

Developing an Approach to Building, Testing, and Evaluating an LLM-based RAG-enabled AI tutor

Rebecca A. Scott
scotra@nmsu.edu

Kaverinathan Dhanapal
kevin4tx@nmsu.edu

New Mexico State University
Las Cruces, NM 88003

Abstract

Tutoring services have the potential to improve educational outcomes for university students. Although the use of artificial intelligence (AI) in tutoring systems is increasing, customized AI tutors designed specifically for university students have not been widely explored. This study explores the process of building, testing, and evaluating a large language model (LLM)-based retrieval-augmented generation (RAG) AI tutor using Google Colab. Such a customized online tutor may provide valuable support to students preparing for final examinations, certification examinations, and professional examinations. However, traditional human-driven online tutoring services can be expensive, decentralized, and limited by tutor availability and operating hours. To address these limitations, this article presents a step-by-step approach for building, testing, and evaluating an LLM-based, RAG-enabled AI tutor. The Foundations of Information Systems textbook published by OpenStax is used as the reference textbook. The proposed approach provides a practical pedagogical framework through which information systems (IS) students can build, test, and use their own customized AI tutor based on course-specific learning materials. The findings show the potential of AI based customized tutor system to address specific needs of students for each one of their coursework. The approach demonstrates how students can engage with AI technologies while developing a personalized self-learning resource within the scope of their course content.

Keywords: Large Language Model (LLM), Retrieval-Augmented-Generation (RAG), Artificial Intelligence (AI), pedagogy, tutor

Developing an Approach to Building, Testing, and Evaluating an LLM-based RAG-enabled AI tutor

Rebecca A. Scott and Kaverinathan Dhanapal

1. INTRODUCTION

University education, from the freshman to senior levels, represents a crucial stage in students' educational development, preparing them to become lifelong learners, problem solvers, and responsible professionals (McGovern & Brewer, 2003). Existing research indicates that tutoring can positively influence undergraduate students' academic success, professional development, and retention (Reinheimer & McKenzie, 2011; Ruiz & Fandos, 2014). Intelligent tutoring systems (ITS) have subsequently received increasing attention as an alternative or complement to traditional tutoring, particularly in response to limitations associated with the availability, consistency, and accessibility of human tutoring services (Maurya & Kochmar, 2025; McFarlane, 2016). Previous research has found that ITS can achieve learning outcomes comparable to those of human tutoring, highlighting their potential as an important approach to supporting undergraduate education (VanLehn, 2011). More recent studies suggest that ITS can facilitate reasoning, support diverse learners through student-centered learning processes, and provide personalized educational experiences for university students (Mousavinasab et al., 2021; Alрахhawi et al., 2023; Lin et al., 2023).

Recent advances in large language models (LLMs) provide additional opportunities to extend the capabilities of ITS. Unlike traditional ITS, which often rely on pred rules, knowledge structures, or domain-specific architectures, LLM-based systems can support more flexible natural-language interactions with learners. However, LLMs may generate responses that are not grounded in a course's prescribed learning materials and may produce inconsistent or factually inaccurate information. Retrieval-augmented generation (RAG) provides a potential approach for addressing this limitation by retrieving relevant information from a defined knowledge source and incorporating that information into the LLM's response. In an educational context, a RAG-enabled tutor can therefore be configured to ground its responses in course-specific materials, such as a recommended textbook, while maintaining the conversational capabilities of an LLM. In this article, the authors focus on demonstrating how undergraduate students can build and use a personalized tutor by combining LLM capabilities with an ITS framework and RAG methodology. The approach specifically describes how the learning process can be anchored in textbook content and how students can interact with the tutor through basic natural-language queries.

Access to commercial LLMs through application programming interfaces (APIs), as well as the use of Google Colab to develop Python-based LLM applications, is not new. Similarly, numerous publicly available tutorials describe the development of RAG-based applications. The contribution of this article is therefore not the introduction of a new RAG or LLM technology. Rather, the article provides an integrated, step-by-step pedagogical reference that brings together freely accessible tools, Google Colab, LLM API access, RAG methodology, and a recommended textbook in PDF format to guide undergraduate students in developing a customized AI tutor for their coursework. This approach is intended to help students interact with an AI tutor whose responses are grounded in the textbook selected for their course, rather than relying solely on the general and potentially changing knowledge of an LLM. The approach also provides students with an opportunity to engage directly with the development and configuration of the tutoring system as part of their learning experience.

The approach presented in this article is related to prior work such as Viswanathan et al. (2026), which demonstrates the use of LLMs in educational contexts. However, the present approach emphasizes a student-configured tutoring system grounded in a specific course textbook, rather than positioning the LLM primarily as a general-purpose teacher. This distinction is important because the proposed tutor is intended to support students' independent engagement with their prescribed learning materials. Seminal research on instructional strategies suggests that meaningful learning occurs when students actively engage with instructional materials and apply knowledge through problem solving (Merrill et al., 1992). When students encounter difficulties while learning from their textbooks, a personalized one-to-one

tutoring approach may provide additional support by allowing them to ask questions and receive explanations grounded in the same instructional material at their own pace. Accordingly, this article provides a practical guide for undergraduate students to develop a RAG-enabled, LLM-based AI tutor whose responses are grounded in the recommended textbook for their course.

