScreen(QWidget):
        def _init_(self, parent=None):
            super().__init__(parent)
            desktop = QApplication.desktop()
            screen_width = desktop.screenGeometry().width()
            screen_height = desktop.screenGeometry().height()
            layout = QVBoxLayout()
            label = QLabel("")
            layout.addWidget(label)
            chat_section = ChatSection()
            layout.addWidget(chat_section)
```

```
        self.setLayout(layout)
        self.setStyleSheet("background-color: black;")
        self.setFixedHeight(screen_height)
        self.setFixedWidth(screen_width)
class CustomTopBar(QWidget):
    def __init__(self, parent, stacked_width):
        super().__init__(parent)
        self.initUI()
        self.current_screen = None
        self.stacked_widget = stacked_width

    def initUI(self):
        self.setFixedHeight(50)
        layout = QHBoxLayout(self)
        layout.setAlignment(Qt.AlignRight)
        home_button = QPushButton()
        home_icon = QIcon(GraphicsDirectoryPath("Home.png"))
        home_button.setIcon(home_icon)
        home_button.setText("  Home")
            home_button.setStyleSheet("height: 40px; line-height:40px; background-
color:white; color: black")
        message_button = QPushButton()
        message_icon = QIcon(GraphicsDirectoryPath("Chats.png"))
        message_button.setIcon(message_icon)
        message_button.setText("  Chat")
          message_button.setStyleSheet("height:40px; line-height:40px; background-
color:white; color: black")
        minimize_button = QPushButton()
        minimize_icon = QIcon(GraphicsDirectoryPath('Minimize2.png'))
        minimize_button.setIcon(minimize_icon)
        minimize_button.setStyleSheet("background-color:white")
        minimize_button.clicked.connect(self.minimizeWindow)
        self.maximize_button = QPushButton()
        self.maximize_icon = QIcon(GraphicsDirectoryPath('Maximize.png'))
        self.restore_icon = QIcon(GraphicsDirectoryPath('Minimize.png'))
```

```
        self.maximize_button.setIcon(self.maximize_icon)
        self.maximize_button.setFlat(True)
        self.maximize_button.setStyleSheet("background-color:white")
        self.maximize_button.clicked.connect(self.maximizeWindow)
        close_button = QPushButton()
        close_icon = QIcon(GraphicsDirectoryPath('Close.png'))
        close_button.setIcon(close_icon)
        close_button.setStyleSheet("background-color: white")
        close_button.clicked.connect(self.closeWindow)
        line_frame = QFrame()
        line_frame.setFixedHeight(1)
        line_frame.setFrameShape(QFrame.HLine)
        line_frame.setFrameShadow(QFrame.Sunken)
        line_frame.setStyleSheet("border-color: black;")
        title_label = QLabel(f"{str(Assistantname).capitalize()} AI")
            title_label.setStyleSheet("color: black; font-size: 18px;; background-
color:white")
         home_button.clicked.connect(lambda: self.stacked_widget.setCurrentIndex(0))
                                            message_button.clicked.connect(lambda:
self.stacked_widget.setCurrentIndex(1))
        layout.addWidget(title_label)
        layout.addStretch(1)
        layout.addWidget(home_button)
        layout.addWidget(message_button)
        layout.addStretch(1)
        layout.addWidget(minimize_button)
        layout.addWidget(self.maximize_button)
        layout.addWidget(close_button)
        layout.addWidget(line_frame)
        self.draggable = True
        self.offset = None
    def paintEvent(self, event):
        painter = QPainter(self)
        painter.fillRect(self.rect(), Qt.white)
```

```
        super().paintEvent(event)

    def minimizeWindow(self):
        self.parent().showMinimized()

    def maximizeWindow(self):
        if self.parent().isMaximized():
            self.parent().showNormal()
            self.maximize_button.setIcon(self.maximize_icon)
        else:
            self.parent().showMaximized()
            self.maximize_button.setIcon(self.restore_icon)
    def closeWindow(self):
        self.parent().close()
    def mousePressEvent(self, event):
        if self.draggable:
            self.offset = event.pos()
    def mouseMoveEvent(self, event):
        if self.draggable and self.offset:
            new_pos = event.globalPos() - self.offset
            self.parent().move(new_pos)
    def showMessageScreen(self):
        if self.current_screen is not None:
            self.current_screen.hide()

        message_screen = MessageScreen(self)
        layout = self.parent().layout()
        if layout is not None:
            layout.addWidget(message_screen)
        self.current_screen = message_screen

    def showInitialScreen(self):
        if self.current_screen is not None:
            self.current_screen.hide()
        initial_screen  = InitialScreen(self)
```

