CodeQA: Advanced Programming Question-Answering using LLM Agent and RAG

Mohamed Ahmed*, Mostafa Dorrah*, Ahmed Ashraf*,Yousef Adel*, Abdelrahman Elatrozy*, Bahaa Eldin Mohamed*, Wael Gomaa*
School of Information Technology and Computer Science (ITCS),
Nile University, Giza, Egypt
{ moham.mohamed, M.Samer, ahme.ashraf, Y.Khalil, A.Elatrozy, B.Moustafa, wabouzed}@nu.edu.eg

2024 6th Novel Intelligent and Leading Emerging Sciences Conference (NILES) | 979-8-3503-7851-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/NILES63360.2024.10753267

***Abstract*—In light of this complex information for programming environments, this paper explores how effectiveness in Large Language Models and concepts from Retrieval-Augmented Generation can be used to augment reduced hallucinated question-answering systems for programming environments. The present scenario of transformer-based models, though a big player in natural language processing, doesn't seem to have adaptability and context sensitivity, which is, in fact, a very important feature for a specialist domain. Our study, by integrating a sophisticated LLM model within the RAG framework, further improves the precision, effectiveness of retrieval, and sensitivity of the context of responses. The model based on LLM improves over this transformer-based model by scoring higher in accuracy and context awareness, supported by its pre-training on a massive corpus and dynamic document retrieval functionality. This result underlines the possibility of great improvement in QAS performance, handling very complex and specialized queries, through the integration of LLMs with the RAG systems, and thereby helps in indicating promising areas for further research in optimizing these methodologies across different domains.**

***Index Terms*—LLM, Transformers, Programming, Python, QAS, RAG**

I. Introduction

The expansion of programming languages and the growth of their ecosystems call for strong mechanisms to help developers navigate documentation and online resources. Traditional question-answering systems (QAS) built on transformer architectures and underlying information retrieval techniques provide the basic approaches to handle queries either through pre-defined answers or by extracting the answer from a given context with sufferers from a lot of hallucinations. However, these methods—developed with the rise of information quantity and complexity—fall short because they are not adaptive or context-sensitive. What Large Language Models (LLMs) have brought into the QA domain is this new twist in the story, which holds out the promise of higher performance by using very large pre-trained knowledge bases and dynamic retrieval mechanisms.

Transformer-based models, for example, BERT [1] and GPT-2 [2], have resulted in a breakthrough in applications in the domain of NLP, effectively covering sequence-to-sequence tasks with the salient mechanism of attention in their core [3]. Which is the kind of architecture that is well-suited in the implementation of a standard QA system in programming environments. The introduction of RAGs refines this approach further, allowing for external information to be directly embedded in the process of response generation and thus enhancing the quality of the response with more contextual information. also, reducing hallucinations. [4].

On the other hand, advanced QA methodologies, using LLMs, go further than that by integrating both in-depth pre-training on diversified corpora and fine-tuning on domain-specific datasets. This dual approach makes these models well-rounded in both the general understanding of languages and the unique nuances they might involve in programming-related content. RAFT [5], for example, improves the use of RAG through training models to recognize and exploit relevant documents, which are a core feature in working with often sparse and very technical data in programming documentation.

Our study seeks to test our workflow for reducing hallucinations in programming question-answering systems and try out the implementation of both methodologies and testing them, making analysis for the experiments and addressing the difference made. This study aims to identify the strengths and limitations within each by performance metrics for each approach along with accuracy, retrieval effectiveness, and context-sensitivity, hence providing insight toward practical applications and potential further improvements.

The rest of this paper is structured as follows: Section II highlights the works related to the topic of this research. Section III presents the methodology used along with the main components that have been carried out on it. Section IV lists the results of our model. Section V analyzes the results achieved. Section VI contains limitations and future work. Finally, Section VII concludes this research.

II. Related Work

The most recent breakthroughs in Question Answering systems have also come from domains as diverse as Programming QAs, brought about by the integration of machine learning, natural language processing, and information retrieval techniques. Several related systems are introduced in this section,

each one tackling different challenges with different requirements in mind. We present our primary summary of question-answering systems (QAS) for the programming domain using a taxonomy illustrated in Figure 1

PythonQAS [6] is a question-answering system that automatically answers questions in natural language over a set of Python programming text. The system would be designed to make interaction with the user effective, producing brief answers from a predefined domain, rather than lists of potentially relevant documents, as most conventional search engines generally do. This materializes a closed domain approach, structured databases, natural language processing techniques, syntax analysis, and keyword extraction in a fashion that can exceptionally deal with the requests in the posed domain. It converts the natural language questions into a database query in the process of getting relevant, concise answers. Although the system performed well in giving correct answers within its domain, it was hampered by the reliance on a prior knowledge in its base, which significantly narrowed the range of questions with which it could be equipped. Further work could be done to increase the database or the parse of algorithms, making the applicability and accuracy higher.