Our paper is structured as follows: In the next section, we outline how to set up Google Colab and access Gen-AI LLM to answer questions from a student. In section 3, we demonstrate how reference text in pdf format is used for setting up RAG and next section is about generation of data chunks from that pdf file. In section 5, we demonstrate how RAG-enabled answers are obtained. Then, section 6 covers the evaluation of the output. Finally, results are discussed in section 7 and conclusions are presented in section 8.

2. QUESTIONS ANSWERED BY A GEN AI LLM TUTOR

The tutoring system is developed and configured by students for their intended coursework, rather than being a fully preconfigured system provided by the instructor. The users need a google account to log in and access Google Colab (GC). GC has provision to add code and text to the workspace. The text box can be used to write descriptions about programming workflow and python codes to help comprehend the code in future. The Appendix 1 provides the python code to install python packages required to build our project. The code builds upon the outputs and variables generated in the preceding sections; therefore, the appendices should not be treated as independent scripts. Students can follow the accompanying explanations in each appendix to understand the purpose of each step while executing the code. This sequential approach allows students to build, test, and use the customized AI tutor through a single integrated workflow. The langchain libraries are needed to build LLM applications and tasks such as document readers, vector databases, embedding, and retrievers. The pymupdf library is needed to read and process PDF files, and tiktoken is used to track LLM token usage. The ragas and evaluate are evaluation libraries. Appendix 2 has the python program script used to load the relevant applications from the libraries added in the Appendix 1. The line number (11) imports the json library needed to be read from the json data file that contains the OpenAI application key. Line number (13) enables access to OpenAI models. Line number (18) imports chroma database to store and search vector embeddings.

The Appendix 3 has the python script to read the API key for OpenAI access, and the base URL from the json config file and initialize the OpenAI client. This process is carried out from line number (23) to (30). The Appendix 4 has the python code script to create a function to query the OpenAI model using student questions to the tutor, then retrieve and provide generated responses. The line number (31) sets the query length using max tokens parameter. The temperature and top_p are the hyperparameters that control the creativity, randomness, and focus of the LLM when it generates responses. A low temperature value leads to logical and highly focused response rather than creative and diverse output. A low top_p value controls the pool of words selected to be high probable values, whereas high top_p value leads to wider range of words enabling diverse and varied responses. The line number (33) defines the LLM model version used in this tutor tool.

In this tutor tool, the reference textbook is an opensource pdf file of "Foundations of Information Systems" provided by openstax.org. The Appendix 5 has three sample questions posted to the ITS tutor by the student, to enable one-on-one interaction to clearly understand the concepts explained in the textbook. The line numbers (42), (46), and (49) are the three questions. The direct responses, provided by the OpenAI LLM tool, for these questions are stored in a data frame as a reference to compare them to the RAG-enabled LLM tutor responses and evaluate the effectiveness of the AI tutors.

3. PREPARING TEXTBOOK FOR RAG

The students who are in need of a tutor to focus on personalized one-on-one support and tailor the lessons to fill learning gaps in any particular class can use the PDF file of their textbook for the RAG-based AI tutor. In this article, we use the pdf file of the textbook "Foundations of Information Systems", an open-source textbook provided by openstax.org. After loading the textbook pdf file to Google Colab files, the line numbers (58) to (61) in the Appendix 6 show how the pdf file is loaded on to the python

environment for further processing. Further, the line numbers (62) to (67) show how to send the textbook pdf file to langchain community document, the next step in the RAG process.

4. DATA CHUNKING

In order to create a text splitter to produce chunks of the textbook content, the line numbers (68) to (73) in Appendix 7 are used. Specifically, the line number (70) denotes that each chunk consists of 512 characters. Further, the OpenAI LLM is used to convert the above text chunks into vector embeddings to enable the machines to understand and process the semantic meaning of text rather than just processing them as keywords. This vector embedding creation code is presented in Appendix 8 from line number (74) to (80). All these vector embeddings need to be stored in uniquely designed vector databases that can handle high-dimensional representations of unstructured text data. In the Appendix 9, the line numbers (81) to (90) demonstrate the python code used for this vector database generation.

5. RAG-BASED RESPONSE GENERATION

To generate a RAG-based response, firstly the vector database is converted into a retriever and five documents most relevant to the student's query are retrieved. In order to do this, a system and user prompt shown in Appendix 10-line numbers (91) to (124) are used. Secondly, the RAG response function is created to generate RAG-based responses to student's queries. This process stretches from line numbers (133) to (154) in Appendix 10. The line number (133) defines the function with parameters such as max tokens, temperature, and top-p. And line number (142) specifies the OpenAI LLM used in this function. Finally, the answers for student's queries, from the RAG-enabled method, are generated and stored in the result data frame. The python code for this process is provided in Appendix 11-line numbers (155) to (163).