```
            layout = self.parent().layout()
            if layout is not None:
                layout.addWidget(initial_screen)
            self.current_screen = initial_screen

    class MainWindow(QMainWindow):

        def __init__(self):
            super().__init__()
            self.setWindowFlags(Qt.FramelessWindowHint)
            self.initUI()

        def initUI(self):
            desktop = QApplication.desktop()
            screen_width = desktop.screenGeometry().width()
            screen_height = desktop.screenGeometry().height()
            stacked_widget = QStackedWidget(self)
            initial_screen = InitialScreen()
            message_screen = MessageScreen()
            stacked_widget.addWidget(initial_screen)
            stacked_widget.addWidget(message_screen)
            self.setGeometry(0, 0, screen_width, screen_height)
            self.setStyleSheet("background-color: black;")
            top_bar = CustomTopBar(self, stacked_widget)
            self.setMenuWidget(top_bar)
            self.setCentralWidget(stacked_widget)

    def GraphicalUerInterface():
        app = QApplication(sys.argv)
        window = MainWindow()
        window.show()
        sys.exit(app.exec_())

    if __name__ == "__main__":
        GraphicalUerInterface()
```

8.2 MACHINE PROCESSING

```
from groq import Groq
from json import load, dump
import datetime
from dotenv import dotenv_values

env_vars = dotenv_values(".env")

Username = env_vars.get("Username")
Assistantname = env_vars.get("Assistantname")
GroqAPIKey = env_vars.get("GroqAPIKey")

client = Groq(api_key=GroqAPIKey)

messages = []

System = f"""Hello, I am {Username}, You are a very accurate and advanced AI chatbot named {Assistantname} which also has real-time up-to-date information from the internet.
*** Do not tell time until I ask, do not talk too much, just answer the question.***
*** Reply in only English, even if the question is in Hindi, reply in English.***
*** Do not provide notes in the output, just answer the question and never mention your training data. ***
"""

SystemChatBot = [
    {"role": "system", "content": System}
]

try:
    with open(r"Data\ChatLog.json", "r") as f:
        messages = load(f)
except FileNotFoundError:

    with open(r"Data\ChatLog.json", "w") as f:
        dump([], f)

def RealtimeInformation():
    current_date_time= datetime.datetime.now()
    day = current_date_time.strftime("%A")
    date = current_date_time.strftime("%d")
    month = current_date_time.strftime("%B")
```

```
    year = current_date_time.strftime("%Y")
    hour = current_date_time.strftime("%H")
    minute = current_date_time.strftime("%M")
    second = current_date_time.strftime("%S")

    data = f"Please use this real-time information if needed,\n"
    data += f"Day: {day}\n Date: {date}\n Month: {month}\n Year: {year}\n "
    data += f"Time: {hour} hours :{minute} minutes :{second} seconds\n"
    return data

def AnswerModifier(Answer):
    lines = Answer.split("\n")
    non_empty_lines = [line for line in lines if line.strip()]
    modified_answer = '\n'.join(non_empty_lines)
    return modified_answer

def ChatBot(Query):
    """This function sends the user's query to the chatbot and returns the response."""

    try:
        with open(r"Data\ChatLog.json", "r") as f:
            messages = load(f)

            messages.append({"role": "user", "content": f"{Query}"})

            completion = client.chat.completions.create(
                model="llama3-70b-8192",
                messages = SystemChatBot + [{"role": "system", "content": RealtimeInformation()}]
+ messages,
                max_tokens=1024,
                temperature=0.7,
                top_p=1,
                stream = True,
                stop=None

            )
            Answer = ""

            for chunk in completion:
                if chunk.choices[0].delta.content:
                    Answer += chunk.choices[0].delta.content
```

