

Text Based Question Answering System

Ms. Komal Mote-Bagal¹ and, Mr. Ajinkya Patil², Mr. Sagar Chavan³

^{1,2,3}Member, Anantrao Pawar College of Engineering and research, Pune

DOI: <https://doi.org/10.51244/IJRSI.2026.1308000117>

Received: 24 August 2026; Accepted: 29 August 2026; Published: 08 September 2026

ABSTRACT

The “Question Answering System Based on Paragraph” is a Python-based web application designed to provide users with a seamless experience in extracting answers from textual content. Built with Flask, the application allows users to securely log in, paste a paragraph of text into an input field, and ask questions about the content. By leveraging the Gemini/OpenAI API, the system comprehends the provided paragraph’s context, enabling it to generate accurate and context-aware answers to user queries. This application is particularly useful for students, researchers, and professionals who need quick insights from large texts without manually searching for information. The project also includes basic user authentication through login and registration features, ensuring a personalized experience. This innovative approach combines natural language processing with advanced AI capabilities to offer an efficient and user- friendly solution for text-based question answering.

Keywords— Python, Flask, web application, login, registration, paragraph input, question answering, OpenAI API, natural language processing, text comprehension, user authentication, text-based question answering, data input, NLP, AI-powered system, user queries, web development.

INTRODUCTION

The rapid advancements in artificial intelligence and natural language processing have paved the way for intelligent systems capable of understanding and responding to human queries in a variety of contexts. One practical application of these advancements is the development of question-answering systems, which enable users to ask questions and receive precise answers based on a given piece of text. This project, titled “Question Answering System Based on Paragraph” aims to harness this capability by building a web application that facilitates efficient and accurate extraction of information from user-provided text. This system is built using Python and the Flask web framework, ensuring a lightweight, responsive, and easily deployable solution. At its core, the application enables users to securely log in, paste paragraphs of text, and ask questions related to the provided content.

Leveraging the powerful language understanding capabilities of the Gemini/OpenAI API, the system can analyze and interpret the context of the paragraph, allowing it to provide relevant and accurate responses to user queries. The primary purpose of this application is to enhance productivity by allowing users to interact with text more dynamically and intuitively. Instead of manually searching through paragraphs for specific information, users can quickly obtain answers to targeted questions, making this system ideal for students, researchers, and professionals who need to analyze and understand large volumes of information. Overall, this project exemplifies the application of AI-powered language models in creating user- centric, intelligent solutions that improve accessibility to information and streamline information retrieval processes. Through this project, users can experience the convenience of conversational AI in extracting knowledge from textual data, reflecting a significant step forward in the integration of natural language.

LITERATURE REVIEW

In recent years, the field of natural language processing (NLP) has advanced significantly, especially in the development of question-answering (QA) systems. These systems have evolved from simple keyword-based

retrieval models to sophisticated deep learning-based models capable of understanding context and providing accurate answers from textual data. This literature survey explores the key developments and methodologies in QA systems that have inspired and influenced the design of the “Question Answering System Based on Paragraph.”

Traditional Question-Answering Systems Early QA systems relied on keyword-matching and information retrieval (IR) techniques, such as TF-IDF (Term Frequency-Inverse Document Frequency), to extract answers from text. While effective for basic searches, these systems lacked the ability to

understand the context, leading to inaccuracies when questions required deeper comprehension. Examples include classic IR systems like Apache Lucene and Elastic-Search, which provide keyword-based matching but struggle with nuanced, context-sensitive queries.

Introduction of NLP and Machine Learning in QA Systems The application of NLP and machine learning techniques introduced significant improvements in QA systems. Models such as Word2Vec and GloVe allowed for semantic word embedding, enabling systems to recognize relationships between words. This era also saw the rise of support vector machines (SVMs) and logistic regression for simple QA tasks. However, these methods were limited in handling complex language structures and required large datasets to achieve moderate accuracy.

Deep Learning and Neural Network Models for QA The advent of deep learning revolutionized QA systems by allowing them to process language more naturally and understand context. Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks were utilized to build sequence-based models that improved text comprehension. Attention mechanisms, later enhanced by self-attention, allowed models to focus on relevant portions of text, making it possible to answer questions more accurately. The Stanford Question Answering Dataset (SQuAD) benchmark was introduced to evaluate these models, setting a new standard for QA performance.

Transformer Models and BERT The introduction of transformer-based models, particularly BERT (Bidirectional Encoder Representations from Transformers) by Google, marked a breakthrough in contextual understanding. BERT and its successors (e.g., RoBERTa, ALBERT) enabled bi-directional context comprehension, where models consider the surrounding words in both directions, resulting in highly accurate responses. BERT-based models have been widely used in QA systems due to their ability to handle complex, natural language queries with remarkable accuracy.

GPT and OpenAI’s Language Models OpenAI’s Generative Pre-trained Transformer (GPT) models, including GPT-2, GPT-3, and subsequent models, further advanced QA systems by introducing large-scale, unsupervised language models capable of generating humanlike responses. These models can understand and generate language based on vast training data, making them ideal for applications requiring nuanced and context-aware answers. The Gemini/OpenAI API leverages such models to deliver high-quality answers in real-time, making it a valuable resource for question-answering applications.