6. EVALUATION OF RAG-BASED OUTPUT

In this output evaluation process, the responses provided directly by LLM model are used as the baseline for evaluation of the potentially high-quality responses obtained from the RAG-based method. For the evaluation process, the system prompts to define the evaluation criteria and to train the evaluator are shown in Appendix 12 from line numbers (164) to (224). In addition to the prompts, a function to evaluate the groundedness and relevance of the answers based on the context of the student queries is provided in Appendix 13 from line numbers (225) to (256). In this article, there are three sample questions from the topics like information systems for business purposes, IT in healthcare operations, and principles of cyber security.

Q1: How does an organization view information technology as a strategic partner driving innovation and new revenue opportunities?

Q2: How do well-designed databases improve operations in healthcare systems?

Q3: What are some key principles of the General Data Protection Regulation (GDPR) that organizations should comply with?

are used to compare the responses from direct LLM tutor and RAG-based AI tutor. The evaluation scores of the RAG-enabled responses are stored in the results_dataframe as shown in Appendix 14-line numbers (257) to (288). The evaluation scores for direct LLM responses are also stored in the results_dataframe, as shown in Appendix 15-line numbers (289) to (321). The scores comparison table (Table 1) is shown below. However, the effectiveness and comparability of the tutoring system across different subjects depend primarily on the PDF textbook used to develop and configure the tutor and the extent to which students' questions remain within the scope of the textbook content. Thus, the tutoring system's responses and learning support are grounded in the specific content provided in the selected textbook.

7. RESULTS

Retrieval-Augmented Generation (RAG) consistently performs best in both relevance and groundedness because it incorporates external documents to support its responses. In contrast, standard large language models often lack grounding and may include vague or inaccurate information due to limited context. The strong groundedness scores of RAG-based methods indicate that its responses are closely tied to source material, reducing errors and improving reliability. Thus, for undergraduate students who are looking for Intelligent tutor services (ITS) with personalized, one-on-one support, and tailor the

lessons to fill specific knowledge gaps, the RAG-enabled AI LLM based tutors seem to perform best when compared to direct LLM based tutors.

The performance of the Direct LLM Tutor and the RAG-enabled Tutor was evaluated using three questions based on the course textbook. Two dimensions were assessed: groundedness and relevance, using a five-point scoring scale, where a higher score indicates better performance (Table 1). Groundedness refers to the extent to which the tutor's response is supported by and consistent with the information contained in the reference textbook. Relevance refers to the extent to which the response appropriately addresses the question asked by the student.

The results indicate a clear difference in groundedness between the two tutoring approaches. For the Direct LLM Tutor, the groundedness scores for Questions 1, 2, and 3 were 3, 4, and 5, respectively, resulting in an average groundedness score of 4.0. In contrast, the RAG-enabled Tutor received a groundedness score of 5 for all three questions, resulting in an average score of 5.0. This suggests that the RAG-enabled Tutor was more consistently grounded in the prescribed textbook content than the Direct LLM Tutor.

For relevance, both tutoring approaches received a score of 5 for all three questions, resulting in an average relevance score of 5.0 for both systems. This indicates that both tutors were able to provide responses that directly addressed the questions posed by the students. However, the difference in groundedness suggests that while a direct LLM can produce relevant responses, its responses may not always be equally well anchored to the specific course material.

Overall, the results suggest that the RAG-enabled Tutor maintained the high relevance of the Direct LLM Tutor while providing more consistent grounding in the designated textbook. The improvement in groundedness is particularly important in an educational setting because students are expected to receive information that is consistent with the prescribed course materials. Nevertheless, these findings should be interpreted cautiously because the evaluation was based on only three questions and therefore provides an initial demonstration rather than a comprehensive assessment of tutor performance.

Table 1: Score comparison

	Questions	Groundedness Score	Relevance Score
Direct LLM Tutor	1	3	5
	2	4	5
	3	5	5
RAG-enabled Tutor	1	5	5
	2	5	5
	3	5	5

8. CONCLUSIONS AND OPPORTUNITIES FOR FUTURE WORK

This article provides a convenient and easy to follow step-by-step approach to building, testing, and evaluation of an LLM based, RAG-enabled AI tutor. The approach accesses the Foundations of Information systems textbook (pdf file) from openstax.org through Google Colab. There were three sample questions from students, that were posted to direct LLM tutor and RAG-enabled AI tutor for comparison of the evaluated answers to the students from the ITS tutors. The evaluation of the tutoring system in this study relies in part on an LLM-as-judge approach, in which a large language model is used to assess the quality and relevance of the tutoring system's responses. This approach was selected because it provides a practical and scalable means of evaluating a relatively large number of responses against the content of the course textbook. In the context of this study, the evaluation is particularly focused on whether the tutor's responses remain grounded in the textbook content and appropriately address students' questions within the scope of the assigned material. Nevertheless, the use of an LLM as an evaluator introduces potential concerns regarding evaluator bias, consistency, and the reliability of automated judgments. The LLM's assessment may also be influenced by the way evaluation criteria and prompts are formulated. Therefore, the results should be interpreted as an automated assessment

of response quality rather than as an objective substitute for human evaluation. Future studies could strengthen the evaluation methodology by incorporating independent human evaluators, inter-rater reliability measures, and comparisons between human and LLM-based assessments. Such validation would provide greater confidence in the reliability and generalizability of the findings.