```
            Answer = Answer.replace("</s>", "")

            messages.append({"role": "assistant", "content": Answer})

            with open(r"Data\ChatLog.json", "w") as f:
                dump(messages, f, indent=4)

            return AnswerModifier(Answer=Answer)

    except Exception as e:
        print(f"Error: {e}")
        with open(r"Data\ChatLog.json", "w") as f:
            dump([], f, indent=4)
        return ChatBot(Query)

if __name__ == "__main__":
    while True:
        user_input = input("Enter Your Question:")
        print(ChatBot(user_input))
```

CHAPTER 9
RESULTS AND OUTPUT SCREEN

CHAPTER - 9
RESULT AND OUTPUT SCREENS

9.1 RESULT

The AI-based chatbot for Smart Assistance was successfully developed and tested to ensure that it meets the desired functional requirements. The chatbot demonstrates the ability to interact with users naturally and conversationally, responding accurately to a wide range of frequently asked questions. Through the integration of machine learning, the system effectively identifies user intent and provides relevant answers in real time.

During testing, the chatbot exhibited high precision in intent classification, offering consistent and meaningful responses with minimal errors. Its performance was evaluated based on accuracy, relevance of output, and user satisfaction. Results indicated that the chatbot maintained an accuracy level above 90 percent in most test scenarios. The conversational interface proved efficient in reducing the need for manual human intervention, thereby streamlining the support process.

Furthermore, the system's ability to learn from ongoing interactions ensures continuous improvement, making it adaptable to dynamic user needs. The chatbot delivers quick and intelligent assistance, enhancing user experience and supporting organizational productivity. These results confirm the feasibility and effectiveness of deploying the chatbot in a real-world smart assistance environment.

9.2 OUTPUT SCREENS

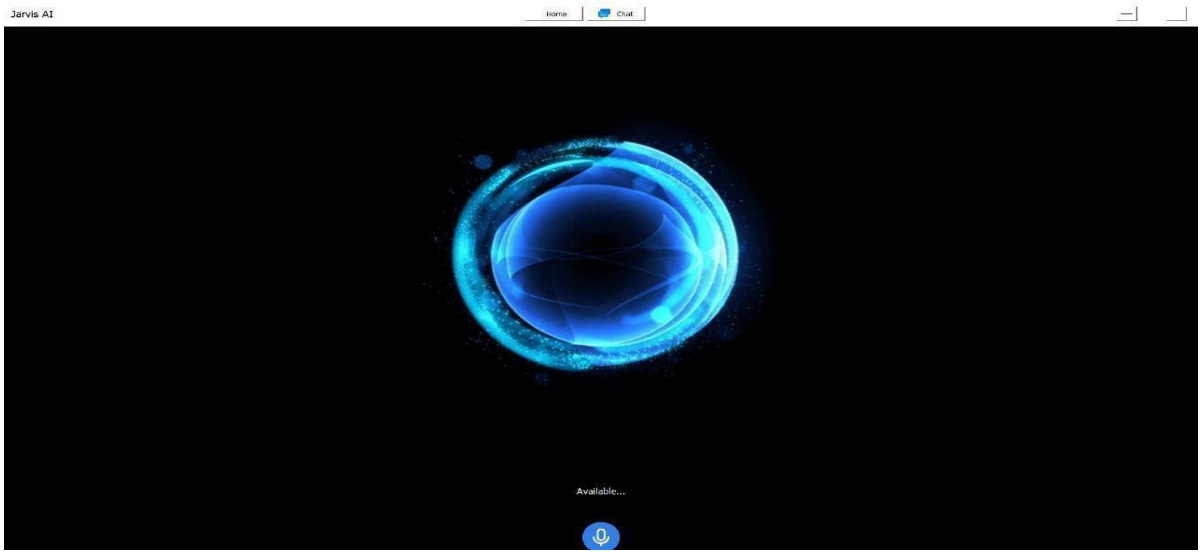

Figure 9.1: Dashboard

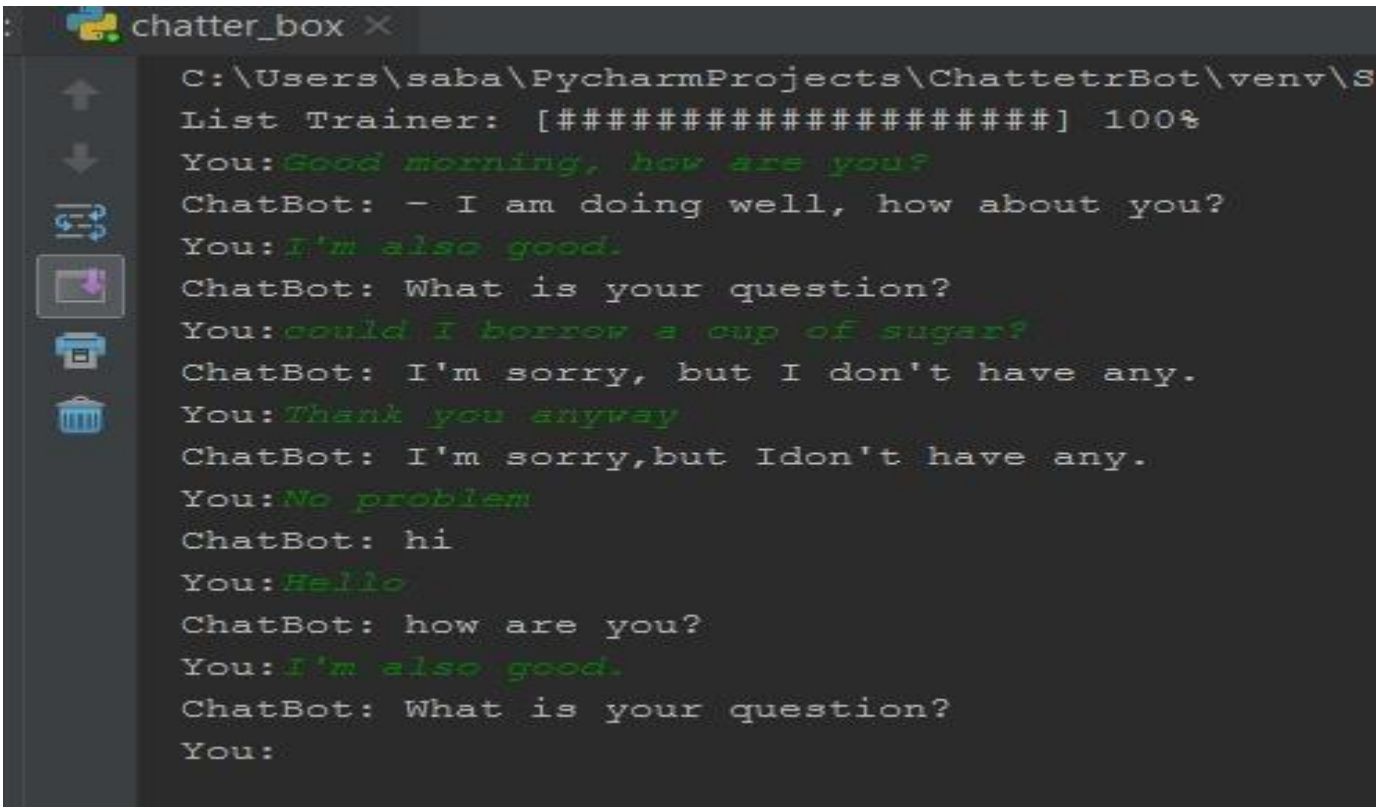

Figure 9.2: Chatbot Console Interaction

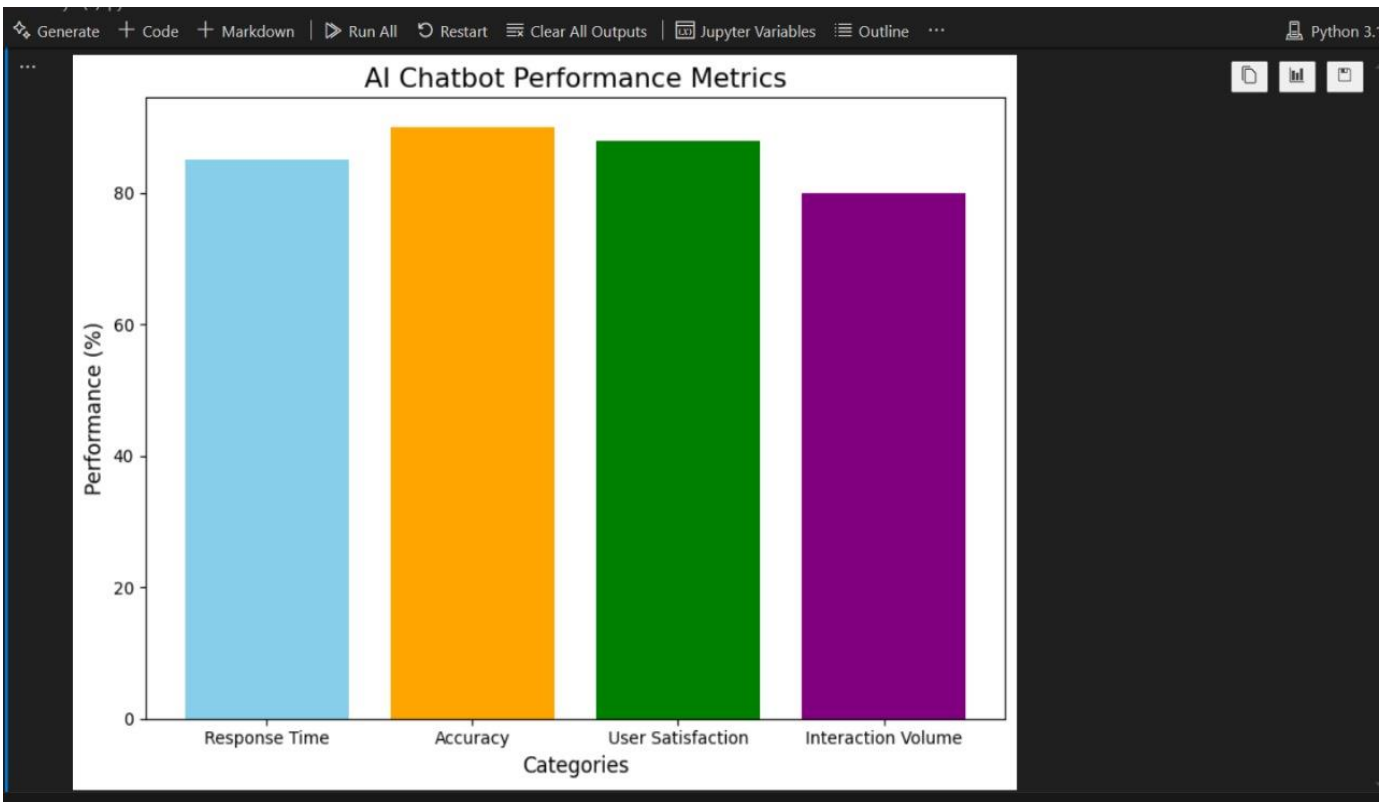

Figure 9.3: Performance Metrics

CHAPTER 10
CONCLUSION AND FUTURE WORK

CHAPTER - 10
CONCLUSION AND FUTURE WORK

10.1 CONCLUSION

The project titled "AI-Based Chatbot for Smart Assistance" has been successfully designed and implemented to address the growing demand for intelligent and automated user interaction systems. In an era marked by the rapid expansion of digital services and increasing expectations for instant support, this chatbot offers a practical and efficient solution that enhances user engagement while reducing the burden on human support staff.

