ArXiv AI Assistant

Varun Nagpal
BTech, 3rd Year
Data Science and Artificial Intelligence, IIT Guwahati

I. Introduction and Motivation

In the realm of academic research, scholars often face the challenge of navigating through an overwhelming amount of complex and dynamic literature. The primary aim of the ArXiv AI Assistant project is to address these challenges by providing a tool that streamlines the literature review process, making it more efficient and comprehensive.

The Key Motivations Behind the ArXiv AI Assistant Project are, **Enhancing Literature Review Efficiency**: The AI assistant is designed to help researchers quickly sift through vast amounts of academic papers, journals, and publications, enabling them to find relevant information swiftly. This efficiency is crucial for researchers who are often pressed for time and need to stay up-to-date with the latest developments in their field.

Providing High-Quality References: One of the critical aspects of any research is the quality of references. The ArXiv AI Assistant aims to use its advanced algorithms to recommend the most credible, relevant, and highly regarded sources, thus ensuring that the researchers have access to the best possible references for their work.

Facilitating Understanding and Analysis: Beyond just retrieving information, the AI assistant is tailored to assist in comprehending complex academic concepts and analyses. This feature is particularly beneficial for new researchers or those venturing into unfamiliar subfields, as it helps them grasp intricate ideas more quickly.

Customized Research Assistance: The AI assistant can be personalized to cater to the specific needs and interests of individual researchers. This customization means that users receive tailored information, saving time and increasing the relevance of the search results.

Encouraging Interdisciplinary Research: By providing access to a broad range of literature across various disciplines, the AI assistant encourages interdisciplinary research, enabling researchers to draw connections between different fields and foster innovative ideas.

Keeping Up with Rapid Advancements: In fields where new discoveries and advancements are frequent, the AI assistant helps researchers stay abreast of the latest research and trends, ensuring that their work remains current and relevant.

Reducing Information Overload: By filtering out irrelevant or less important information, the AI assistant helps researchers avoid the pitfalls of information overload, which can often lead to analysis paralysis.

Providing Writing Suggestions on Niche Topics: The AI assistant is adept at offering writing suggestions and guidance, particularly on niche or specialized topics. This feature is invaluable for researchers working in less-explored areas, as it helps them articulate their ideas more effectively and ensures that their writing is aligned with current scholarly standards and practices.

Conversational Interface for Enhanced User Experience: The AI assistant boasts a conversational interface, making interactions more natural and user-friendly. This approach reduces the learning curve associated with using the tool and allows researchers to articulate their queries in a more intuitive manner, akin to a dialogue with a knowledgeable colleague.

In essence, the ArXiv AI Assistant project is motivated by the desire to revolutionize the way academic research is conducted. By leveraging AI and machine learning, it aims to make the research process more streamlined, insightful, and impactful. This project comes under the domain of Natural Language Processing (NLP) and Information Retreival (IR).

II. Problem Statement

The development of an AI Assistant for Research is a significant endeavor, encompassing a wide array of potential components and capabilities, as outlined in the introduction. For this project, we have decided to narrow our focus to two key areas:

1. **ArXiv as the Primary Source for Literature**: ArXiv, a renowned and widely used repository of academic papers, will serve as the primary source of literature for our AI Assistant. This platform offers a vast collection of preprints and published papers across various fields of science and technology, making it a rich and valuable resource for academic research. By concentrating on ArXiv, we can ensure that the AI Assistant has access to a comprehensive and up-to-date database of scholarly articles, facilitating effective literature reviews and information retrieval. This focus also allows us to tailor the AI Assistant's capabilities to the specific formats and content types prevalent in ArXiv, thereby enhancing the precision and relevance of its outputs.

