Integrating Advanced Language Models and Vector Database for Enhanced AI Query Retrieval in Web Development

Xiaoli Huan[1], Hong Zhou[2]
Department of Computer Science, Troy University, Troy, Alabama, USA[1]
Department of Mathematics and Computer Science, University of Saint Joseph, West Hartford, Connecticut, USA[2]

***Abstract*—In the dynamic field of web development, the integration of sophisticated AI technologies for query processing has become increasingly crucial. This paper presents a framework that significantly improves the relevance of web query responses by leveraging cutting-edge technologies like Hugging Face, FAISS, Google PaLM, Gemini, and LangChain. We explore and compare the performance of both PaLM and Gemini, two powerful LLMs, to identify strengths and weaknesses in the context of web development query retrieval. Our approach capitalizes on the synergistic combination of these freely accessible tools, ultimately leading to a more efficient and user-friendly query processing system.**

***Keywords*—LLM (Large Language Model); vector databases; retrieval-augmented generation**

I. Introduction

In the rapidly evolving landscape of web development, the quest for efficient and accurate query retrieval systems has become a cornerstone of enhancing user experience and information accessibility. While effective to a certain extent, traditional query processing methods often fall short in coping with the complexity and dynamism of user-generated queries in real-time web environments.

Generative AI, including models like the GPT series [1], Google PaLM, and Gemini, have demonstrated remarkable capabilities in generating human-like text and answering queries in a contextually relevant manner. These models leverage large-scale transformer architectures to understand and generate complex language, making them highly suitable for sophisticated query processing tasks.

Retrieval-Augmented Generation (RAG) [2] is a cutting-edge approach that combines retrieval-based and generative models to enhance the accuracy and relevance of responses. RAG models retrieve relevant documents or pieces of information from a database and use these as context to generate more precise and contextually aware answers. This technique has been particularly effective in scenarios where the generative model alone might lack the necessary contextual knowledge to provide accurate responses [3].

Our approach integrates freely accessible tools like Hugging Face [4], FAISS [5], Google PaLM [6], Gemini [7], and LangChain [8]. Each tool brings its strengths to the table, contributing to a more robust query processing framework.

We explore and compare the performance of both PaLM and Gemini, two powerful Large Language Models (LLMs), to identify which is more effective in the context of web development query retrieval. This comparative analysis provides valuable insights into the strengths and weaknesses of each model for this specific task. By combining these cost-free technologies, we create a query processing system that is not only more efficient but also delivers significantly more relevant responses to user queries. This cost-effectiveness allows for the development of sophisticated AI-driven solutions without the burden of API usage fees or proprietary restrictions.

This research contributes novel insights to web development by:

- Highlighting the potential of combining sophisticated open-source AI models and advanced methodologies like RAG for improved user query handling.
- Providing a comparative analysis of PaLM and Gemini, offering valuable insights into their effectiveness for web development query retrieval.
- Emphasizing accessibility and cost-effectiveness through the utilization of freely available tools.

The following sections will delve into the technical architecture, implementation details, and performance evaluation of the system in Sections II, III, IV, V and VI, providing a comprehensive understanding of its capabilities and potential impact on the future of web development in Section VII.

II. Evolution of Language Models

The evolution of language models in query processing is crucial in natural language processing (NLP) and artificial intelligence (AI), witnessing significant advancements over the past few decades. This section explores the trajectory of these developments, focusing on how they have revolutionized query processing and understanding.

The journey began with early language models like n-gram models and statistical language models. These models, such as those used in early versions of machine translation and speech recognition systems, relied heavily on statistical probabilities of word sequences. However, their major limitation was the inability to capture long-range dependencies and contextual nuances in language, leading to suboptimal performance in complex query processing [9].

The introduction of neural network-based models marked a significant shift. Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks began to address the shortcomings of traditional models by better capturing sequential information and context [10]. Despite their improvements, these models still struggled with processing longer sequences and required substantial computational resources.

The introduction of transformer architecture [11] in 2017 marked a significant transformation in language modeling. Distinct from earlier models, the transformer employs self-attention mechanisms to analyze entire text sequences at once, allowing for more effective context capturing. This innovative framework serves as the foundational structure for models such as Google's BERT (Bidirectional Encoder Representations from Transformers) and the GPT (Generative Pre-trained Transformer) series developed by OpenAI.