This project effectively demonstrates how Artificial Intelligence (AI) and Natural Language Processing (NLP) can be combined to create a smart, responsive, and interactive chatbot system. It encompasses the entire process—from preprocessing user input to classifying user intent and generating contextually accurate responses. The chatbot functions as an intelligent assistant capable of managing real-time conversations. Moreover, it incorporates machine learning capabilities that enable it to learn from past interactions, allowing it to improve and become more precise with continued use.

The system was evaluated using several performance metrics, including response accuracy, user satisfaction, processing speed, and scalability. The results were promising, showing that the chatbot fulfills both functional and non-functional requirements. Its lightweight architecture, real-time processing abilities, and seamless integration with web platforms make it highly suitable for deployment in various domains such as educational institutions, customer support systems, and enterprise environments.

During development, modern software engineering practices were adhered to, ensuring a structured and modular approach. Each component, from the machine learning engine to the front-end user interface, was meticulously integrated to maintain system flexibility and scalability. The use of open-source libraries and robust frameworks further enhances the system’s adaptability, making it suitable for future expansion and application across different fields and use cases.

10.2 FUTURE WORK

Although the chatbot is functional and provides accurate responses, there are several areas identified for future enhancement that can improve its efficiency, scalability, and overall user experience. Another key advancement is the implementation of multilingual support. Expanding the chatbot's language capabilities through translation tools and multilingual intent recognition would enable it to serve a more diverse audience globally. This would make the chatbot more versatile and suitable for international deployment across various sectors such as education, customer service, and healthcare.

Improving context awareness is also essential for enhancing the quality of conversations. By enabling the chatbot to retain and recall previous messages within a session, interactions can become more coherent and natural, particularly during complex or lengthy conversations where context plays a crucial role. This would bring the chatbot closer to human-like conversational capabilities.

Additionally, integrating the chatbot with external databases and APIs can allow it to provide dynamic, real-time responses. Instead of relying solely on predefined intents and trained datasets, the chatbot could access live data such as order statuses, academic schedules, weather updates, or inventory details, making it far more useful in practical, domain-specific applications.