Step-by-step approach provided in this article is valuable to undergraduate students and other interested categories of students. Using a tutor that provides responses grounded in your textbook can enable personalized, one-on-one educational setting.

There are a variety of options available for this work to be expanded. For example, various topics can be explored and customized intelligent tutoring systems can be developed for individualized education plans for students who want to learn at their own pace.

9. REFERENCES

- Alrakhawi, H. A., Jamiat, N., & Abu-Naser, S. S. (2023). Intelligent tutoring systems in education: a systematic review of usage, tools, effects and evaluation. *Journal of Theoretical and Applied Information Technology*, 101(4), 1205–1226. DOI:<https://www.jatit.org/volumes/Vol101No4/6Vol101No4.pdf>
- Anderson, J. R., Boyle, C. F., & Reiser, B. J. (1985). Intelligent tutoring systems. *Science*, 228(4698), 456–462. <https://doi.org/10.1126/science.228.4698.456>
- Lin, C.-C., Huang, A. Y. Q., & Lu, O. H. T. (2023). Artificial intelligence in intelligent tutoring systems toward sustainable education: a systematic review. *Smart Learning Environments*, 10(1), 41. <https://doi.org/10.1186/s40561-023-00260-y>
- Maurya, K. K., & Kochmar, E. (2025). Pedagogy-driven Evaluation of Generative AI-powered Intelligent Tutoring Systems. *ArXiv*. <https://doi.org/10.48550/arxiv.2510.22581>
- McFarlane, K. J. (2016). Tutoring the tutors: Supporting effective personal tutoring. *Active Learning in Higher Education*, 17(1), 77–88. <https://doi.org/10.1177/1469787415616720>
- McGovern, T. V., & Brewer, C. L. (2003). Undergraduate education. *Handbook of Psychology: Volume 1, History of Psychology*, 465. <https://doi.org/10.1002/0471264385.wei0123>
- Merrill, D. C., Reiser, B. J., Ranney, M., & Trafton, J. G. (1992). Effective tutoring techniques: A comparison of human tutors and intelligent tutoring systems. *Journal of the Learning Sciences*, 2(3), 277–305. https://doi.org/10.1207/s15327809jls0203_2
- Mousavinasab, E., Zarifsanaiey, N., R. Niakan Kalhori, S., Rakhshan, M., Keikha, L., & Ghazi Saeedi, M. (2021). Intelligent tutoring systems: a systematic review of characteristics, applications, and evaluation methods. *Interactive Learning Environments*, 29(1), 142–163. <https://doi.org/10.1080/10494820.2018.1558257>
- Reinheimer, D., & McKenzie, K. (2011). The impact of tutoring on the academic success of undeclared students. *Journal of College Reading and Learning*, 41(2), 22–36. <https://doi.org/10.1080/10790195.2011.10850340>
- Ruiz, N., & Fandos, M. (2014). The role of tutoring in higher education: improving the student's academic success and professional goals. *Revista Internacional de Organizaciones*, 12, 89–100. DOI: <https://rpd.unibo.it/article/view/21423>
- VanLehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4), 197–221. <https://doi.org/10.1080/00461520.2011.611369>

Viswanathan, V., Weitz, R., & Rosenthal, D. (2026). Teaching Tip: Generative AI in Education: Building Intelligent Tutors With Large Language Models. *Journal of Information Systems Education*, 37(2), 167-183. <https://doi.org/10.62273/COCJ8767>

Appendix A

The python code below is used to install and import required python libraries to the Google Colab (GC) environment. The user must register as a google account holder and log in to GC and use a code box to add these commands. Comments about the command lines can provided using the commenting sign #. Each line in the code provided in the appendices are numbered to enable the user to reference the article and follow along easily. The line numbers automatically appear in GC code box.

```
(1) !pip install -q \           # Installs Python packages
(2)     langchain_community==0.4.1 \       # connectors and tools for
                                           document loaders, vector databases,
                                           embedding, retrievers.

(3)     langchain==1.3.2 \       # helps to build LLM applications
(4)     chromadb==1.5.5 \       # a vector database
(5)     pymupdf==1.27.0 \       # to read and process PDF files
(6)     tiktoken==0.11.0 \      # to count and process tokens used by LLMs
(7)     ragas==0.4.0 \          # to evaluate quality of RAG system
(8)     datasets==4.3.0 \       # to handle datasets for AI experiments
(9)     evaluate==0.4.6         # to calculate machine learning evaluation metrics
```

While executing the above code box, any warnings and error messages can be ignored.
The GC runtime should be restarted after running the above code box.

