# Chapter 1: Introduction

## 1.1 Background

I’ve always thought it’s pretty cool how we can talk to machines these days—like asking Siri about the weather or chatting with a bot to order pizza. Chatbots are everywhere, and they’re basically little AI helpers that talk back to us. I’m not aiming to build something as fancy as those, but I want to create my own simple version. Something that can have a basic chat, answer questions, and maybe even be useful for stuff like school. This project feels like a perfect way to get my hands dirty with AI and see what I can pull off. Plus, it’s a chance to learn some neat tech—like natural language processing (NLP)—that makes these bots tick.

## 1.2 Problem Statement

Here’s the thing: I’ve noticed that a lot of us students like me, or even teachers, could use a quick, easy way to get answers or help with small tasks. But the chatbots out there? They’re either super complicated to set up, cost money, or need you to be a tech wizard to use them. That’s a bummer, especially if you just want something simple, like a bot to answer basic school questions or give study tips. I want to fix that by making a chatbot that’s straightforward, free, and doesn’t need a PhD to figure out. It’s not about solving world hunger, but it’s a start to making tech a little handier for regular folks like us.

## 1.3 Objectives

### General Objective

### Build a simple AI chatbot that chats and helps with basic stuff.

### Make it feel natural and super easy to use.

### Get it working smoothly

### Specific Objectives

* **Add Some Smarts:** Use a bit of NLP so it’s not just parroting stuff but actually gets what you’re asking.
* **Keep It Simple:** Design it so anyone can use it, maybe even tweak it if they want, without needing to be a coder.
* **Check If It’s Good:** Test it out, see how it holds up, and ask some friends what they think hoping they don’t hate it!

## 1.4 Scope and Limitations

### Scope

* Text chats only (no voice yet).
* It comes with ready answers like a mini-FAQ bot for school, tips, or general info.
* Simple setup think typing in a console or a basic webpage.

### Limitations

* It won’t sound human or handle complex topics.
* Focus on one thing (like school) for now.
* No image, no other languages, no learning on its own

# Chapter 3: System Analysis

## 3.1 System Analysis

System analysis is the process of studying a system or a specific aspect of a system to understand its components, functions, interactions and requirements. There are various types of system analysis such as requirement analysis, feasibility analysis, system design analysis, risk analysis and others.

### 3.1.1 Requirement Analysis

Requirement analysis is a critical phase in system development that involves identifying, documenting, analysing and prioritizing the needs and expectations. This process typically includes gathering both functional and non-functional requirements.

### Functional Requirement

It described the specific functions that the system must perform.

* **Chatting Basics:** It’s going to take text input from users and give replies that make sense like a mini conversation buddy.
* **Answer Bank:** Needs a set of pre-loaded responses for common questions (e.g., school FAQs or study tips).
* **Understanding Words:** Should figure out simple questions by picking up key words or intent not super smart, just enough to get by.
* **Simple Interface:** It work through a basic setup maybe typing in a console or a no-frills webpage.

### B. Non-Functional Requirements

Non-functional requirements specify the quality attributes or constraints that the system must adhere to, such as performance, reliability, usability and security. These requirements define how the system should behave or perform rather that what it should do.

* Performance: Performance refers to how well the system performs in terms of speed, responsiveness, and efficiency. It includes factors like response time, throughput, and resource utilization.
* Reliability: The system should be available whenever needed, minimizing downtime. It should ensure the integrity and consistency of employee data, preventing data loss or corruption.
* Usability: Usability focuses on the ease of use and user experience of the system. It includes factors like user interface design, navigation, accessibility, and user support features. A usable system provides intuitive interfaces, clear instructions, and efficient workflows to enhance user satisfaction and productivity.

### 3.1.2 Feasibility Analysis

A feasibility study is a systematic analysis to determine the practicality and potential success of a proposed project or venture.