BERT [12] was groundbreaking due to its bidirectional training, enabling it to comprehend the context of a word by considering all of its surrounding words. This feature made it particularly effective for tasks like question answering and language inference.

The GPT series demonstrated remarkable capabilities in generating human-like text and answering queries in a contextually relevant manner. Its large-scale transformer model, trained on vast amounts of data, could generate coherent and contextually relevant text over extended passages.

The most recent advancements, such as Google's PaLM (Pathways Language Model) and Gemini, have pushed the boundaries further. PaLM, with its even larger scale and more sophisticated training, has shown capabilities in not just understanding but also generating complex and nuanced language, making it highly effective for sophisticated query processing tasks. On the other hand, Gemini showcases strength in its multimodality, seamlessly processing text, images, and code. This versatility could prove advantageous in web development scenarios where queries might incorporate screenshots or snippets of code alongside textual information.

The impact of these advancements on query processing has been profound. Language models have transitioned from simply predicting the next word in a sequence to understanding and generating human-like responses to complex queries. This evolution has enabled the development of more sophisticated AI-driven applications, such as virtual assistants, chatbots, and advanced search engines, capable of understanding and responding to user queries with unprecedented accuracy and relevance.

III. Vector Databases

Integrating vector databases in information retrieval significantly advances AI and web development. This section reviews the evolution and application of vector databases, particularly focusing on their role in enhancing information retrieval capabilities in AI systems.

Vector databases, fundamentally different from traditional relational databases, are designed to store and retrieve high-dimensional vector data efficiently. This capability is crucial in handling the outputs of advanced AI models, especially in the context of natural language processing and machine learning (ML). The early conceptualization and use of vector spaces in information retrieval set the stage for the development of these databases [13].

Developing technologies like FAISS (Facebook AI Similarity Search) marked a significant milestone in vector databases. FAISS is designed for efficient similarity search and clustering of dense vectors. Its ability to handle billions of vectors makes it particularly suitable for large-scale AI applications, including those in web development and query processing [14].

Vector databases have found extensive application in AI-driven systems, particularly in enhancing the efficiency and accuracy of information retrieval. For instance, their integration into recommendation systems and search engines has significantly improved the relevance of results based on user queries and preferences [15].

Despite their advancements, vector databases face challenges, particularly in scalability and real-time processing in web environments. Future research is directed toward optimizing these databases for more efficient real-time query processing and integration with evolving AI models [16].

IV. System Architecture

This section outlines the system architecture of our AI-driven web application, emphasizing the integration of Streamlit and various AI components to enhance user experience and query processing efficiency.

The front-end of our system is built using Streamlit [17], an innovative framework that allows for rapid development and deployment of data applications. Streamlit's simplicity and efficiency make it an ideal choice for integrating complex AI models into web applications.

Key components of the front-end include:

- User Interaction: It presents a web interface where users can type questions in a text box.
- Visual Elements: Streamlit elements like headers, subheaders, and text boxes are used to create a user-friendly interface.
- Feedback Mechanism: The script provides feedback to the user by displaying placeholders that change state based on the application's progress. Initially, it displays "Awaiting your question..." and transitions to "Processing..." when the user submits a question. Finally, it displays "Answer" if a successful response is generated or an error message if the process fails.
- Communication: The front-end interacts with the back-end by passing the user's question as input and displaying the generated response.

Fig. 1 shows the web application's initial input interface.

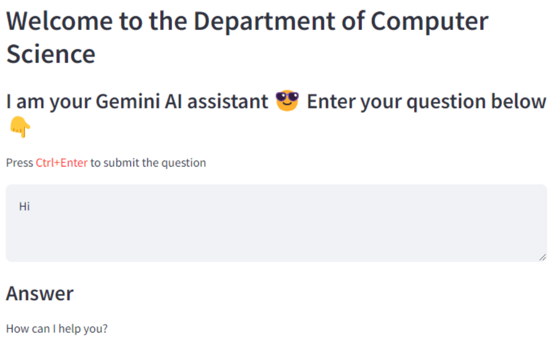

Fig. 1. The web application interface.

The front-end acts as the communication layer for users, while the back-end handles the core functionalities of the application. It serves as the engine that processes user queries, interacts with the data and AI models, and generates the final response.

