REPORT

ABSTRACT:

In recent years, chatbots have emerged as a revolutionary tool in various sectors, leveraging advances in artificial intelligence (AI) and natural language processing (NLP). This project presents the development of a chatbot designed specifically for first and second-year biology intermediate students at our college. The motivation behind this project stems from the need to provide students with a reliable and accessible platform for addressing their academic queries. The chatbot aims to facilitate students in resolving their doubts efficiently, thereby enhancing their learning experience and academic performance. Leveraging cutting-edge generative AI technologies, including large language models (LLMs) and Retrieval-Augmented Generation (RAG), by using RAG It retrieves relevant documents and pieces of information from a vast corpus and uses this data to generate precise and contextually appropriate responses and also it enhances the chatbot's capabilities. To manage and access the vast amount of academic content, MongoDB Atlas is employed as the knowledge base. MongoDB Atlas provides a scalable and flexible platform to store and retrieve information efficiently. The implications of this project for the college students are substantial. By providing an accessible and efficient academic support tool the students can self-directed to learning, enabling students to explore and understand topics at their own pace.

The value of this project lies in its potential to transform the academic support landscape within the college. It serves as a scalable solution that can be extended to other subjects and departments, offering a versatile tool for academic assistance. Furthermore, the integration of advanced AI technologies sets a precedent for future educational innovations, positioning the college at the forefront of educational technology adoption.

INTRODUCTION:

In the realm of modern education, the integration of advanced technological solutions has become indispensable for fostering an enriching learning environment. One such innovation is the deployment of chatbots, which harness the power of artificial intelligence (AI) to provide instant and tailored support to students. This project undertakes the development of a sophisticated chatbot tailored specifically for first and second-year biology students at our institution, aimed at addressing their academic queries with unparalleled efficiency. The motivation for this project is to overcome the limitations of traditional educational support, which often cannot provide timely assistance to students outside of classroom hours. By using advanced AI technologies, including Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) tools, this chatbot is designed to give accurate and contextually relevant answers, thus improving the learning experience. The objectives of this project are clear and ambitious. We aim to develop a robust, scalable platform that can handle a diverse array of queries, ranging from simple factual questions to complex conceptual clarifications.

The chatbot is trained on a comprehensive dataset encompassing NCERT books and supplementary academic resources, ensuring an extensive coverage of the biology curriculum. Additionally, the integration of MongoDB Atlas as the knowledge base provides a scalable and flexible framework for efficient data retrieval and management. This project not only aims to improve the academic support available to biology students but also serves as a model for future educational innovations. The implications of this project extend beyond immediate academic benefits, positioning our college at the forefront of educational technology and innovation.

In summary, this project demonstrates the potential of AI in education, showing how chatbots can create a more responsive and effective learning environment. The following sections of this report will cover the literature review, detailed methodology, implementation process, and key results, leading to a discussion of the project's broader impacts and future possibilities.

Literature Review:

The field of artificial intelligence (AI) has seen significant advancements in recent years, with applications spanning across various domains, including education. Chatbots, in particular, have garnered considerable attention for their potential to enhance educational experiences by providing instant and personalized support to students. This literature review explores the existing research and developments in the areas of chatbots, generative AI technologies, and their applications in educational contexts, with a specific focus on Large Language Models (LLMs), Retrieval-Augmented Generation (RAG) tools, and the utilization of MongoDB Atlas as a knowledge base.

Chatbots in Education

The deployment of chatbots in educational settings is not a novel concept; however, recent technological advancements have significantly augmented their capabilities and efficacy. Chatbots have been utilized to assist with administrative tasks, provide language learning support, and facilitate tutoring in various subjects. According to Winkler and Sollner (2018), chatbots can enhance student engagement and motivation by providing immediate feedback and fostering interactive learning environments. Furthermore, studies by Hill et al. (2015) highlight the effectiveness of chatbots in offering personalized learning experiences tailored to individual student needs.

Large Language Models (LLMs)

Large Language Models, such as OpenAI's GPT-3, have revolutionized the field of natural language processing (NLP). These models are trained on vast datasets and possess the ability to generate human-like text based on given prompts. Research by Brown et al. (2020) demonstrates the versatility of LLMs in performing a wide array of language tasks, from translation to summarization and question answering. In the context of education, LLMs have shown promise in generating contextually relevant responses and explanations, thereby serving as effective tools for academic support.