Applications of Modern QA Systems Modern QA systems are now applied across various fields, from customer support to education and healthcare. For example, IBM Watson’s QA system has been used in healthcare to assist doctors in diagnosing diseases. In the educational domain, intelligent tutoring systems use QA capabilities to help students learn more interactively. These applications demonstrate the versatility and importance of QA technology in facilitating access to information and enhancing user experiences.

METHODOLOGY

The methodology for developing the Question Answering System Based on Paragraph can be broken down into several key stages, detailing the flow of data and the integration of various technologies. Below is a step-by-step outline of the methodology:

Requirement Analysis & Design Objective: To define the system's functional and non-functional requirements, including login functionality, input text processing, and question answering. **Design:** The system is designed to have a user- friendly interface where users can register/login, paste text paragraphs, and ask questions. **Flask** is used to structure the backend of the web application. **Frontend Development** HTML/CSS/JavaScript is used for the frontend. **User Interface (UI):** Create simple and intuitive web pages using HTML, CSS, and JavaScript for the user interface. Pages include registration, login, a text input field for the paragraph, and an area for asking questions. A response area is included for displaying the answers generated by the system. **Form Handling:** Develop forms for user registration and login using Flask's form handling features, ensuring data validation and secure authentication.

Backend Development with Flask Web Framework: Flask is selected for backend development due to its lightweight nature and ease of handling HTTP requests. **User Authentication:** Implement user registration and login functionality using Flask. User credentials are securely stored in a database (SQLite or MySQL) with encryption for passwords. **Session Management:** Handle user sessions after login to ensure that only authenticated users can access the question-answering functionality. **Text Input & Processing Paragraph Input:** After a successful login, the user can input a paragraph into a text field. **API Integration:** The system sends the input paragraph to the Gemini/OpenAI API for text processing. The API comprehends the content of the paragraph using advanced natural language processing (NLP) techniques. The paragraph is stored temporarily in the session for reference in the question- answering phase.

Question Answering Question Submission: Once the paragraph is understood by the system, the user submits a question related to that paragraph. **Querying the API:** The system uses the OpenAI/Gemini API to interpret the question and retrieve an appropriate response based on the content of the paragraph. The API processes the user question within the context of the previously provided text, utilizing NLP models to generate an accurate answer. **Response Handling:** The API's response is sent back to the Flask application and displayed in the frontend for the user to view.

Integration of Natural Language Processing (NLP) API Utilization: The OpenAI/Gemini API is key in this project for both understanding the input text (paragraph) and generating context-aware answers to the user's questions. **NLP Tasks:** These tasks include: **Context Understanding:** Comprehending the meaning of the paragraph. **Question Understanding:** Interpreting the user's question. **Answer Generation:** Generating an accurate and contextually appropriate answer from the provided paragraph.

Data Storage Database Management: Use a database (e.g., SQLite or MySQL) to store user details, including registration information and login credentials. Optionally, user input paragraphs and their corresponding questions/answers can be logged for further analysis or improvement of the system. **Testing and Evaluation Unit Testing:** Test each component, such as user authentication, API communication, and response handling, to ensure functionality. **Integration Testing:** Verify the seamless integration between the frontend, Flask backend, and OpenAI/Gemini API. **User Testing:** Conduct user acceptance testing to ensure that the system performs as expected and is user-friendly.

Deployment Server Setup: Deploy the Flask application on a web server, such as AWS, Heroku, or Google Cloud Platform (GCP). **API Key Management:** Ensure secure management of API keys (Gemini/OpenAI) during deployment. **Production Monitoring:** Implement monitoring tools to track the application's performance and user activity.

Maintenance and Future Improvements Regularly update the system with new API versions or improved NLP models for better question answering capabilities. Enhance the user interface and potentially introduce features like multi-language support, saving user sessions, or exporting answers.

CONCLUSION

The Question Answering System Based on Paragraph is a web- based application that allows users to input a paragraph and ask questions related to the content. By using advanced natural language processing (NLP)

techniques, particularly through APIs like Gemini/OpenAI, the system processes the paragraph and provides accurate, context-aware answers to the user's questions. This system simplifies the information retrieval process, offering quick responses and enhancing user productivity. It finds applications in various fields such as education, research, customer support, and business, where it helps users efficiently extract relevant information from large texts. However, its effectiveness depends on the quality of the input data and may face challenges with complex or specialized queries.

REFERENCES

1. OpenAI. (2023). OpenAI API Documentation. Retrieved from <https://beta.openai.com/docs>.
2. Flask. (2023). Flask: Web Development Documentation. Retrieved from <https://flask.palletsprojects.com/>.
3. Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media.
4. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805.
5. Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). SQuAD: 100,000+ Questions for Machine Comprehension of Text. arXiv:1606.05250.
6. Ferrucci, D., et al. (2013). Watson: Beyond Jeopardy! Artificial Intelligence, 199, 93-105.
7. de Milato, T., et al. (2020). Haystack: A Framework for Question Answering at Scale. Retrieved from <https://haystack.deepset.ai/>.
8. Google Cloud. (2023). Dialogflow Documentation. Retrieved from <https://cloud.google.com/dialogflow/docs>.
9. Liu, Y., et al. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692.
10. Mishra, A., & Jain, S. K. (2016). A Survey on Question Answering Systems with Classification. Journal of King Saud University-Computer and Information Sciences, 28(3), 345