While APIBot [7], is a question-answering bot for information retrieval automation from the documentation on APIs and is designed to negate the huge inefficiency in manually looking through documents. The bot, therefore, is a further extension of the existing SiriusQA system, which entails multiple domain-specific adaptions to make it more suitable for software engineering contexts, such as custom question-and-answer patterns specific to APIs and enhanced document-handling capability regarding the structured nature of API documentation.

Their experiment results presented that APIBot had significantly improved performance, with a Hit@5 of 0.706 over the original SiriusQA system, to handle API-specific questions. However, the paper also brings to the fore the limitations on which structured API documentation depends and on what scope of questions the bot can understand and process.

Whereas Bansal, Eberhart, Wu, and McMillan [8] present a question-answering system tailored specifically for basic questions to develop conversational AI software engineering applications about subroutines in programming, which are indispensable for these cases. The system is based on an encoder-decoder neural network and is trained on a dataset of 1.56 million Java methods annotated to question-and-answer templates obtained by empirical studies. Automated metrics and user studies with professional programmers suggest the system is at least promising for natural language interpretation and generation of responses to queries in programming languages. We provide evidence it is feasible to include such a tool in more general dialogue systems but raise questions about how well the tool generalizes to broader, more diverse, or complex programming scenarios than those in the training data.

In contrast, Python-Bot [9] is a step in the direction of aiding the integration of artificial intelligence in education using a chatbot developed to aid novice programmers in learning Python. The current bot is created on the SnatchBot platform. The bot is oriented to raise the level of understanding of programs via interaction with the user in a conversation manner, explaining basic syntactic structures and proposing programming exercises. Experimental outcomes based on the use of Python-Bot indicate that the learning experience improves in both tailored assistance and engaging content. But, it works best only for common concepts of programming, since it is yet to be developed to teach much about advanced programming topics or other programming languages.

Subsequently, Yu, Gu, and Shen introduce CodeMaster [10], a novel approach that leverages the benefits of task-adaptive sequence-to-sequence pre-training to further enhance the code QA system. Given such a gap between pre-trained language models (PLMs) and the targeted downstream code QA tasks, CodeMaster builds upon the highly acknowledged PLM for code, CodeT5. It introduces multiple self-supervised learning tasks like Partial Comment Completion and Noun Phrase Prediction to better align the pre-training with the demands of Code QA. The approach presented above has significantly higher improvements over earlier models, with a gain of 80% in Exact Match and 24% in F1-score on the benchmark CodeQA. It pushes the state-of-the-art performance. The CodeMaster approach is very effective only if labeled data for pre-training are constantly available. This approach eludes the challenge of adaptability and scalability to diverse programming environments.

However, there exists Gorilla [11], an LLM connected with massive APIs, solving in this way what current state-of-the-art Large Language Models (LLMs), like GPT-4, need to address to create an effective API call with accuracy. The main challenge is in the creation of correct input arguments and the prevention of hallucinated API use. The authors fine-tune a model on top of LLaMA, integrating a document retriever with the ability to adapt to real-time changes in the document in order to minimize hallucinations. Using the full APIBench dataset, Gorilla is better able to perform in terms of both API call accuracy and adaptability than GPT-4. Although integrating retrieval systems is not trivial, it may not necessarily lead to better performance, revealing the complexity of balancing retrieval integration with model accuracy and response quality.