2. **KBQA Chatbot as the Representation of the AI Assistant**: The core of our AI Assistant will be embodied in a Knowledge-Based Question Answering (KBQA) Chatbot. This chatbot will leverage advanced AI algorithms to interpret and respond to user queries in a conversational manner. Its primary function will be to assist users in navigating the wealth of information available on ArXiv, by providing precise,

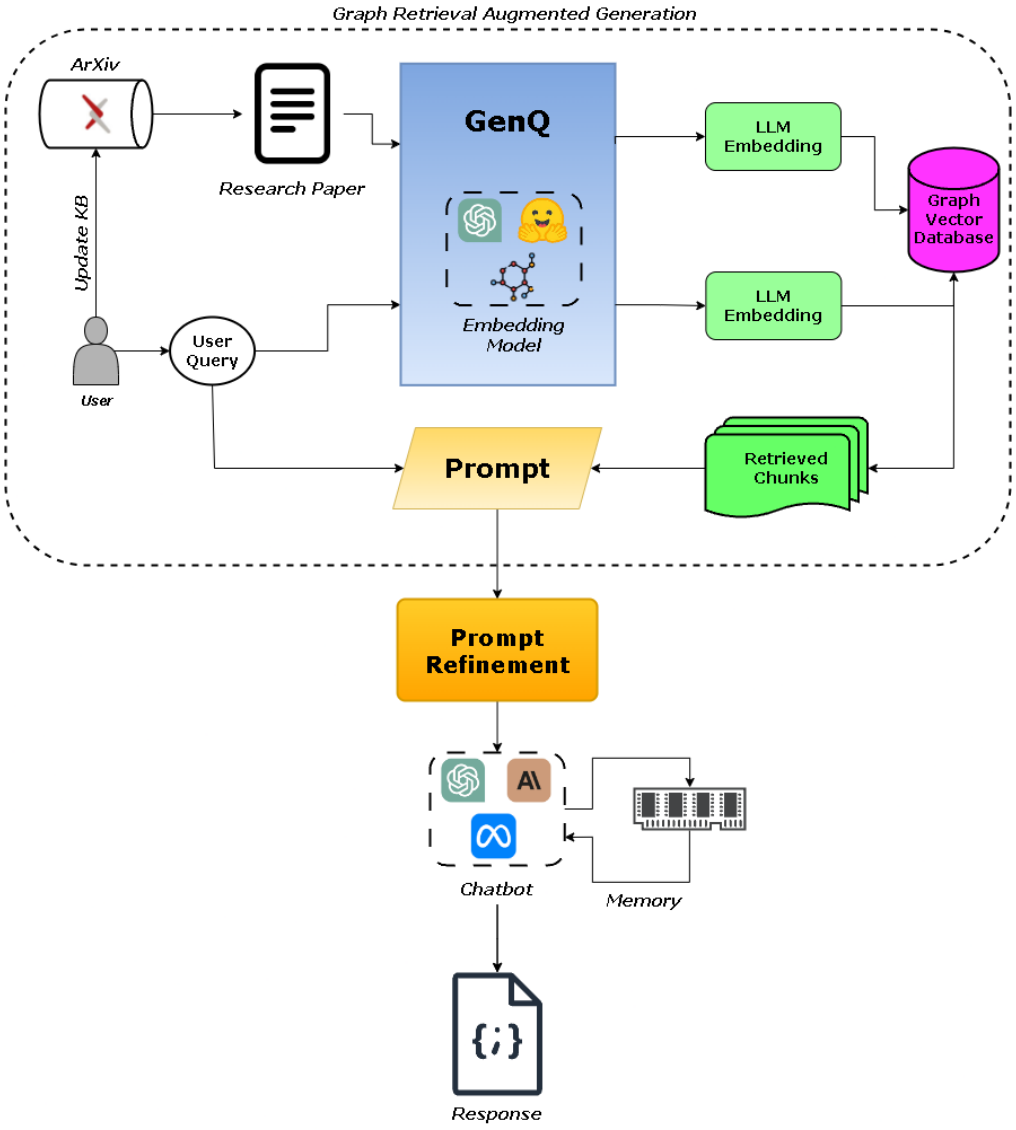

Fig. 1. Overview of pipeline for the ArXiv AI Assistant

contextually relevant answers and suggestions based on the content of the repository.