Retrieval-Augmented Generation (RAG) Tools

Retrieval-Augmented Generation (RAG) represents a sophisticated approach that combines the strengths of information retrieval and generative modelling. RAG models, as described by Lewis et al. (2020), first retrieve relevant documents from a corpus based on the input query and then generate a response using this retrieved information. This dual mechanism enhances the accuracy and contextual relevance of the responses, making RAG tools particularly valuable in educational applications where precise and contextually appropriate answers are crucial.

MongoDB Atlas in Educational Applications

The use of robust database solutions is critical for managing the vast amounts of information required for effective educational support systems. MongoDB Atlas database offers scalability, flexibility, and efficient data retrieval capabilities. As noted by Chodorow (2013), MongoDB's schema-less architecture and powerful query capabilities make it an ideal choice for dynamic and data-intensive applications. In educational contexts, MongoDB Atlas has been employed to store and manage diverse educational content, enabling quick and reliable access to information.

Integration of AI Technologies in Educational Chatbots

The integration of LLMs, RAG tools, and MongoDB Atlas in educational chatbots represents a significant advancement in the field. Studies by Xie et al. (2021) and Rajpurkar et al. (2018) underscore the potential of these technologies to provide accurate and contextually enriched academic support. The combination of advanced AI models with efficient data management systems enhances the chatbot's ability to deliver timely and relevant assistance to students, thereby improving their learning outcomes.

Implications and Future Directions

The existing literature underscores the transformative potential of AI-driven educational chatbots. The integration of LLMs and RAG tools, supported by robust database solutions like MongoDB Atlas, can significantly augment the capabilities of educational support systems. Future research should focus on refining these technologies to further enhance their accuracy and contextual understanding. Additionally, exploring the application of these advancements across various subjects and educational levels can provide deeper insights into their efficacy and broader impact.

Module 1: Preprocessing the document in the backend

1. Document and Data Preprocessing

- **Documents**: The process begins with the collection of documents, which contain the relevant information and content needed for the chatbot.
- **Pre-process**: These documents undergo a preprocessing stage where the text is cleaned, normalized, and formatted for further analysis. This may include steps such as removing special characters, correcting types, and tokenizing text into manageable units.

Module 2: Converting text into vectors

Embedding Generation

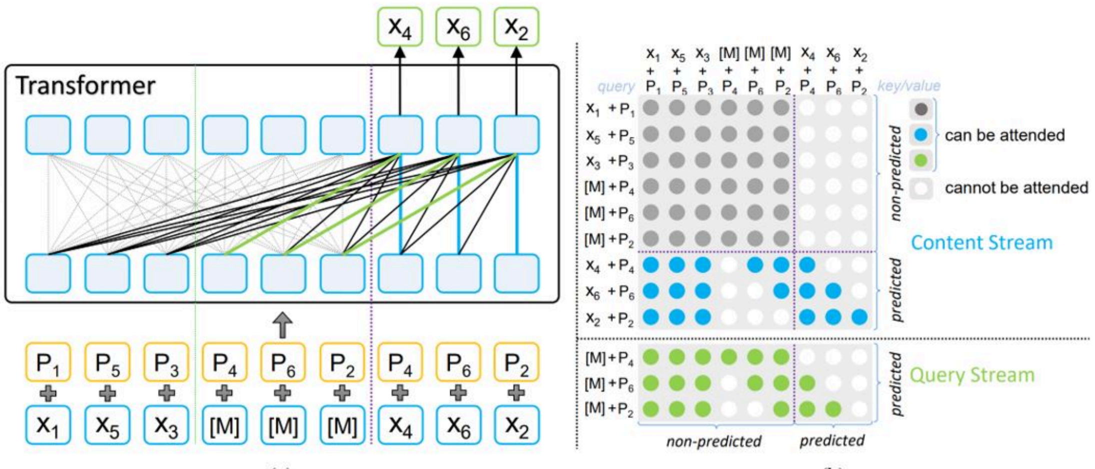
- Sentence Transformers convert sentences and paragraphs into dense vector embeddings. These embeddings are high-dimensional vectors that encode the semantic meaning of the text, making them useful for tasks that require understanding the nuances of language.
- Sentence Transformers come with several pre-trained models optimized for different tasks and datasets. The all-mpnet-base-v2 model, for instance, is known for its robust performance in generating high-quality embeddings for various applications