At the same time, Nakhod [12] introduces the Retrieval-Augmented Generation (RAG) model with the infusion of domain information into Large Language Models in its quest to upskill developers working with low-code. Being so important in low-code development, the model uses a vector database to store text representations of domain information and match stored information whenever matched with user queries, giving context-enhanced responses. This methodology employs cosine similarity in document retrieval and, therefore, results in higher quality and relevance to the response of an LLM. The results have shown that the RAG-based models have outperformed baseline LLMs in providing accurate and up-to-date information. The paper does admit to the limitation of its scope within the subject and suggests possible improvements

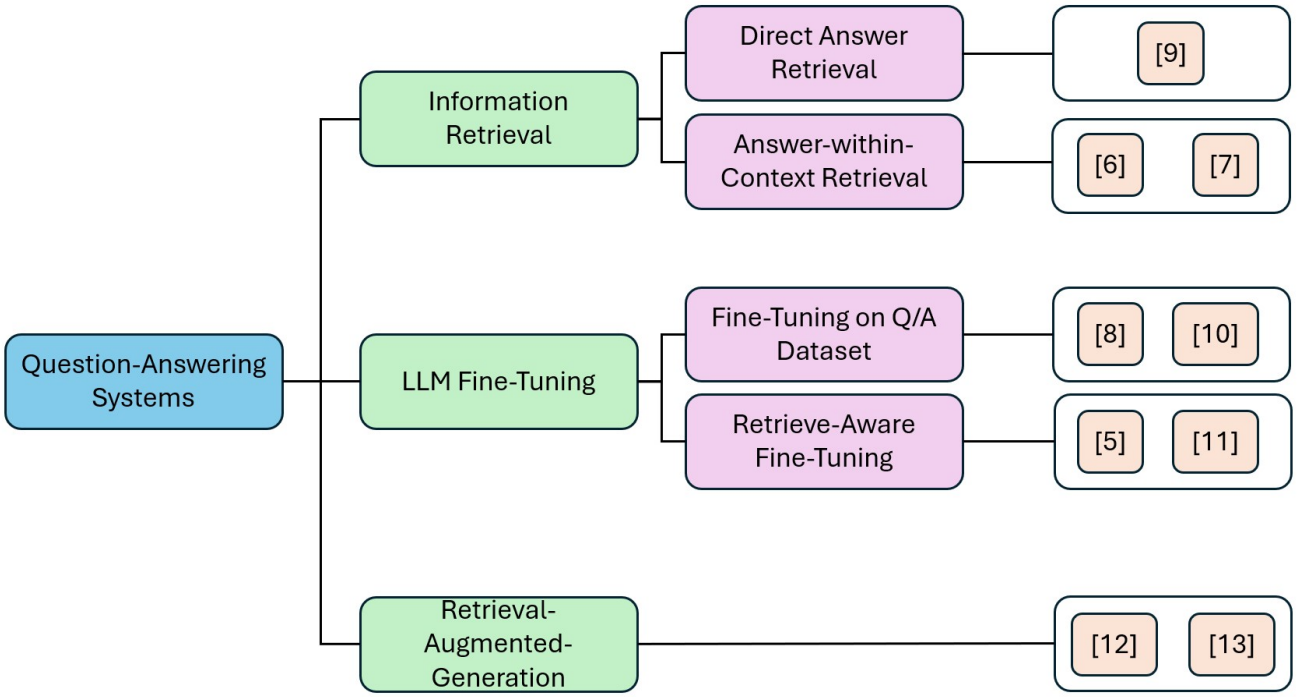

Fig. 1. A Taxonomy of QA Systems for Programming Domain.

with better data storage and retrieval techniques in the future.

Nevertheless, AI-TA [13], the most recent of these systems, tackles the challenge of fully automating responses to the questions that students post on systems like Piazza, in an attempt to alleviate the labor-intensive task of responding to thousands of questions every semester in rapidly expanding computing courses. This will be achieved by following current state-of-the-art methods, Retrieval-Augmented-Generation (RAG), together with Supervised Fine-Tuning (SFT), and Direct-Preference-Optimization (DPO), on top of open-source LLMs of the LLaMA-2 family. Actually, RAG would appear to contribute 30% on a dataset from an introductory CS course, leading to a significant increase in answer quality. In regard to system successes, it remains at the stages of structured input and manual fine-tuning, which are indicative of further automation and adaptation of the system to diverse educational environments as possible development prospects.

III. Methodology

Data collection & Pre-processing, and, Modeling are the three main components of our proposed solution. As shown in Figure 2, the workflow of our system architecture starts with the user asking a specific question, and then the agent decides whether that question is programming-related or not, if not, then the system produces no answer. If the question is programming-related, then the question is being compared with each document in our vector database by using cosine-similarity to produce a score for the similarity between the question and the documents and retrieving the top 3 relevant documents to the question in an order in which the most relevant document is on top and the least relevant document is in the bottom. After that, the agent observes the retrieved documents and if none of the retrieved documents contain the answer to the user's question then the agent decides to perform a web search with the user's question using the Tavily search tool and produces the top 3 relevant results from the internet and use it as a context for the question instead of the locally stored documents. Otherwise, the model would answer the question from the already retrieved local documents without the use of web search.