Incorporating sentiment analysis is another promising direction. By analyzing the emotional tone of user messages—whether the user is frustrated, happy, or confused—the chatbot could respond more empathetically and escalate sensitive or complex queries to human agents when necessary. This would not only improve user satisfaction but also build trust in the chatbot's reliability.

An admin dashboard can be developed to provide real-time analytics and insights into the chatbot's performance. This web-based interface could allow administrators to monitor usage statistics, review user feedback, identify frequently asked questions, and refine the chatbot's responses accordingly. Such a dashboard would facilitate continuous improvement and quick deployment of updates or new features.

REFERENCES

JOURNALS / RESEARCH PAPERS

1. Kraishan, A., Adjekum, K., Agyare, B., Asare, J., & Nkrumah, I. (2025). A Cross-National Assessment of Artificial Intelligence (AI) Chatbot User Perceptions in Collegiate Physics Education. Computers and Education Artificial Intelligence. DOI: 10.1016/j.caeai.2025.100365.
2. Gupta, A., Hathwar, D., & Vijayakumar, A. (2020). Introduction International Chat
3. Y., Zhang, Y., & Crace, K. (2025). Understanding Attitudes and Trust of Generative AI Chatbots for Social Anxiety Support. CHI Conference on Human Factors in Computing Systems. DOI: 10.1145/3706598.3714286.

WEBSITES (with exact URL up to page)

1. https://www.digitalocean.com/community/tutorials/how-to-build-a-chatbot-with-flask
2. https://rasa.com/docs/rasa
3. https://cloud.google.com/dialogflow/docs

.

PROJECT SUMMARY

About Project

Title of the project	AI-Based Chatbot for Smart Assistance
Semester	8th
Members	2
Team Leader	Prince Rajak
Describe role of every member in the project	Neha Daryani: worked on the GUI and documentation Prince Rajak: worked on the backend development
What is the motivation for selecting this project?	The rapid growth of Artificial Intelligence (AI) and Natural Language Processing (NLP) has revolutionized human-computer interaction. Traditional chatbots lack adaptability and intelligence, making interactions inefficient. This project aims to develop an AI-driven chatbot that can understand user queries, generate NLP responses, integrate with APIs for real- time data, and perform automation tasks, making it a versatile virtual assistant applicable in multiple domains.
Project Type **(Desktop Application, Web Application, Mobile App, Web)**	Web Application

Tools &Technologies

Programming language used	Python
Compiler used **(with version)**	Python 3.9+
IDE used **(with version)**	VS Code (Version 1.70+)