1. Technical Feasibility: Technical feasibility is concerned with the availability of hardware and software required for the development of the system. After the study we came to conclusion that we can proceed further with the tools and development environment chosen by us.
2. Operational Feasibility: Operational feasibility is all about problems that may arise during operations. There are two aspects related with the issue. What is the probability that the solution developed may not be put to use or may not work? What is the inclination of the management and end users towards the solution? Though, there is very least possibility of management being averse to the solution, there is significant probability that the end users may not be interested in using the solution due to lack of training, insight, etc.
3. Time Feasibility: Time feasibility refers to the assessment of whether a proposed project can be completed within a reasonable timeframe. After the study of the tasks involved in completing the project including requirement gathering, frontend development, backend development, coding, testing and others, we have concluded that the project timeline appears feasible. By carefully analysing tasks, resource availability & potential risks or delays, we can assess the feasibility of meeting the projects deadlines and make adjustments to the schedule as needed to ensure timely completion.
4. Cost Feasibility: The cost feasibility analysis for the **AI-Powered Knowledge Base (Deep** **Seek AI Integration)** involves evaluating expenses related to development, maintenance, and operational aspects. This includes estimating costs for software development, infrastructure, ongoing maintenance, integration, and any other associated expenses. After the study, it was determined that the AI-Powered Knowledge Base (Deep Seek AI Integration) is cost-feasible, with expenses well within the allocated budget.
5. Legal Feasibility:Legal feasibility entails ensuring compliance with relevant laws and regulations governing data privacy. Specifically, for the AI-Powered Knowledge Base (Deep Seek AI Integration), it involves adherence to data protection laws ensuring proper handling and security of user data. The system was found to be legally feasible since all user is allowed to use the system which will further enhance data privacy.

# CHAPTER 4: SYSTEM DESIGN

## 4.1 SDLC model

The Software Development Life Cycle (SDLC) model is a structured approach used by software development teams to design, develop, test, deploy, and maintain software systems. It encompasses a series of phases or stages, each with specific activities and deliverables. Common SDLC models include Waterfall, prototype & spiral model with its own set of principles, methodologies, and best practices tailored to different project requirements and organizational needs.

The phases in the Software Development Life Cycle (SDLC) typically include:

* 1. Planning: This phase involves defining the project scope, objectives, timelines, and resources required. It may also include feasibility studies and risk assessments to ensure the project's viability.
  2. Requirement Analysis: During this phase, the development team gathers and analyses requirements from stakeholders, such as users, customers, and business owners. The goal is to understand the needs and expectations of the software system to be developed.
  3. Design: In this phase, the system architecture and design specifications are created based on the requirements gathered. This includes defining the software components, data structures, interfaces, and algorithms to be used in the system.
  4. Implementation: Also known as coding or development, this phase involves translating the design specifications into actual code. Developers write, compile, and test the code to ensure it meets the requirements and design standards.
  5. Testing: Once the code is developed, it undergoes rigorous testing to identify and fix defects or bugs. Testing includes various techniques such as unit testing, integration testing, system testing, and UAT.
  6. Deployment: After successful testing, the software is deployed or released to the production environment. This phase involves installing the software on users' machines or servers and configuring it for use.
  7. Maintenance: The final phase involves maintaining and supporting the software post-deployment. This includes addressing any issues or bugs reported by users, making enhancements or updates as needed, and ensuring the software remains functional and efficient over time.

These phases may vary slightly depending on the specific SDLC model or methodology being used, but they generally encompass the key activities involved in software development projects.

## 4.2 Selected model

Our project is aimed to develop AI-Powered Knowledge Base (Deep Seek AI Integration)using Python programming language. After going through all the SDLC models we concluded on using waterfall model. The Waterfall Model is a linear and sequential approach to software development, where each phase must be completed before moving on to the next. It typically consists of distinct phases such as requirements gathering, design, implementation, testing, deployment, and maintenance. Once a phase is completed, the process moves to the next phase in a cascading manner, with minimal opportunity for revisiting earlier stages. It's characterized by its rigidity and emphasis on extensive upfront planning and documentation.

A diagram of a process

AI-generated content may be incorrect.

***Figure 1: Waterfall Model***

Since the requirements & objective of our project is clear to us, we selected waterfall model because it suits the best for completion of our mini project. Following the waterfall model, the project will be progressed through sequential phases including requirement gathering, designing, implementation, testing, deployment and maintenance. Extensive documentation will be maintained at each stage to ensure clarity and facilitate future maintenance. Clarity in requirements, fixed scope and objective are some of the reasons behind the selection of this model.

## 4.3 Context diagram

A white circle with black text

AI-generated content may be incorrect.A context diagram is a high-level visual representation that illustrates the scope and boundaries of a system or process within its environment. It provides an overview of the interactions between the system being analyzed and its external entities, such as users, other systems, or external stakeholders. Context diagrams help stakeholders understand the context in which the system operates and facilitate discussions about its requirements, interfaces, and dependencies.