A. Data Collection and Pre-processing

For the corpus collection, we collected Python documentation and standard library documentation, using the BeautifulSoap library in Python. For the pre-processing, redundant white spaces, new lines, and special characters are cleaned, in addition to the irrelevant text with no contextual meaning for our scope. Resulting in 290 documents in text file format, each representing a specific library and topic in Python. Where the chunking operation is done on each document using the pre-defined NLTK text splitter, and then embedding is done to the documents using the multilingual-e5-base embedding model [14] to make the documents ready to store in a local vector database. Chroma db [15] vector database was used for storing the embedings. As seen in Figure 3 below, the following RAG algorithm is used in the proposed system architecture.

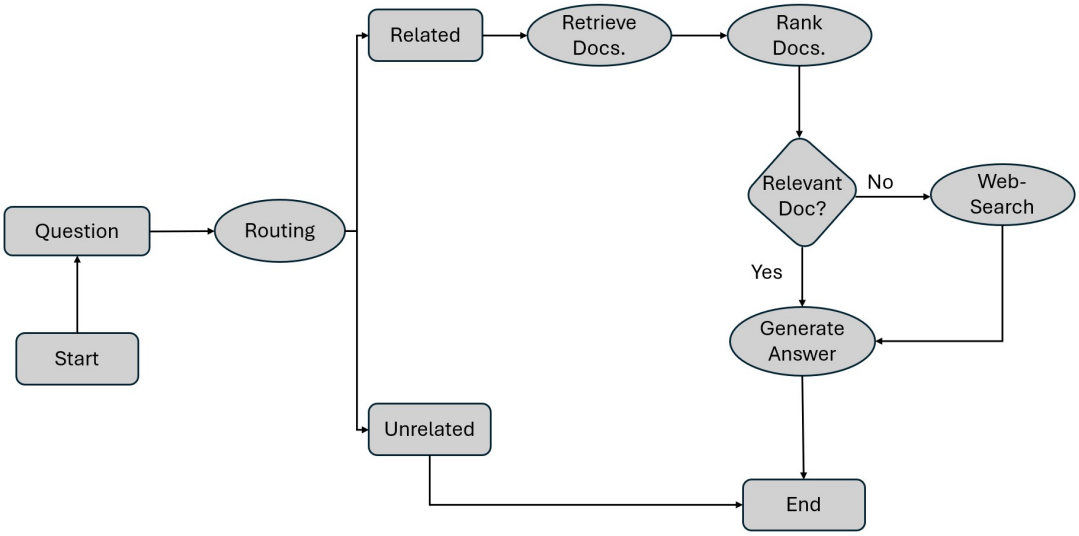

Fig. 2. Workflow of the proposed System Architecture.

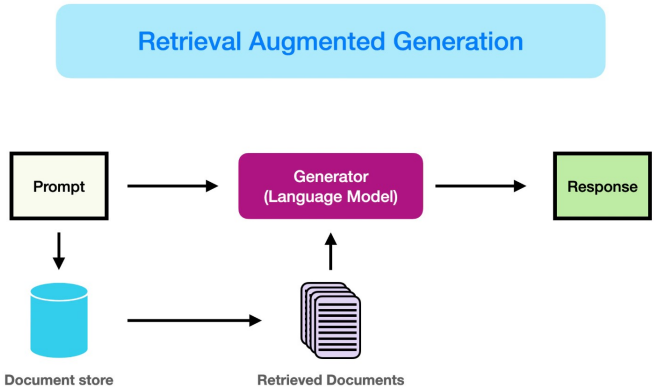

Fig. 3. [16]