FrontEndTechnologies **(with version, wherever Applicable)**	Python (version 3.13.2)
BackEndTechnologies **(with version, wherever applicable)**	Python (version 3.13.2)
Database used **(with version)**	MySQL (Version 8.0)

Software Design& Coding

Is prototype of the software developed?	Yes
SDLC model followed (Waterfall, Agile, Spiral etc.)	Agile
Why above SDLC model is followed?	Agile allows iterative development and continuous feedback, ensuring improvements based on user input and testing. It enhances adaptability to changing requirements, making it ideal for chatbot development.
Justify that the SDLC model mentioned above is followed in the project.	Regular updates and modifications based on feedback, Continuous testing ensure error-free chatbot performance, and flexibility in feature enhancements are supported.
Software Design approach followed (Functional or Object Oriented)	Object-Oriented Approach (OOP)
Name the diagrams developed (According to the Design approach followed)	Use Case Diagram
In case Object Oriented approach is followed, which of the OOPS principles are covered in design?	• Encapsulation • Inheritance • Polymorphism • Abstraction
No. of Tiers (example 3-tier)	3-tier
Total no. of front-end pages	-
Total no. of tables in the database	2
Databasein which Normal Form?	3rd Normal Form (3NF)
Are the entries in database encrypted?	Yes
Front end validations applied (Yes / No)	Yes
Session management done (in case of web applications)	Yes
Isapplicationbrowser compatible (in case of web applications)	Yes
Exception handling done (Yes / No)	Yes

Commenting done in code (Yes / No)	Yes
Naming convention followed (Yes / No)	Yes
What difficulties faced during deployment of project?	API integration and response time optimization. Handling speech-to-text and text-to-speech accuracy. Ensuring smooth UI/UX experience across different devices.
Total no. of Use-cases	1
Give titles of Use-cases	Use case diagram for AI-based chatbot

Project Requirements

MVC architecture followed (Yes / No)	Yes
If yes, write the name of MVCarchitecturefollowed (MVC-1, MVC-2)	-
Design Pattern used (Yes / No)	No
If yes, write thename of Design Pattern used	-
Interface type (CLI / GUI)	GUI (Graphical User Interface)
No. of Actors	2
Name of Actors	User, chatbot system
Total no. of Functional Requirements	8
List few important non-Functional Requirements	Storage, and Scalability, Security, Performance Optimization, Cross-Platform Compatibility, Usability

Testing

Which testing is performed? (Manual or Automation)	Manual
Is Beta testing done for this project?	Yes

Write a project narrative covering the points mentioned above.