The core capability of any chatbot lies in its sophisticated algorithms and AI models that enable it to interpret the nuances of human language. This means users can pose questions in their natural, everyday language, without the need for specific keywords or structured queries. The chatbot then processes these queries, identifies the intent and relevant context, and fetches the appropriate responses from its knowledge base.

A KBQA Chatbot is essentially an advanced interactive system designed to understand and respond to user queries through a conversational interface. Its primary function is to provide precise and contextually relevant answers drawn from an extensive knowledge base. This knowledge base is not just a static repository of information; it is continuously updated and expanded, ensuring that the chatbot is always equipped with the latest data and insights.

While there are numerous chat models available in the market, such as ChatGPT, a significant limitation often encountered is their lack of access to specialized, contextual knowledge that might be stored in private knowledge bases (KBs). This gap underscores the need for chatbots equipped with Retrieval Augmented Generation (RAG) capabilities, which we aim to address in our AI Assistant project.

III. Data

For this AI Assistant project, the primary data source is the comprehensive collection of papers available on ArXiv. ArXiv, as a leading digital repository, hosts a wide array of scientific papers across various disciplines, making it an ideal source for our knowledge base.

We use arxiv.py, a python wrapper over the ArXiv API to assist us with the data collection and Knowledge Base CRUD.

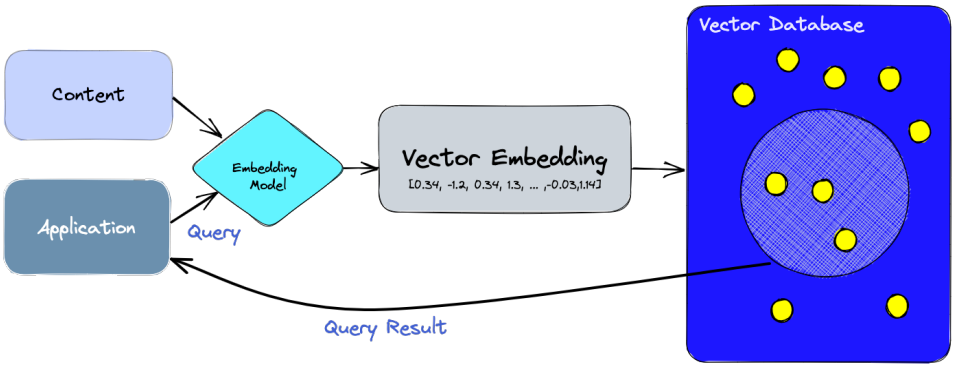

Fig. 2. Overview of working of a Vector Database

A. Seed Selection

To establish the foundation of our knowledge base, we begin by selecting an initial set of papers from ArXiv. This process is termed 'Seed Selection.'

We ensure that these seed papers cover all the categories represented on ArXiv. This broad coverage is crucial to develop a versatile and comprehensive AI Assistant capable of handling queries across various scientific domains.

Our sampling approach involves selecting five random papers from each category. However, to ensure the relevance and quality of these papers, we use a weighted sampling method where the weights are assigned based on the number of citations and references each paper has. This approach helps in selecting papers that are not only representative of their categories but also influential or foundational within their respective fields.

After selecting the initial papers, we expand this seed set by constructing and traversing a graph where the connections (links) are based on the references cited in these papers. This method allows us to organically grow our knowledge base, including papers that are contextually related or foundational to the topics covered in the seed papers.

B. Knowledge Graph Representation

We view and organize the research literature as a knowledge graph. In this graph, each node represents an individual paper from ArXiv.

The connections or edges in this graph are established based on the citations and references between papers. This means that if one paper cites another, an edge is formed between them in the graph.

This graph-based approach enables a more contextual understanding of the literature. By analyzing the connections between papers, the AI Assistant can discern the relationships, influences, and linkages between different research works.

As new papers are added to ArXiv, they are also incorporated into the knowledge graph. This ensures that the knowledge base remains current and reflective of the latest research trends and developments.