In Fig. 2, the init_google_palm_model() function initializes the Google PaLM model. It retrieves an API key [18], which is used to authenticate Google's services. The model is initialized with a temperature parameter set to 0.1, which influences the model's output's randomness (lower temperature values result in more deterministic outputs). The changes to the init_google_gemini_model function are relatively straightforward in leveraging Gemini's capabilities instead of PaLM. Import the necessary libraries for Gemini and modify the line that initializes the model. Instead of GooglePaLM, use ChatGoogleGenerativeAI and specify the model="gemini-pro" argument to indicate the Gemini model variant.

In Fig. 3, the init_hf_embeddings() function initializes embeddings from the Hugging Face's InstructEmbeddings [19] model. These embeddings can be used to convert text into numerical vectors, which represent the semantic meaning of the text and can be used for similarity search and vector-based analysis.

```
@st.cache_resource
def init_google_palm_model():
    api_key = st.secrets["GOOGLE_API_KEY"]
    return GooglePalm(google_api_key=api_key, temperature=0.1)


@st.cache_resource
def init_google_gemini_model():
    #api_key = st.secrets["GOOGLE_API_KEY"]
    genai.configure(api_key=st.secrets["GOOGLE_API_KEY"])
    return ChatGoogleGenerativeAI(model="gemini-pro",temperature=0.1)
```

Fig. 2. Functions to initialize PaLM and Gemini models.

```
@st.cache_resource
def init_hf_embeddings():
    return HuggingFaceInstructEmbeddings(model_name="hkunlp/instructor-large")
```

Fig. 3. The init_hf_embeddings() Function.

```
@st.cache_resource
def setup_vector_database(file_path="faiss_index"):
    csv_loader = CSVLoader(file_path='prompt_answer.csv', source_column="prompt")
    faq_data = csv_loader.load()
    hf_embeddings = init_hf_embeddings()
    faiss_db = FAISS.from_documents(documents=faq_data, embedding=hf_embeddings)
    faiss_db.save_local(file_path)
```

Fig. 4. The setup_vector_database() Function.

The setup_vector_database() function in Fig. 4 sets up a vector database using FAISS. It loads question-and-answer pairs from a CSV file (prompt_answer.csv) using LangChain's CSVLoader. The function then initializes the Hugging Face embeddings (init_hf_embeddings function) to convert the loaded FAQ data into vector embeddings. These embeddings are then used to create a FAISS database with the FAISS.from_documents method. Finally, the FAISS database is saved locally using the provided file path, allowing the application to quickly retrieve relevant answers based on similarity searches in the future.

Fig. 5 shows samples of the provided data file, 'prompt_answer.csv'. The dataset comprises a collection of questions and corresponding responses intended for the computer science department. This spreadsheet is structured with two primary columns: prompt and response. The prompt column contains a variety of questions that might be asked by students, faculty, or other stakeholders, while the response column provides the appropriate answers. The data can be easily updated to reflect new queries or changes in the information provided, ensuring that the dataset remains current and helpful in addressing the diverse inquiries directed towards the CS department.

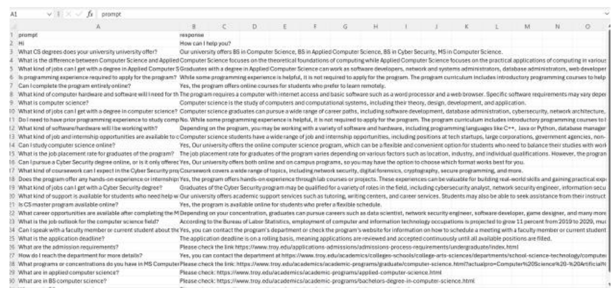

Fig. 5. The prompt_answer.csv data file.

After the front-end invoked the setup_qa_chain() function and passed the user question, Fig. 6's setup_qa_chain() function first initializes embeddings using the function init_hf_embeddings. These embeddings are then used to load a local FAISS database from the specified file path generated by the setup_vector_database() function.

Fig. 6. The setup_qa_chain() function.

In line 45 of Fig. 6, a retriever object is created from the FAISS database with a score threshold of 0.7, meaning it will only consider results that meet or exceed this similarity score. This retriever is used to fetch relevant context for incoming questions.

The code then defines a prompt_template, which is a structured text template for generating prompts to be used with a large language model. Finally, the function initializes a LLM (Google PaLM or Gemini model) and generates a QA chain that takes a query as input, uses the retriever to fetch relevant context based on the query, and then generates an answer using the LLM and the structured prompt.

Throughout the program, robust error-handling mechanisms are in place to manage potential failures and processing errors.