Appendix B

```
(10) import os                 # Interact with the operating system
(11) import json               # Read and write JSON data

(12) from langchain.document_loaders import PyMuPDFLoader    # Load and extract text
                                                             from PDF files
(13) from openai import OpenAI  # Access OpenAI models and services

(14) import tiktoken           # Tokenizer used for counting and splitting text for models
(15) import pandas as pd       # Load, manipulate, and analyze tabular data

(16) from langchain.text_splitter import RecursiveCharacterTextSplitter  # Break text into
                                                                           overlapping chunks for processing
(17) from langchain.embeddings.openai import OpenAIEmbeddings            # Create vector
                                                                           embeddings using OpenAI models
(18) from langchain.vectorstores import Chroma                          # Store and search vector embeddings
                                                                           using Chroma DB
(19) from ragas import evaluate
(20) from ragas.metrics import (
    Faithfulness,
    AnswerRelevancy,
    LLMContextPrecisionWithoutReference,
)
(21) from datasets import Dataset    # Used to structure the input in tabular format
(22) from langchain_openai import ChatOpenAI  # to communicate with LLM for metrics
```

Appendix C

```
(23) file_name = 'config.json'           # Name of the configuration file
(24) with open(file_name, 'r') as file:  # Open the config file in read mode
(25)     config = json.load(file)        # Load the JSON content as a dictionary
(26)     OPENAI_API_KEY = config.get("OPENAI_API_KEY")    # Extract the API key
                                                    from the config
(27)     OPENAI_API_BASE = config.get("OPENAI_API_BASE")  # Extract the OpenAI
                                                    base URL from the
config
                                                    # Store API credentials in environment variables
(28) os.environ['OPENAI_API_KEY'] = OPENAI_API_KEY        # Set API key as
                                                    environment variable
(29) os.environ["OPENAI_BASE_URL"] = OPENAI_API_BASE     # Set API base URL as
                                                    environment variable
                                                    # Initialize OpenAI client
(30) client = OpenAI()                                # Create an instance of the OpenAI client
```

Appendix D

```
                                                    # Define a function to get a response
(31) def response(user_prompt, max_tokens=500, temperature=0.3, top_p=0.95):
                                                    # Create a chat completion using the OpenAI client
(32)     completion = client.chat.completions.create(
(33)         model="gpt-4o-mini",           # Specify the model to use
(34)         messages=[
(35)             {"role": "user", "content": user_prompt}    # User prompt is the input/query to respond to
(36)         ],
(37)         max_tokens=max_tokens,          # Max number of tokens to generate in the response
(38)         temperature=temperature,        # Controls randomness in output
(39)         top_p=top_p                    # Controls diversity through nucleus sampling
(40)     )
(41)     return completion.choices[0].message.content        # Return the text content from
                                                    the model's reply
```

Appendix E

Collecting answers directly from AI LLM for comparison with RAG enabled responses

```
(42) question_1 = "How does an organization view information technology as a strategic partner  
(43) driving innovation and new revenue opportunities?"  
(44) base_prompt_response_1 = response(question_1)  
(45) base_prompt_response_1  
  
(46) question_2 = "How do well-designed databases improve operations in healthcare systems?"  
(47) base_prompt_response_2 = response(question_2)  
(48) base_prompt_response_2  
  
(49) question_3 = "What are some key principles of the General Data Protection Regulation  
(50) (GDPR) that organizations should comply with?"  
(51) base_prompt_response_3 = response(question_3)  
(52) base_prompt_response_3  
  
# Create the DataFrame  
(53) result_dataframe = pd.DataFrame({  
(54)   "questions": [question_1, question_2, question_3],  
(55)   "base_prompt_responses": [base_prompt_response_1, base_prompt_response_2,  
(56)   base_prompt_response_3]})  
  
# Display the DataFrame  
(57) result_dataframe.head()
```

Appendix F

The Reference Textbook

The dataset consists of a textbook in PDF format.

Here is the textbook and link:

Foundations of Information Systems - openstax (<https://openstax.org/details/books/foundations-information-systems>)

```
# Uploading multiple pdf files:  
# Path to your single PDF file  
(58) pdf_path = "/content/Information_Systems.pdf"  
  
# Example: open and read the file (basic check)  
(59) with open(pdf_path, "rb") as f:  
(60)   pdf_files = f.read()  
(61) print("PDF loaded successfully!")  
  
Load the PDF file:  
(62) from langchain_community.document_loaders import PyMuPDFLoader  
(63) pdf_path = "/content/Information_Systems.pdf"  
(64) loader = PyMuPDFLoader(pdf_path)  
(65) documents = loader.load()  
  
# If you still want the same variable structure:  
(66) all_documents = documents  
(67) print(f"Total pages loaded: {len(all_documents)}")
```