Document Embedding Generation:

- All documents and relevant texts are preprocessed and passed through the all-mpnet-base-v2 model to generate dense vector embeddings. These embeddings are stored in a vector database for efficient retrieval.
- After processing the text through transformer layers, Sentence Transformers use pooling layers to aggregate the contextual information into a single fixed-size embedding. Common pooling strategies include mean-pooling, max-pooling, or using the CLS token's representation.
- The output is a document embedding, represented as a vector (e.g., <0.1, 0.4, -0.2, ...>), which can be efficiently stored and retrieved.

MPNet Embedding Architecture:

- This is a pre-trained deep learning model used to create numerical representations (embeddings) of text. It analyzes the context of words and captures their semantic meaning and relationships.
- In RAG, the MPNet model takes queries and document chunks as input and generates dense vectors for each. These vectors represent the meaning of the text in a high-dimensional space.



(a) (b)

Research paper: https://arxiv.org/pdf/2004.09297
Code implementation: https://github.com/microsoft/MPNet
Fine-tuning MPNet:
https://github.com/microsoft/MPNet/blob/master/MPNet/README.glue.md

In the context of the MPNet architecture, the symbols x1, x2, x3, x4, x5, and x6 represent individual tokens or words within an input sentence. These tokens are the fundamental units of text that the model processes. The symbols p1, p2, p3, p4, p5, and p6 represent the positional embeddings associated with each token.Positional embeddings provide information about the location of each token in the sentence, which is crucial for the model to understand the order and relationships between words

Flow Depicting Arrow Marks:

The arrow marks in the MPNet architecture diagram illustrate the flow of information within the model during the training process. Here's a breakdown of the flow:

1. Input: The input sentence is tokenized (split into individual words or subwords) and converted into embeddings (x1, x2,..., xn) along with their corresponding positional embeddings (p1, p2..... pn).

2. Permutation: The order of the tokens is randomly permuted (shuffled). This is a key aspect of MPNet, as it helps the model learn bidirectional relationships between words.

3. Masking: Some of the tokens in the permuted sequence are masked, meaning they are replaced with a special "[MASK]" token.

4. Two-Stream Processing: The masked sequence is then processed by two parallel streams:

* **Content Stream:** This stream receives the entire permuted sequence, including both masked and non-masked tokens. It processes the full context of the sentence.

* **Query Stream:** This stream receives the permuted sequence with masked tokens. It focuses on predicting the masked tokens based on the context provided by the content stream.

5. Self-Attention: Both streams use self-attention mechanisms to weigh the importance of different tokens in the sequence. The query stream attends to the non-masked tokens in the content stream to gather information for predicting the masked tokens.

6. Position Compensation: The query stream also incorporates positional information to compensate for the fact that it cannot see the masked tokens directly. This helps maintain consistency between pre-training and fine-tuning.

7. Prediction: The query stream produces output probabilities for each possible token that could fill the masked positions. The model is trained to maximize the probability of predicting the correct masked tokens.

In essence, the arrows represent the flow of token and positional embeddings through the MPNet model, highlighting the interaction between the content and query streams, the role of self-attention, and the incorporation of positional information for accurate language understanding

Module 3: Vector database storage

Vector databases are designed to handle high-dimensional vector data, enabling efficient storage, retrieval, and similarity search. MongoDB Atlas, the cloud-based managed database service by MongoDB, has introduced features that support vector data, making it suitable for applications that require handling embeddings and performing similarity searches.

- MongoDB allows the creation of specialized indexes to support fast vector search queries. These indexes can be used to accelerate the performance of similarity search operations.
- Before storing vectors, the data (e.g., documents, images) is preprocessed to extract relevant features or embeddings using models like Sentence Transformers. The output is a high-dimensional vector that represents the semantic content of the data.

Document Structure:

- Each document in MongoDB Atlas can include the original data (e.g., text, metadata) and the corresponding vector embedding.
- These documents are inserted into a MongoDB collection. MongoDB's flexible schema allows storing vectors as arrays within the document, ensuring all relevant information is stored together.

Module 4: User Query Submission

1.User Interface Interaction

The user interacts with a web-based interface that allows for seamless query submission. The interface is designed to be intuitive and user-friendly, ensuring that students can easily navigate and utilize the chatbot's features.