When a query is processed by the AI Assistant, it can use this knowledge graph to retrieve not just direct answers from individual papers but also related information, such as foundational research, subsequent developments, and related works in the field.

This methodology of data sourcing and organization through a knowledge graph is fundamental to the operation of the ArXiv AI Assistant. It allows the tool to provide comprehensive, context-aware responses and recommendations, making it a valuable resource for researchers navigating the extensive body of literature on ArXiv.

C. Vector Database

A vector database is a type of database that indexes and stores vector embeddings for fast retrieval and similarity search, with capabilities like CRUD operations, metadata filtering, and horizontal scaling.

It's working in the context of our project is as follows,

1) First, we use the embedding model to create vector embeddings for the content we want to index.
2) The vector embedding is inserted into the vector database, with some reference to the original content the embedding was created from.
3) When the application issues a query, we use the same embedding model to create embeddings for the query and use those embeddings to query the database for similar vector embeddings. As mentioned before, those similar embeddings are associated with the original content that was used to create them.

D. Knowledge Base Updation

We provide 3 methods to add research papers to the knowledge base.

1) **Direct PDF Addition**: Users can add a research paper to the knowledge base by directly uploading its PDF. This method is straightforward and convenient for users who already have the document at hand.
2) **URL Addition**: If the user has a link to the research paper, they can simply add the URL. The system then retrieves the paper from the given link, facilitating ease of data entry.
3) **DOI Input**: Users can also add papers by inputting the Digital Object Identifier (DOI) of the paper. The

DOI is a unique alphanumeric string assigned to a document, providing a permanent link to its location on the internet. This method ensures accuracy in identifying and retrieving the exact paper.

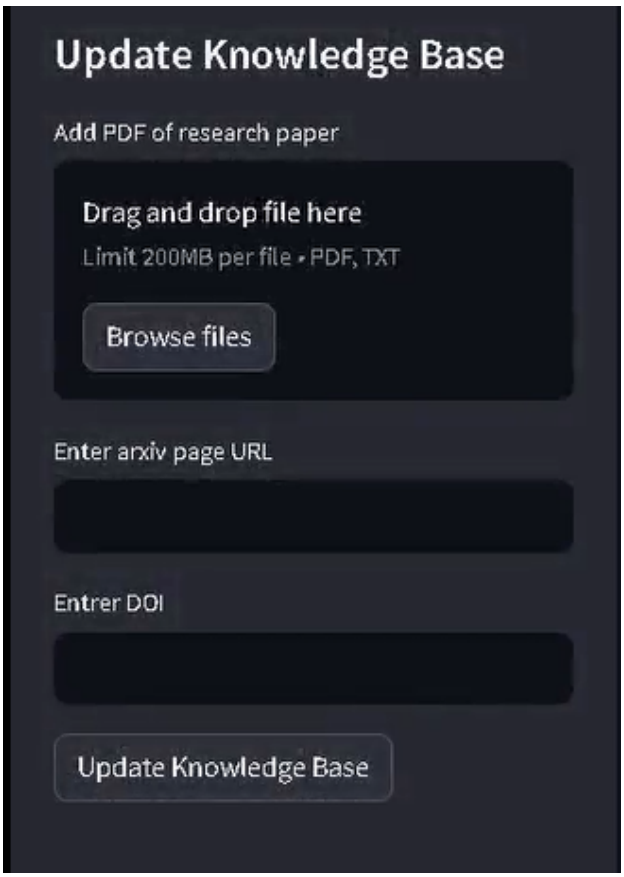

Fig. 3. Methods for Knowledge Base Updation

E. Metadata

In order to enable us to answer more complex queries about our Knowledge Base, we also incorporate metadata information about the research papers. This metadata includes authors, publication dates, journal or conference names, keywords, and citation information.

Metadata also provides contextual and relational insights into the papers. For instance, analyzing citation patterns can reveal the influence and relevance of a paper in a particular field.