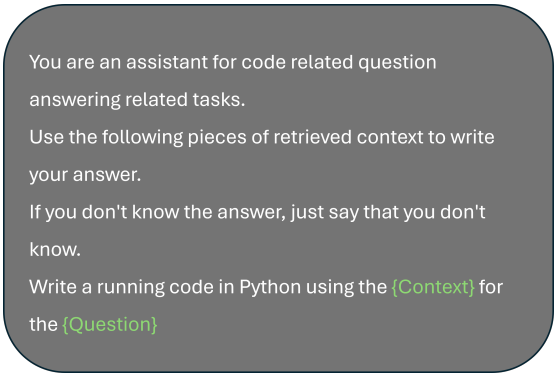

Fig. 4. Zero-shot inference prompt

B. Modeling

We performed a zero-shot learning technique on our models given a context (even retrieved from a vector database or using the search tool) in the prompt to make the model answer based on it. While doing the zero-shot learning, multiple prompts have been considered to make the model act as needed. The prompt as shown in figure 4 is used in defining that the LLM is an assistant for code-related question-answering. that Uses a context to respond to the input question. And to say ”don’t know” if the LLM didn’t know the answer.

Some prompts have been tested. the prompt above resulted in the best performance in our experiments. In this study, we attempted two experiments, the first experiment is with using a small transformer-based model called Flan-T5-Base [17], while the other experiment is with using a large language model called deepseek [18]. there is a significant difference between the 2 models as shown in figure 5

Noting that both experiments inherit the same system architecture, to make a fair comparison between the performance of two different models in terms of preciseness in using the given context to answer the user’s question.

IV. Results

To fairly evaluate each model, we have used a large language model Llama-70B [19] to evaluate our experiments. We also performed this LLM zero-shot learning to mimic the actor-critic technique. we have used a pre-defined prompt for evaluation as seen in figure 6.

To evaluate both models on questions regarding Python, we used a dataset called conala [20] and tested our models on 50 programming questions. we have fed each question and its answers in the prompt and sent it to the LLM to evaluate and give scores.

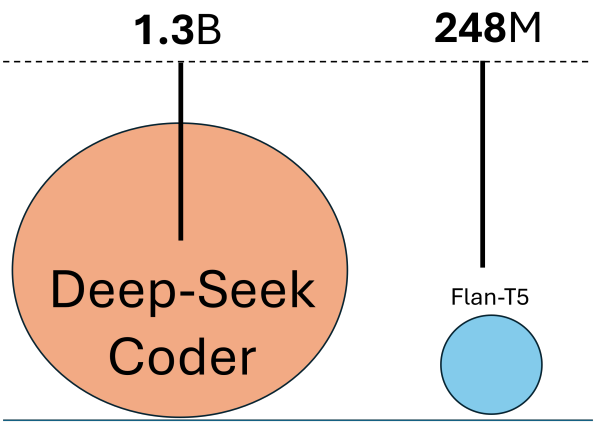

Fig. 5. Model's Size

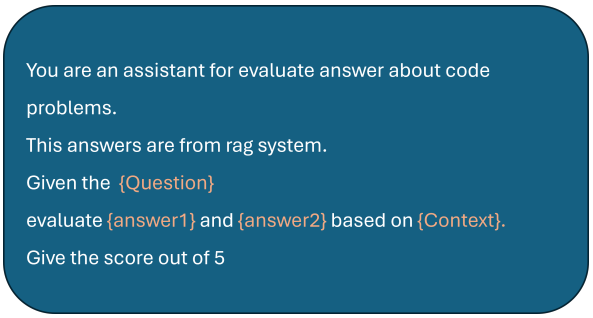

Fig. 6. Evaluation Prompt.

A score of 5 means the model answered the question correctly, while a score of 0 means the answer is unrelated or does not answer the user's question. After 50 questions, the transformer Flan-T5-Base achieved an average score of 2.54 out of 5. On the other hand, the Large language model deepseek-coder-1.3b-instruct has achieved a score of 3.32 out of 5. As shown in table I

TABLE I
EVALUATION SCORES

Model	Average Score (out of 5)
Flan-T5-Base	2.54
deepseek-coder-1.3b-instruct	3.32

V. DISCUSSION

It is clear that the LLM achieved a higher and better score than the small transformer and that is due to many reasons such as the number of training data is larger in the LLM than in the transformer. Also, the type of trained data on the LLM is majorly about programming and codes, in addition to the complexity of the LLM is approximately 6 times more complex than the transformer. Thus, all the reasons mentioned are important factors for justifying and analyzing the results of both models and experiments.