Appendix G

Data Chunking

```
                                # Initialize a text splitter that uses OpenAI's token encoder
(68) text_splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
(69)     encoding_name='cl100k_base',          # Encoding used by popular LLMs
(70)     chunk_size=512,                      # Each chunk will have up to 512 tokens
(71) )

                                # Use the text splitter to divide the PDF content into smaller chunks
(72) document_chunks = text_splitter.split_documents(all_documents)          # Returns a list of
                                                                              chunked
documents
(73) print(f"Created {len(document_chunks)} chunks.")
```

Appendix H

Open AI to convert text chunks into vector embeddings

```
                                # Initialize the OpenAI Embeddings model with API credentials
(74) embedding_model = OpenAIEmbeddings(
(75)     openai_api_key=OPENAI_API_KEY,        # Your OpenAI API key for authentication
(76)     openai_api_base=OPENAI_API_BASE      # The OpenAI API base URL endpoint
(77) )

                                # Generate embeddings (vector representations) for the first two document chunks
(78) embedding_1 = embedding_model.embed_query(document_chunks[0].page_content) # Embedding for chunk 0
(79) embedding_2 = embedding_model.embed_query(document_chunks[1].page_content) # Embedding for chunk 1

                                # Check and print the dimension (length) of the embedding vector
(80) print("Dimension of the embedding vector ", len(embedding_1))          # Typically 1536 or
                                                                              2048 depending on
model
```

Appendix I

Vector Database Generation

```
(81) from langchain_openai import OpenAIEmbeddings
(82) embedding_model = OpenAIEmbeddings(
(83)     model="text-embedding-3-small",
(84)     chunk_size=50 # ▼ reduce batch size (default is too large)
(85) )
(86) from langchain_community.vectorstores import Chroma
(87) vectorstore = Chroma.from_documents(
(88)     document_chunks,
(89)     embedding_model
(90) )
```

Appendix J

Generate RAG-based responses

```
# Wrapping the vector store into a retriever object to fetch the most relevant documents for a given
# query using similarity search
(91) retriever = vectorstore.as_retriever(
(92)     search_type='similarity',          # Use similarity search (based on vector distance)
(93)     search_kwargs={'k': 5})          # Retrieve top 5 most relevant documents
(94) )

# Define the system prompt for the model
(95) qna_system_message = """
(96) You are an AI assistant designed to support Students in tutoring and efficiently reviewing
(97) Chapters. Your task is to provide evidence-based, concise, and relevant summaries based on
(98) the context questions provided from textbook chapters.
(99) User input will include the necessary context for you to answer their questions. This context
(100) will begin with the token:

# Context
(101) The context contains excerpts from one or more research papers, along with associated
(102) metadata such as titles, authors, abstracts, keywords, and specific sections relevant to the
(103) query.
(104) When crafting your response
(105) -Use only the provided context to answer the question.
(106) -If the answer is found in the context, respond with concise and insight-focused
(107) summaries.
(108) -If the question is unrelated to the context or the context is empty, clearly respond with:
(109) "Sorry, this is out of my knowledge base."
(110) Please adhere to the following response guidelines:
(111) -Provide clear, direct answers using only the given context.
(112) -Do not include any additional information outside of the context.
(113) -Avoid rephrasing or generalizing unless explicitly relevant to the question.
(114) -If no relevant answer exists in the context, respond with:
(115) "Sorry, this is out of my knowledge base."
(116) -If the context is not provided, your response should also be: "Sorry, this is out of my
(117) knowledge base." this is the description for an AI research assistant. Give me one for AI
(118) based tutor for a college student on the subject, Foundations of Information Systems
(119) Here is a guidance of how to structure your response:
(120) Answer:
(121) [Answer based on context]
(122) Source:
(123) [Source details with page or section]
(124) """
```

Message Template

```
# Define the user message template
(125) qna_user_message_template = """
(126) # Context
(127) Here are some excerpts from Information Systems Textbook and their sources that are (128)
relevant to the Students' question to a tutor mentioned below:
(129) {context}
(130) # Question
(131) {question}
(132) """
```

Response Function

```
(133) def generate_rag_response(user_input,k=5,max_tokens=500,temperature=0.3,top_p=0.95):
(134)     global qna_system_message,qna_user_message_template
# Retrieve relevant document chunks
(135)     relevant_document_chunks = retriever.get_relevant_documents(query=user_input,k=k)
(136)     context_list = [d.page_content for d in relevant_document_chunks]
# Combine document chunks into a single context
```

```
(137) context_for_query = ". ".join(context_list)

(138) user_message = qna_user_message_template.replace('{context}', context_for_query)
(139) user_message = user_message.replace('{question}', user_input)                                # Generate the response

(140) try:
(141)     response = client.chat.completions.create(
(142)         model="gpt-4o-mini",
(143)         messages=[
(144)             {"role": "system", "content": qna_system_message},
(145)             {"role": "user", "content": user_message}
(146)         ],
(147)         max_tokens=max_tokens,
(148)         temperature=temperature,
(149)         top_p=top_p
(150)     )
                                                    # Extract and print the generated text from the response
(151)     response = response.choices[0].message.content.strip()
(152) except Exception as e:
(153)     response = f'Sorry, the system encountered the following error: \n {e}'
(154)     return response
```