The availability of metadata is essential for advanced features such as trend analysis, author collaboration networks, and impact assessment of research papers.

IV. Retrieval

Retrieval is a fundamental process in Knowledge Base Question Answering (KBQA) Chatbots. Considering the vast size of typical knowledge bases, like the collection of ArXiv papers, efficient retrieval becomes a critical challenge. The knowledge base contains a wide array of documents, many of which may be irrelevant to a specific query.

The goal of the retrieval process is to sift through this extensive collection and pinpoint only the documents that are relevant to the user's question. This precision is crucial for providing accurate and useful responses, and for maintaining the efficiency of the chatbot.

Efficient retrieval also helps in avoiding information overload for the user and the model. By filtering out unrelated documents, the chatbot ensures that the user is not overwhelmed with unnecessary information, thus enhancing the user experience.

A. Sentence Transformers

Sentence Transformers are a pivotal tool in the retrieval process. They are used to generate embeddings, which are essentially vector representations of text (in this case, sentences or paragraphs from research papers).

These embeddings capture the semantic essence of the text, allowing for more accurate matching with the user's query. The combination of Sentence Transformers and vector databases facilitates semantic similarity searches. This means that the system can find documents that are contextually similar to the user's query, even if they don't contain the exact keywords.

Sentence Transformer embeddings are dynamic, meaning they can adapt to the evolving nature of language and different contexts. This adaptability is crucial in handling the diverse and expanding content of a knowledge base like ArXiv.

By using these advanced embeddings, the retrieval process becomes more refined, ensuring that the chatbot retrieves the most contextually relevant and up-to-date information in response to queries. This directly contributes to the quality and reliability of the answers provided by the chatbot.

1) Why not Vanilla Transformers?: Why transformer embeddings are so much richer and where does the difference lie between a vanilla transformer and a sentence transformer.

Transformer models have one issue when building sentence vectors: Transformers work using word or token-level embeddings, not sentence-level embeddings.

Before sentence transformers, the approach to calculating accurate sentence similarity with BERT was to use a cross-encoder structure. This meant that we would pass two sentences to BERT, add a classification head to the top of BERT — and use this to output a similarity score.

The cross-encoder network does produce very accurate similarity scores (better than SBERT), but it's not scalable. Ideally, we need to pre-compute sentence vectors that can be stored and then used whenever required. If these vector representations are good, all we need to do is calculate the cosine similarity between each.

With the original BERT (and other transformers), we can build a sentence embedding by averaging the values across all token embeddings output by BERT (if we input 512 tokens, we output 512 embeddings). Alternatively, we can use the output of the first [CLS] token (a BERT-specific token whose output embedding is used in classification tasks).