The AI-Based Chatbot for Smart Assistance is an intelligent virtual assistant leveraging Natural Language Processing (NLP), speech processing, and AI-powered automation to enhance user interactions. It supports text and voice inputs, providing real-time, context-aware responses. Unlike rule-based chatbots, it employs machine learning to improve accuracy and efficiency. Key features include multimodal support, real-time information retrieval, and task automation. Designed for applications in healthcare, education, and customer service, the chatbot enhances accessibility and productivity. Future upgrades will include emotion AI and adaptive learning for personalized experiences, making it a robust and scalable solution for AI-driven digital assistance.

Neha Daryani 0187CS211107
Prince Rajak 0187CS211125

Guide Signature
(Prof.Deepti Jain)

APPENDIX 1
GLOSSARY OF TERMS

A	
Artificial Intelligence	AI refers to the development of computer systems that can perform tasks requiring human intelligence. These include learning, reasoning, and self-correction. AI is widely used in automation, robotics, and data analysis. It enables machines to make decisions based on data inputs.
C	
ChatBot	A chatbot is a software application designed to simulate human conversations through text or voice. It can assist users by answering queries, providing recommendations, and automating responses. Chatbots are widely used in customer service and virtual assistance. They improve efficiency by reducing the need for human intervention.
N	
Natural Language Processing (NLP)	NLP is a field of AI that enables machines to understand and interpret human language. It allows computers to process and analyze large amounts of natural language data. Applications of NLP include translation, sentiment analysis, and voice recognition. It helps in creating more interactive and human-like AI systems.
M	
Machine Learning (ML)	ML is a subset of AI that enables computers to learn from data and improve their performance without explicit programming. It uses algorithms to detect patterns and make predictions based on input data. ML is used in various fields, including finance, healthcare, and recommendation systems. It helps improve efficiency and accuracy in decision-making.

S	
Speech Processing	Speech processing involves analyzing and interpreting human speech for various applications. It includes speech recognition, synthesis, and enhancement technologies. This technology is used in virtual assistants, transcription services, and voice-controlled devices. It improves human-computer interactions by enabling voice-based commands.
T	
Task Automation	Task automation refers to the use of AI and software to perform repetitive tasks without human intervention. It increases efficiency, reduces errors, and saves time. Automation is widely used in industries such as manufacturing, customer support, and IT operations. It helps businesses streamline workflows and improve productivity.