V. Technical Details

Google PaLM or Google Gemini models generate responses to user queries. The models have customized behavior (e.g., setting temperature) to tailor the response generation. At a high level, temperature controls the likelihood distribution over words or tokens the model might select at each step in generating text. A lower temperature makes the model more confident and conservative in its choices, leading to more predictable text. A higher temperature increases randomness, making the model more likely to produce varied and sometimes more creative or less likely outputs. Choosing the right temperature is a balancing act: too low, and the model might generate dull, repetitive text; too high, and its outputs might become too random and less coherent. The optimal setting often depends on the specific application and desired user experience. For a technical query retrieval system, a slightly lower temperature might be preferred to ensure the reliability and relevance of the information provided.

Langchain is a backbone that connects various AI and machine learning components, ensuring seamless interaction. The library assembles components (like the Google PaLM or Gemini language model, HuggingFaceInstructEmbeddings, and the FAISS vector database) into a cohesive QA chain. This chain orchestrates the process of receiving a query, processing it through the model, and fetching relevant answers. The CSVLoader component in LangChain is used to load data from CSV files. The CSV file contains two columns: one for the prompts (or questions) and another for the corresponding answers or information. These pairs are used to build a knowledge base for the FAISS vector database, allowing the system to retrieve relevant answers based on the embeddings generated from user queries.

The system uses the following prompt_template: "Please provide an answer to the question below, ensuring that your response is derived solely from the provided context. Focus on using the text from the 'response' section of the source document, altering it as little as possible. If the context does not contain the information necessary to answer the question, simply reply with 'I am not sure. Please call: 1-334-808-6576' to avoid creating or inferring any information not explicitly stated in the context. Exceptions: Answer all computer science questions using your own knowledge and give tutorials and explain in detail".

The model is instructed to base its responses solely on the provided context, specifically focusing on the ‘response’ section of the source documents. This restriction ensures factual accuracy and reduces the risk of the model generating misleading or fabricated information (hallucination). Prompt engineering, which involves carefully crafting the input text, helps achieve this objective. When the context lacks sufficient information to answer a department question, the model is instructed to provide a clear default message ("I am not sure..."). This transparency helps manage user expectations and prevents frustration in cases where a definitive answer cannot be found.

There is also a crucial exception for computer science questions. In these cases, instead of being confined solely to the provided department question context, the model can draw on its own knowledge base to answer computer science questions. Comprehensive tutorials and detailed explanations for computer science topics can be delivered. This exception caters to students seeking deeper understanding and learning resources beyond basic department questions in the CSV file.

The PaLM Model app can be accessed at this link: https://troy-cs-ai-assistant.streamlit.app/.

The Gemini Model app can be accessed at this link: https://troy-ai-gemini.streamlit.app/.

VI. Results analysis

Fig. 7 shows a programming tutorial interface facilitated by an AI PaLM assistant, exemplifying an interactive learning environment. It features a question-and-answer dialogue where the user inquiry, "Can you give me C++ tutorial?" is met with a comprehensive response from the AI assistant. The AI assistant outlines C++'s applicability in system, application, and game development, and offers resources for further learning. This exchange is encapsulated in a clean, structured layout, promoting an engaging and educational user experience.

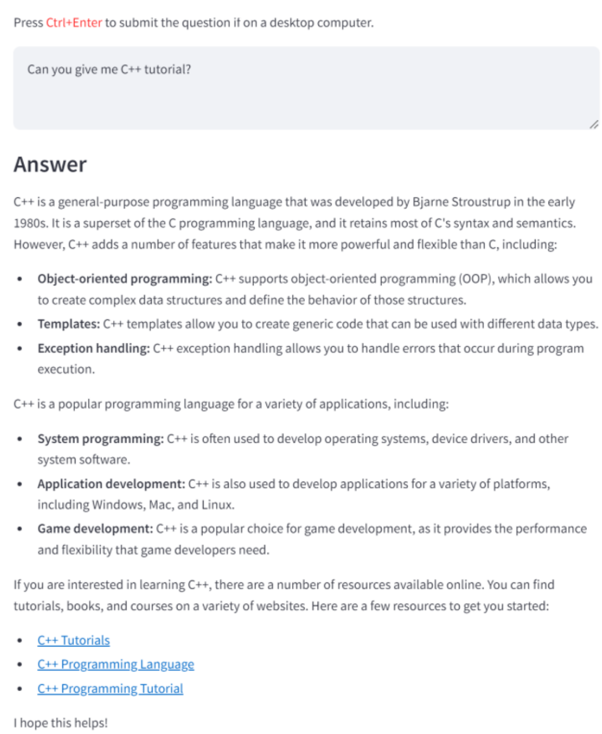

Fig. 7. Interactive Programming Tutorial via AI PaLM Assistant.