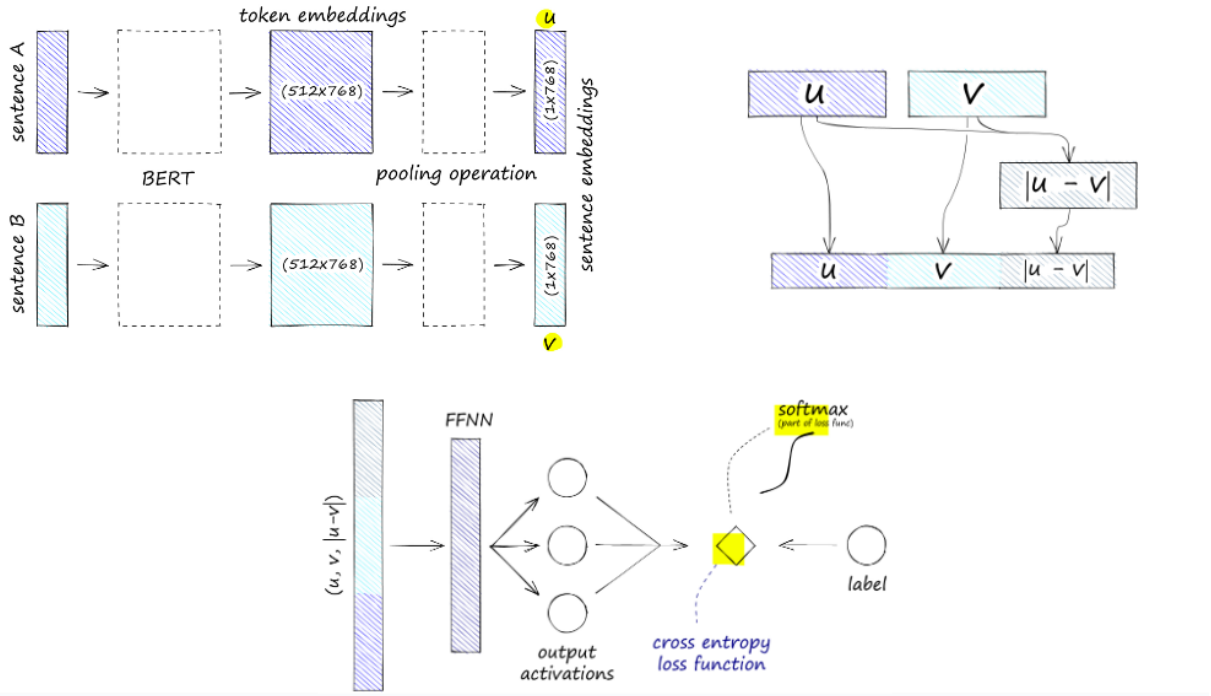

Fig. 4. Overview of Training Process of Sentence Transformers

Using one of these two approaches gives us our sentence embeddings that can be stored and compared much faster. However, the accuracy is not good, and is worse than using averaged GloVe embeddings

B. GenQ Fine Tuning

Fine-tuning effective dense retrieval models is challenging. Bi-encoders (sentence transformers) are the current best models for dense retrieval. Unfortunately, they're also notoriously data-hungry models that typically require a particular type of labeled training data.

One of the most impressive is GenQ. This approach to building bi-encoder retrievers uses the latest text generation techniques to synthetically generate training data. In short, all we need are passages of text. The generation model then augments these passages with synthetic queries, giving us the exact format we need to train an effective bi-encoder model.

A brief description of the method is as follows:

1) Generate queries for pre-existing but unlabeled passages: Creating (query, passage) pairs.
2) Fine-tune a bi-encoder model using these (query, passage) pairs and Multiple Negatives Ranking (MNR) loss.

Margin-Negative-Ranking (MNR) loss plays a vital role in optimizing retrieval models. It focuses on enhancing the model's ability to correctly match a query with its most relevant passage among a set of candidate passages. Here's an elaboration on how MNR loss operates:

MNR loss operates on the principle of **batch processing**. In this context, a batch consists of multiple pairs of queries (Q_i) and passages (P_j). Each query in the batch is paired with a corresponding passage that is considered the most relevant to that query.

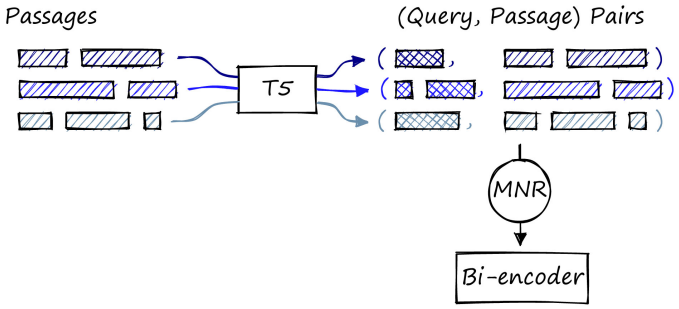

Fig. 5. Unsupervised Fine-Tuning of Retrievers using GenQ

The primary objective of MNR loss is to optimize the model in such a way that, within each batch, the similarity score between a query and its correct corresponding passage is maximized compared to other query-passage pairs.