***Figure 2: Context Diagram***

## 4.4 Data Flow Diagram

DFD stands for Data Flow Diagram. It's a graphical representation that illustrates how data flows through a system or process. DFDs consist of processes, data stores, data flows, and external entities. Processes represent activities or transformations that occur within the system, data stores depict where data is stored, data flows show the movement of data between processes and data stores, and external entities represent sources or destinations of data outside the system. The level 1 DFD of our proposed system is as shown below:

A black background with white rectangular objects

AI-generated content may be incorrect.

***Figure 3: Data Flow Diagram***

## 4.5 ER-Diagram

An ER diagram is a visual representation that depicts the relationships among entities within a database. ER diagrams help in understanding the structure of a database and are commonly used during the database design phase to model the relationships between different entities and their attributes. The ER diagram of our purposed system which will be further modified according to our requirements.

## 4.6 Use Case Diagram

A Use Case Diagram is a type of behavioral diagram in Unified Modelling Language (UML) that illustrates the interactions between actors (users or external systems) and a system to accomplish specific goals or tasks. It shows the functionality of a system from the perspective of its users and helps to understand how users interact with the system.

# CHAPTER 5: IMPLEMENTATION & TESTING

## 5.1 Tools used

## 5.2 Gantt chart

****We have outlined the timeline for the implementation of project below using a Gantt chart. This chart illustrates the major tasks, their dependencies and the estimated duration for each task.

***Figure 6: Gantt chart***

## 5.3 Testing

Software testing is a process of analyzing an application's functionality as per the requirements. If we want to ensure that our software is bug-free or stable, we must perform the various types of software testing because testing is the only method that makes our application bug free. Here’s a brief overview of various types of software testing:

1. **Functional Testing:** In functional testing, all the components are tested by giving the value, defining the output, and validating the actual output with the expected value. Functional testing. In functional testing, all the components are tested by giving the value, defining the output, and validating the actual output with the expected value.
2. Unit Testing: Unit testing is the first level of functional testing in order to test any software. In this, the test engineer will test the module of an application independently or test all the module functionality is called unit testing.
3. Integration Testing: Once we are successfully implementing the unit testing, we will go integration testing. It is the second level of functional testing, where we test the data flow between dependent modules or interface between two features is called integration testing.
4. System Testing: Whenever we are done with the unit and integration testing, we can proceed with the system testing. In system testing, the test environment is parallel to the production environment. In this type of testing, we will undergo each attribute of the software and test if the end feature works according to the business requirement. And analysis the software product as a complete system.
5. **Non-Functional Testing:** Non-functional testing is a type of software testing that focuses on the attributes of a system that do not relate to specific behaviors’ or functions. Instead, it assesses qualities such as performance, reliability, scalability, usability, security, and compatibility. Non-functional testing helps evaluate how well a system meets its requirements in terms of these attributes and ensures that it performs satisfactorily under various conditions beyond functional correctness.

For the completion of our mini project we will be using functional testing method as it ensures that software meets specified requirements by testing individual functions or features, detecting bugs early, and ensuring user satisfaction and overall quality.

# CHAPTER 6: EXPECTED OUTCOME

## 6.1 Final System Expectation

By the end of this project, We aiming to create a simple AI chatbot built in Python, packed with some cool features like asking query, modify query and get responses. It’s not going to be a world-changer, but it’ll be a handy little buddy for chatting and helping out with basic stuff. Here’s what I expect this chatbot to deliver by the time I’m done:

* **Chat Ready:** A working chatbot that can talk back to users, answering simple questions with replies we have set up like a mini-FAQ helper.
* **Editable Answers:** User can Edit answer given by chatbot.
* **View & Search:** Let users peek at the list of responses it knows and search for specific ones to see what’s in its brain.
* **User-Friendly :** Easy to use, quick to respond.

# CHAPTER 7: CONCLUSION & DISCUSSION

## 7.1 Conclusion

After sorting out what our chatbot’s can do all about and what it needs to do, I’m super excited to bring it to life with Python. It’ll handle stuff like adding, tweaking, and searching responses, making chats quick and easy while keeping answers spot-on. With some basic user checks, it’ll stay secure too.

## 7.2 Future Enhancements

* **Smarter Chats:** Maybe add some real NLP—like Java libraries or a lightweight AI model—so it understands questions better, not just keyword matches.
* **Voice Mode:** How cool would it be if it could talk? Adding voice input and output could make it feel more alive.
* **Learn on the Fly:** Teach it to pick up new responses from users over time instead of manually adding everything.
* **More Topics:** Expand it beyond one area (like school) to handle all sorts of questions make it to take all inputs.
* **Fancy Looks:** Spruce up the interface with some graphics or a friendlier design less geeky, more fun.