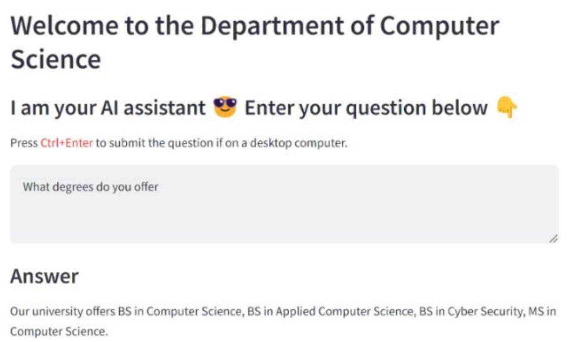

Fig. 8. Departmental Inquiry Response by AI Gemini Assistant.

Fig. 8 illustrates an AI Gemini assistant answering a prospective student's inquiry, "What degrees do you offer?" The AI assistant provides a detailed response listing the degrees offered by the university, facilitating ease of information retrieval for prospective students or interested parties.

Forty unique questions were used to test the PaLM and Gemini models, assessing their accuracy and capabilities in answering questions such as the cost of the master's program, Python tutorials, identifying the faculty of computer science, differences between Python and C++, potential jobs post-graduation, and details about the degree concentrations.

Performance:

1) PaLM demonstrated better accuracy, correctly answering 37 out of 40 questions. For example, when asked, "Can I get a bachelor's degree online?" PaLM provided a clear and informative response, highlighting the university's online program and its benefits.

2) In contrast, Gemini answered 32 questions correctly. While some answers were accurate, others showed limitations, like the response to "Can I get a bachelor's degree online?". In this instance, Gemini offered uncertainty and directed the user to contact the department or instructor, potentially hindering a user's ability to get a quick answer. This difference in handling specific questions highlights areas for improvement in Gemini's knowledge base and its ability to provide definitive answers.

3) User feedback indicates response time differences between the PaLM and Gemini models. Users report that the PaLM web application loads noticeably faster in both Safari and Google Chrome browsers.

Contextual Understanding and Prompting:

1) PaLM excels at the provided prompt and adheres to strict guidelines. The prompt template specifically instructs the model to "give tutorials and explain in detail" for computer science questions. PaLM effectively obeys this direction, retrieving relevant information from its knowledge base.

2) Gemini's strength lies in its multimodality. It can handle various data formats like text, images, and code. This versatility makes it a good candidate for web development scenarios where user queries might incorporate diverse information, such as screenshots or code snippets alongside textual content. However, Gemini is less adept when working with pre-defined CSV data for tasks requiring strict adherence to specific prompts. Its focus might shift towards pattern matching within the CSV content, potentially leading to less comprehensive responses or failing to follow the prompt's instructions. This could be particularly true for tasks requiring in-depth explanations or generating new content, such as computer science tutorials. However, in domains like the medical field, where factual accuracy and information retrieval from structured data are crucial, Gemini's ability to handle diverse formats and find relevant patterns could prove advantageous.

Data Restriction:

1) Both models rely heavily on the information stored in the CSV files. Gemini's ability to generate content would be inherently limited if those files lack rich information or detailed explanations for departmental or computer science concepts.

2) PaLM might be better at drawing from external knowledge sources or generating explanations from its own understanding, even if the CSV data is somewhat limited.

Table I compares the performance of PaLM and Gemini on a set of sample questions. The table categorizes the questions into three sections: scenarios where both systems failed due to limitations set by the prompt, instances where PaLM provided more accurate or helpful responses, and examples where Gemini outperformed PaLM. The examples highlight the strengths and weaknesses of each system in handling specific types of inquiries, such as basic greetings, specific program information, and technical support questions.