Appendix K

Answers from RAG enabled method

```
(155) response_with_rag_1 = generate_rag_response(question_1)
(156) response_with_rag_1

(157) response_with_rag_2 = generate_rag_response(question_2)
(158) response_with_rag_2

(159) response_with_rag_3 = generate_rag_response(question_3)
(160) response_with_rag_3
```

Storing the answer output

```
                                                    # Add the results to a new column in the DataFrame
(161) result_dataframe['responses_with_RAG'] = [response_with_rag_1, response_with_rag_2, (162)
response_with_rag_3]
                                                    # Display the DataFrame
(163) result_dataframe.head()
```

Appendix L

RAG Output Evaluation

System prompt to define the evaluation criteria and train the evaluator:

```
(164) groundedness_rater_system_message = ""
(165) You are tasked with rating AI generated answers to questions posed by users.
(166) You will be presented a question, context used by the AI system to generate the answer (167)
and an AI generated answer to the question.
(168) In the input, the question will begin with ###Question, the context will begin with
      ###Context while the AI generated answer will begin with ###Answer.
(169) Evaluation criteria:
(170) The task is to judge the extent to which the metric is followed by the answer.
(171) 1 - The metric is not followed at all
(172) 2 - The metric is followed only to a limited extent
(173) 3 - The metric is followed to a good extent
(174) 4 - The metric is followed mostly
(175) 5 - The metric is followed completely
(176) Metric:
(177) The answer should be derived only from the information presented in the context
(178) Instructions:
(179) 1. First write down the steps that are needed to evaluate the answer as per the metric.
(180) 2. Give a step-by-step explanation if the answer adheres to the metric considering the
(181) question and context as the input.
(182) 3. Next, evaluate the extent to which the metric is followed.
(183) 4. Use the previous information to rate the answer using the evaluation criteria and assign a
(184) score.
(185) Return only the Score in last in a dictionary format not json and score should be in the (186)
range of 1 to 5.
(187) Example {groundedness_score:4}
(188) ""
(189) relevance_rater_system_message = ""
(190) You are tasked with rating AI generated answers to questions posed by users.
(191) You will be presented a question, context used by the AI system to generate the answer (192)
and an AI generated answer to the question.
(193) In the input, the question will begin with ###Question, the context will begin with
      ###Context while the AI generated answer will begin with ###Answer.
(194) Evaluation criteria:
(195) The task is to judge the extent to which the metric is followed by the answer.
(196) 1 - The metric is not followed at all
(197) 2 - The metric is followed only to a limited extent
(198) 3 - The metric is followed to a good extent
(199) 4 - The metric is followed mostly
(200) 5 - The metric is followed completely
(201) Metric:
(202) Relevance measures how well the answer addresses the main aspects of the question, based
(203) on the context.
(204) Consider whether all and only the important aspects are contained in the answer when (205)
evaluating relevance.
(206) Instructions:
(207) 1. First write down the steps that are needed to evaluate the context as per the metric.
(208) 2. Give a step-by-step explanation if the context adheres to the metric considering the (209)
question as the input.
(210) 3. Next, evaluate the extent to which the metric is followed.
(211) 4. Use the previous information to rate the context using the evaluation criteria and assign a
(212) score.
(213) Return only the Score in last in a dictionary format not json and score should be in the (214)
range of 1 to 5.
(215) Example {relevance_score:4}
(216) ""
```

```
(217) user_message_template = ""
(218) ###Question
(219) {question}

(220) ###Context
(221) {context}

(222) ###Answer
(223) {answer}
(224) ""
```

Appendix M

Function to evaluate groundedness and relevance of the answers based on context using AI LLM

```
(225) def generate_ground_relevance_response(user_input,response,
(226) max_tokens=500,temperature=0,top_p=0.95):
(227)     global qna_user_message_template
(228)     context_for_query = [doc.page_content for doc in
(229) retriever.get_relevant_documents(user_input, k=5)]
(230)         # Combine user_prompt and system_message to create the prompt
(231)     groundedness_prompt = f""[INST]{groundedness_rater_system_message}\n
(232)         {'user': {user_message_template.format(context=context_for_query,
(233) question=user_input, answer=response)}
(234)         [/INST]""
(235)         # Combine user_prompt and system_message to create the prompt
(236)     relevance_prompt = f""[INST]{relevance_rater_system_message}\n
(237)         {'user': {user_message_template.format(context=context_for_query,
(238) question=user_input, answer=response)}
(239)         [/INST]""
(240)     response_1 = client.chat.completions.create(
(241)         model="gpt-4o",
(242)         messages=[
(243)             {"role": "user", "content": groundedness_prompt}
(244)         ],
(245)         max_tokens=max_tokens,
(246)         temperature=temperature,
(247)         top_p=top_p
(248)     )
(249)     response_2 = client.chat.completions.create(
(250)         model="gpt-4o",
(251)         messages=[
(252)             {"role": "user", "content": relevance_prompt}
(253)         ],
(254)         max_tokens=max_tokens,
(255)         temperature=temperature,
(256)         top_p=top_p
(257)     )
(258)     return response_1.choices[0].message.content,response_2.choices[0].message.content
```