In a batch, let's say of size 32, each query Q_i is paired with a passage P_j. The index i of the query corresponds to the index j of its correct passage, meaning $P_j = i$ is the correct passage for Q_i.

The model is optimized so that the similarity score between each query Q_i and its corresponding passage $P_j = i$ is higher than the similarity score between that query and any other passage in the batch where $P_j \neq i$.

For instance, if we consider the query Q_3 in a batch, the goal is to ensure that the similarity score between Q_3 and P_3 (the correct passage for Q_3) is higher than the similarity between Q_3 and any other passage in that batch (e.g., P_1, P_2,

P_4, etc., where $P_j \neq 3$).

This method is effective in training the model to accurately identify and rank the most relevant passage for a given query. It helps the model to learn fine-grained distinctions between passages, enhancing its ability to retrieve the most relevant content in response to a query.

The MNR loss is particularly beneficial in dealing with complex and ambiguous queries where multiple passages might seem relevant, but only one (or a few) are truly pertinent. By maximizing the similarity score for correct pairs, the model becomes adept at discerning and prioritizing the most appropriate passages.

V. Question Answering

A. Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) is a pivotal component in modern KBQA systems. It integrates the retrieval of relevant information with a Large Language Model (LLM) for generating answers.

Current LLMs, although powerful, are static and often lack domain-specific knowledge. They might not be up-to-date with the latest research or specialized information contained in niche databases.

RAG works by taking an input query and retrieving a set of relevant or supporting documents from a given source, like Wikipedia or a specialized database. These documents are then concatenated with the original input prompt, providing enriched context to the LLM. The LLM uses this combined input to generate a more informed and accurate response.

In our project, RAG is implemented to work on knowledge graphs, enhancing its capability to fetch contextually relevant and interconnected information from a structured knowledge base.

B. Conversational Memory

Conversational memory allows stateless LLMs to remember previous interaction. Conversational memory is how a chatbot can respond to multiple queries in a chat-like manner. It enables a coherent conversation, and without it, every query would be treated as an entirely independent input without considering past interactions.

Forms of Conversational Memory:

- **Conversation Buffer Memory**: Stores entire conversation for reference.
- **Conversation Summary Memory**: Keeps a summarized record of past interactions.
- **Conversation Buffer Window Memory**: Maintains a moving window of recent conversation parts.
- **Conversation Summary Buffer Memory**: Combines summary and buffer approaches for a comprehensive recall of past interactions.

These forms of memory enable the chatbot to recall previous queries and responses, ensuring continuity and relevance in ongoing conversations.

We are utilizing Conversation Buffer Window Memory in this project.

C. Query Refinement

(takes in the past conversational history and the current query and refines the query accordingly)

Query refinement involves taking the current query and the conversational history, and then refining the query based on this context. This process ensures that the query being processed is in line with the ongoing conversation's context and user intent. By considering past interactions, the query refinement process can more accurately interpret the user's current query, leading to more relevant and accurate responses. This capability significantly enhances the user experience, as it allows the chatbot to provide responses that are not just based on the immediate query, but are informed by the entirety of the conversation.

In summary, these components – Retrieval Augmented Generation, Conversational Memory, and Query Refinement – collectively enhance the functionality of a KBQA Chatbot. RAG provides the necessary depth and breadth in answers by leveraging external knowledge sources. Conversational memory ensures the chatbot understands and remembers the context of ongoing interactions, and query refinement fine-tunes the queries based on conversational history, leading to a more coherent and contextually aware conversational experience.

```
def query_refiner(conversation, query):

    response = openai.Completion.create(
        model=QUERY_MODEL,
        prompt=f"Given the following user query and conversation log,\
              formulate a question that would be the most relevant to provide \
                the user with an answer from a knowledge base.\
                    \n\nCONVERSATION LOG: \n{conversation}\n\nQuery: {query}\n\nRefined Query:",
        temperature=0.7,
        max_tokens=256,
        top_p=1,
        frequency_penalty=0,
        presence_penalty=0
    )
    return response['choices'][0]['text']
```

Fig. 6. Code for Query Refinement

VI. User Interface

The user interface (UI) of the ArXiv AI Assistant is developed using Streamlit, a powerful and efficient tool for building interactive and user-friendly web applications.