- **Text Input Field**: A clearly marked text box where users can type their questions. This field supports natural language input, enabling users to phrase their questions as they would in a conversation.

2.Query Submission

- Once the user has entered their query or uploaded an image, they initiate the submission process by clicking the "Submit" button. The interface captures the query data and sends it to the backend for processing.

Module 5: Preprocessing the query

Data Handling and Preprocessing

Upon receiving the query, the backend performs initial data handling and preprocessing to ensure the input is in a suitable format for further processing.

- **Text Queries**: The text input is cleaned and tokenized. Cleaning involves removing unnecessary whitespace, correcting common types, and normalizing text to a consistent format. Tokenization breaks the text into smaller units (tokens), which are easier for the AI models to process.

Module 6: Converting text into vectors

Query Embedding Generation:

- When a user submits a query, it is similarly preprocessed and transformed into an embedding using the same all-mpnet-base-v2 model. This ensures that both document and query embeddings are in the same vector space, facilitating accurate similarity comparisons.

Module 7: Retrieving Relevant Information

- The query embedding is used to search the vector database for relevant documents. The database retrieves documents whose embeddings are most similar to the query embedding, based on cosine similarity.

Semantic and Similarity Techniques

1. **Semantic Embeddings**:
   - Semantic embeddings are dense vector representations of text (or other data) that capture the meaning and context of the content. These embeddings are generated using models like Sentence Transformers, which transform input text into high-dimensional vectors.
   - The core idea is that semantically similar pieces of text will have similar embeddings, meaning their vectors will be close to each other in the embedding space.
2. **Similarity Search**:
   - Similarity search involves comparing the query embedding with embeddings stored in the database to find the closest matches. The measure of "closeness" or similarity can be computed using various mathematical techniques, with cosine similarity being one of the most common.

Cosine Similarity:

- This is a mathematical metric used to compare the direction of two vectors. It measures how closely aligned the vectors are in the high-dimensional space.
- In RAG, cosine similarity is applied to the dense vectors generated by the MPNet model for the query and each document chunk.
- A higher cosine similarity score indicates the document chunk's vector is closer in direction to the query vector, suggesting greater semantic similarity between the document and the user's query.
- Cosine similarity is a metric used to measure how similar two vectors are. It calculates the cosine of the angle between two vectors in a multidimensional space. The cosine similarity value ranges from -1 to 1, where 1 indicates that the vectors are identical, 0 indicates orthogonality (no similarity), and -1 indicates diametric opposition.

Dense Retriever:

- This is a component in RAG responsible for retrieving relevant information based on the user's query.
- It leverages the MPNet model and cosine similarity to identify document chunks that are semantically similar to the query.
- The dense retriever ranks the document chunks based on their cosine similarity scores with the query. The top-ranked chunks are considered the most relevant and are then provided to the large language model (LLM) for the generation process.

🎬 **Using Dot Product and Magnitudes**:

The standard formula for cosine similarity is given by:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

Module 8: Create Contextualized Prompt

The retrieved relevant text and features are combined to create a contextualized prompt. This prompt integrates the user's query with the most pertinent information from the database, ensuring the large language model (LLM) has the context needed to generate an accurate and relevant response.

Fine-Tuning the Prompt

- **Refinement**: Iteratively refine the prompt to ensure clarity and relevance. This might involve adjusting the level of detail in the context or rephrasing the query for better understanding.
- **Templates**: Use predefined templates to maintain consistency in prompt design. Templates can include placeholders for the query and context, ensuring a uniform approach. Here we give custom prompt about it's ethics and if user ask query outbound of its domain it simply give the response like please ask me questions with in biology related domain related queries.

Module 9: Feeding the Prompt to the LLM

- Input the contextualized prompt into the LLM (e.g., LLaMA) to generate a response.
- LLaMA is a family of large language models developed by Meta AI. It's designed to perform a wide range of natural language processing (NLP) tasks, including text generation, question answering, translation, and more.

Integrating LLama for Question Answering:

- In this project, LLaMA is used as the core engine for generating responses to student queries. When a student asks a question, LLaMA processes the contextualized prompt created by the system and generates a detailed and accurate answer.
- The use of LLaMA ensures that responses are coherent, contextually relevant, and informative, enhancing the learning experience for biology students.