TABLE I. MODEL PERFORMANCE COMPARISON ON SAMPLE QUESTIONS

The sample question that neither PaLM nor Gemini can address due to the prompt restricting responses to information outside the CSV data file:
Q: What is the best website to find sports data
PaLM: I am not sure. Please call 1-334-808-6576…
Gemini: I am not sure. Please call 1-334-808-6576…
Examples of questions where PaLM performed better than Gemini:
Q: Hello?
PaLM: Hello, how can I help you?
Gemini: I am not sure. Please call: 1-334-808-6576…
Q: How much does the masters of computer science program cost
PaLM: Please check: https://www.troy.edu/scholarships-costs-aid/costs/
Gemini: I am not sure. Please call: 1-334-808-6576…
Q: How big are the graduate classes?
PaLM: The average class size for graduate computer science courses at Troy University is approximately 20 students.
Gemini: I am not sure. Please call: 1-334-808-6576…
Q: Can you give Python tutorials?
PaLM: Python is a general-purpose programming language that is designed to be easy to read and write…
Gemini: I am not sure. Please call: 1-334-808-6576…
Examples of questions where Gemini outperformed PaLM:
Q: What are the admission requirements for graduate students?
PaLM: To be considered for admission to a graduate program at Troy University, you must meet the following requirements…
Gemini: please check: https://www.troy.edu/applications-admissions/admissions-process-requirements/graduate/index.html

Q: Where can I find additional help on programming
PaLM: An error occurred: list index out of range
Gemini: You can contact the course instructors, tutors or use resources such as chatGPT or other generative AI (GenAI).

VII. Conclusion and Future Work

This paper presented a framework that integrates advanced AI models and vector databases to enhance the effectiveness of query retrieval in web development significantly. Our system leverages freely available tools, making it cost-effective and accessible for developers. The comparative analysis between PaLM and Gemini revealed their unique strengths: PaLM can learn from a few examples, which might be helpful for limited datasets. In contrast, Gemini focuses on factual accuracy and aims to reduce factual errors and hallucinations. Future work will involve comprehensive testing and evaluation of the system's performance across diverse user scenarios to ensure scalability and robustness. The system's effectiveness is highly dependent on the quality and comprehensiveness of the CSV data. Future work will explore techniques for continuous knowledge base improvement. Strategies for automatic data augmentation, user feedback integration, and potentially incorporating external knowledge sources to enrich the information available to the LLMs can be considered.

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal and A. Neelakantan, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2020, pp. 1877--1901.

[2] P. Lewis, E. Perez, A. Piktus, F. Petroni and V. Karpukhin, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

[3] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen and W. Yih, "Dense Passage Retrieval for Open-Domain Question Answering," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 2020.

[4] "hkunlp/instructor-large," Hugging Face, [Online]. Available: https://huggingface.co/hkunlp/instructor-large. [Accessed 2024].

[5] "Facebook AI Similarity Search," Meta, [Online]. Available: https://ai.meta.com/tools/faiss/. [Accessed 2024].

[6] A. Chowdhery, S. Narang, J. Devlin and M. Bosma, "PaLM: scaling language modeling with pathways," *The Journal of Machine Learning Research*, vol. 24, no. 1, p. 11324–11436, 2022.

[7] S. Pichai and D. Hassabis, "Introducing Gemini: our largest and most capable AI model," 6th December 2023. [Online]. Available: https://blog.google/technology/ai/google-gemini-ai/#sundar-note.

[8] "Applications that can reason. Powered by LangChain.," [Online]. Available: https://www.langchain.com/. [Accessed 2024].

[9] D. Jurafsky and J. H. Martin, Speech and language processing, 2nd ed., Prentice Hall, 2009.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

[12] J. Devlin, M. W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[13] G. Salton, A. Wong and C. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, p. 613–620, 1975.

[14] J. Johnson, M. Douze and H. Jégou, "Billion-Scale Similarity Search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535 - 547, 2019.

[15] Y. Wang, X. Chen, J. Fang, Z. Meng and S. Liang, "Enhancing Conversational Recommendation Systems with Representation Fusion," *ACM Transactions on the Web*, vol. 17, no. 1, pp. 1-34, 2023.

[16] Y. Han, C. Liu and P. Wang, "A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge," arxiv.org, [Online]. Available: https://arxiv.org/pdf/2310.11703.

[17] "A faster way to build and share data apps," [Online]. Available: https://streamlit.io/. [Accessed 2024].

[18] "Get API key," Google, [Online]. Available: https://aistudio.google.com/app/apikey. [Accessed 2024].

[19] "Instruct Embeddings on Hugging Face," [Online]. Available: https://python.langchain.com/v0.1/docs/integrations/text_embedding/instruct_embeddings/. [Accessed 2024].