Appendix N

Evaluation of RAG-enabled response and storing the score

```
(257) # Question 1
(258) llm_judge_rag_ground_1, llm_judge_rag_rel_1 =
(259) generate_ground_relevance_response(user_input=result_dataframe.questions[0],
(260) response=result_dataframe.responses_with_RAG[0], max_tokens=500)
(261) print(llm_judge_rag_ground_1, end="\n\n")
(262) print(llm_judge_rag_rel_1)
(263) # Question 2
(264) llm_judge_rag_ground_2, llm_judge_rag_rel_2 =
(265) generate_ground_relevance_response(user_input=result_dataframe.questions[1],
(266) response=result_dataframe.responses_with_RAG[1], max_tokens=500)
(267) print(llm_judge_rag_ground_2, end="\n\n")
(268) print(llm_judge_rag_rel_2)
(269) # Question 3
(270) llm_judge_rag_ground_3, llm_judge_rag_rel_3 =
(271) generate_ground_relevance_response(user_input=result_dataframe.questions[2],
(272) response=result_dataframe.responses_with_RAG[2], max_tokens=500)
(273) print(llm_judge_rag_ground_3, end="\n\n")
(274) print(llm_judge_rag_rel_3)
```

Store the Score

```
                                     # Create a DataFrame to store the RAG response evaluation results
(275) rag_response_evaluation_df = pd.DataFrame({
(276)     "question": [question_1, question_2, question_3],
(277)     "rag_response": [result_dataframe.responses_with_RAG[0], result_
(278) dataframe.responses_with_RAG[1], result_dataframe.responses_with_RAG[2]],
(279)     "groundedness_score": [llm_judge_rag_ground_1[-2], llm_judge_rag_ground_2[-2], (280)
llm_judge_rag_ground_3[-2]],
(281)     "relevance_score": [llm_judge_rag_rel_1[-2], llm_judge_rag_rel_2[-2],
(282) llm_judge_rag_rel_3[-2]]
(283) })
(284) rag_response_evaluation_df['groundedness_score'] =
(285) pd.to_numeric(rag_response_evaluation_df['groundedness_score'], errors='coerce')
(286) rag_response_evaluation_df['relevance_score'] =
(287) pd.to_numeric(rag_response_evaluation_df['relevance_score'], errors='coerce')
                                     # Display the DataFrame
(288) display(rag_response_evaluation_df)
```

Appendix O

Evaluation of direct AI LMM response and storing the score

```
(289) # Question 1
(290) llm_judge_base_ground_1, llm_judge_base_rel_1 =
(291) generate_ground_relevance_response(user_input=question_1,
(292) response=result_dataframe.base_prompt_responses[0])
(293) print(llm_judge_base_ground_1, end="\n\n")
(294) print(llm_judge_base_rel_1)
(295) # Question 2
(296) llm_judge_base_ground_2, llm_judge_base_rel_2 =
(297) generate_ground_relevance_response(user_input=result_dataframe.questions[1],
(298) response=result_dataframe.base_prompt_responses[1], max_tokens=500)
(299) print(llm_judge_base_ground_2, end="\n\n")
(300) print(llm_judge_base_rel_2)
(301) # Question 3

(302) llm_judge_base_ground_3, llm_judge_base_rel_3 =

(303) generate_ground_relevance_response(user_input=result_

(304) dataframe.questions[2], response=result_dataframe.base_prompt_responses[2],

(305) max_tokens=500)
(306) print(llm_judge_base_ground_3, end="\n\n")
(307) print(llm_judge_base_rel_3)
```

Store the Score

```
                                # Create a DataFrame to store the base prompt evaluation results
(308) base_prompt_evaluation_df = pd.DataFrame({
(309)     "question": [question_1, question_2, question_3],
(310)     "base_prompt_response": [result_dataframe.base_prompt_responses[0], result_
(311) dataframe.base_prompt_responses[1], result_dataframe.base_prompt_responses[2]],
(312)     "groundedness_score": [llm_judge_base_ground_1[-2], llm_judge_base_ground_2[-2], (313)
llm_judge_base_ground_3[-2]],
(314)     "relevance_score": [llm_judge_base_rel_1[-2], llm_judge_base_rel_2[-2],
(315) llm_judge_base_rel_3[-2]]
(316) })
(317) base_prompt_evaluation_df['groundedness_score'] =
(318) pd.to_numeric(base_prompt_evaluation_df['groundedness_score'], errors='coerce')
(319) base_prompt_evaluation_df['relevance_score'] =
(320) pd.to_numeric(base_prompt_evaluation_df['relevance_score'], errors='coerce')

                                # Display the DataFrame
(321) display(base_prompt_evaluation_df)
```