The main component of the UI is a chatbot interface designed for ease of use and functionality. It presents users with a clean, intuitive platform to interact with the AI Assistant. The chatbot interface includes an interactive chat window where users can type their queries and receive responses. This interaction mimics a natural conversation flow, enhancing the user experience.

A distinctive feature of the UI is the integration of a sidebar that allows users to update the Knowledge Base. This feature ensures that the system remains dynamic and adaptable to new information.

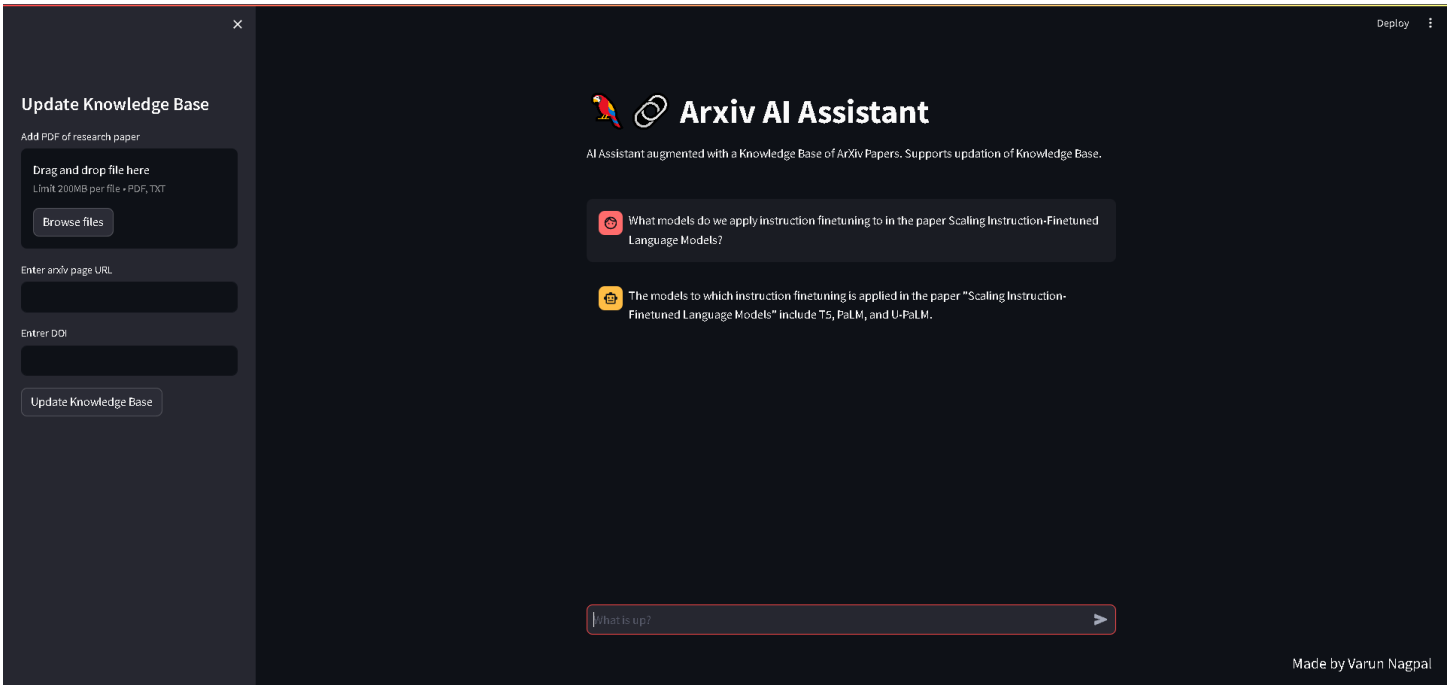

Fig. 7. Sample User Interface for the ArXiv AI Assistant