Handling Complex Queries:

- LLaMA's ability to understand and generate text based on complex prompts allows it to handle intricate and multi-faceted questions from students. This is particularly useful in subjects like biology, where concepts can be interrelated and require detailed explanations.

Model Architecture:

- LLaMA uses a transformer-based architecture, characterized by its self-attention mechanism that enables the model to weigh the importance of different words in a sentence, capturing the context effectively.
- The architecture includes multiple layers of attention and feedforward networks, allowing it to build complex representations of text.

Flow among them:

1. User submits a **query**.
2. **MPNet model** generates dense vector representations for both the **query** and each **document chunk** in the retrieval database.
3. **Cosine similarity** is calculated between the query vector and each document chunk vector.
4. The **dense retriever** ranks the document chunks based on their cosine similarity scores with the query.
5. The **top-ranked document chunks** are provided to the **LLM** as context for generating a response.

Module 10: Response to user query

The generated response is sent back to the user, completing the interaction cycle. The user receives a detailed answer to their query, supported by the integrated information from the knowledge base and the LLM's generative capabilities.

Scenario:

- **User Query: "What are the steps involved in Respiration?"**
- **Document Chunk: The provided text snippet from the database.**

(Respiration involves the following steps: (i) Breathing or pulmonary ventilation by which atmospheric air is drawn in and CO2 rich alveolar air is released out. (ii) (iii) Diffusion of gases (O2 and CO2) across alveolar membrane. Transport of gases by the blood. (iv) Diffusion of O2 and CO2 between blood and tissues. (v) Utilisation of O2 by the cells for catabolic reactions and resultant release of CO2 (cellular respiration as dealt in the Chapter 12).)

1. Manual Vector Creation (Simulation):
Since we're not using an actual MPNet model, let's create simplified vectors to represent the query and the document chunk. These vectors will capture the key concepts but won't have the same complexity as real MPNet embeddings.

- Query Vector (x):
  - We can represent the main concepts in the query as elements in the vector.
  - Here, "steps," "respiration," and "involved" are key terms.
  - We can assign a weight to each concept based on its importance in the query.

x = [3, 2, 1] # Weights: steps (3), respiration (2), involved (1)

- Document Chunk Vector (y):
  - We can identify key concepts in the document chunk.
  - Here, "respiration," "steps," "breathing," "diffusion," "gases," "blood," "cells," and "CO2" are important terms.

y = [2, 3, 1, 2, 1, 1, 1, 1] # Weights: respiration (2), steps (3), etc.

2. Cosine Similarity Calculation (Manual):

We can now calculate the cosine similarity between the query vector (x) and the document chunk vector (y) using the formula:

```
Cosine Similarity=>k(x,y) = (x . y) / ||x|| * ||y||
```

where:

- `.` denotes the dot product of the vectors.
- `||x||` and `||y||` represent the magnitudes (lengths) of vectors x and y, respectively.

Calculation:

- Dot product (x . y) = (3 * 2) + (2 * 3) + (1 * 1) = 13
- Magnitude of x (||x||) = sqrt(3^2 + 2^2 + 1^2) = sqrt(14)
- Magnitude of y (||y||) = sqrt(2^2 + 3^2 + 1^2 + 2^2 + 1^2 + 1^2 + 1^2 + 1^2) = sqrt(18)
- Cosine Similarity = 13 / (sqrt(14) * sqrt(18)) ≈ 0.82

3. Interpretation:
The calculated cosine similarity score (0.82) is relatively high, indicating that the document chunk is semantically similar to the user query. This suggests the document chunk likely describes the steps involved in respiration.

Important Note:
In reality, MPNet models generate much more complex and nuanced vectors with hundreds or thousands of dimensions. These vectors capture the relationships between words and their context, leading to more accurate semantic similarity comparisons. Our manual vectors provide a simplified understanding for demonstration purposes.

4. Retrieval:
If we had multiple document chunks in the database, we could calculate the cosine similarity for each chunk with the query vector. The document chunk with the highest cosine similarity score would be considered the most relevant to the user's query.

Conclusion:
This example demonstrates how cosine similarity can be used with simplified vector representations to identify relevant document chunks based on their semantic similarity to the user's query. In a real RAG system, pre-trained models like MPNet would provide more sophisticated vector representations, leading to more accurate retrieval results.