The results of Flan-T5-Base and Deepseek-coder-1.3b-instruct models revealed a clear difference in their performance when answering programming questions using Retrieval-Augmented Generation (RAG). Flan-T5-Base achieved an average score of 2.54 out of 5, demonstrating good performance in handling straightforward questions but struggling with more complex queries due to its smaller model size and capacity. However, Flan-T5 is notably faster in inference, making it a good option for many scenarios where response speed is more important than the quality of the answer. In contrast, Deepseek-coder-1.3b-instruct scored an average of 3.32, demonstrating its ability to understand and generate accurate responses to the queries. This model's larger complexity architecture allows it to capture programming questions better than Flan-T5. Finally, the results suggest that while Flan-T5-Base excels in speed and efficiency, Deepseek-coder-1.3b-instruct is better at providing in-depth solutions for complex programming challenges.

VI. CONCLUSION & FUTURE WORK

In conclusion, this study has so far attempted to address the difficult development of effective QAS for the programming domain, which is very dynamic and within which much heterogeneous kind of user queries is put forward. Traditional QAS systems generally face complexity in giving short and contextually relevant replies because of the static nature of knowledge bases and lack of adaptability to the intricate details of programming languages. In this paper, we attempt to go a step further and integrate large-language models and retrieval-augmented generation systems to build up a dynamic system that is able to evolve and adapt to the nuanced demands of programming-related inquiries. Our comparative analysis shows that the Transformer models and LLMs boost the output for accuracy and relevance, but LLMs will have better understanding capabilities and adaptability resulting from dual-process extensive pre-training and domain-specific fine-tuning. Although these successes are present, we also discovered a number of remaining problems, including the need for a fine balance between effectively integrating the retrieval and maintaining high model accuracy. Our work not only highlighted these challenges but also paved the way for further progress to indicate the way forward for more research on advanced mechanisms for retrieval in varied programming contexts. This continual advancement stands to have a profound effect on the nature of how programming documentation is interacted with, with far-reaching influence on educational techniques and professional practice in software engineering.

While the results achieved by both experiments can be improved, in future work, we are planning to improve the rag system in which a semantic chunking system will be implemented to tailor the chunking to our documents and domain. Also, we will generate multiple questions for each chunk in which the similarity will be more precise since it

will compare a question with a question. In addition to fine-tuning the traditional transformer on a programming dataset. Lastly, the search tool has room for improvement and will be improved.

VII. References

[1] 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, Volume 1 (Long and Short Papers)* (J. Burstein, C. Doran, and T. Solorio, eds.), (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics, June 2019.

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[4] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[5] T. Zhang, S. G. Patil, N. Jain, S. Shen, M. Zaharia, I. Stoica, and J. E. Gonzalez, "Raft: Adapting language model to domain specific rag," *arXiv preprint arXiv:2403.10131*, 2024.

[6] M. Ramos, M. J. V. Pereira, and P. R. Henriques, "A qa system for learning python," in *Conference on Computer Science and Information Systems*, 2017.

[7] Y. Tian, F. Thung, A. Sharma, and D. Lo, "Apibot: Question answering bot for api documentation," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 153–158, 2017.

[8] A. Bansal, Z. Eberhart, L. Wu, and C. McMillan, "A neural question answering system for basic questions about subroutines," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 60–71, 2021.

[9] O. Chinedu and A. Ade-Ibijola, "Python-bot: A chatbot for teaching python programming," *Engineering Letters*, vol. 29, pp. 25–34, 02 2021.

[10] T. Yu, X. Gu, and B. Shen, "Code question answering via task-adaptive sequence-to-sequence pre-training," in *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 229–238, 2022.

[11] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: Large language model connected with massive apis," *arXiv preprint arXiv:2305.15334*, 2023.

[12] N. O, "Using retrieval-augmented generation to elevate low-code developer skills," *Artificial Intelligence*, 2023.

[13] Y. Hicke, A. Agarwal, Q. Ma, and P. Denny, "Ai-ta: Towards an intelligent question-answer teaching assistant using open-source llms," 2023.

[14] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, and F. Wei, "Multilingual e5 text embeddings: A technical report," *arXiv preprint arXiv:2402.05672*, 2024.

[15] Chroma, "Chroma open-source vector databse." Database accessed from Chroma, https://www.trychroma.com/.

[16] prompting guide research, "Retrieval augmented generation (rag) for llms," 2024. Article accessed from promptingguide.ai, https://www.promptingguide.ai/research/rag.

[17] Google, "flan-t5-base," 2023. Model accessed from HuggingFace, https://huggingface.co/google/flan-t5-base.

[18] deepseek ai, "deepseek-coder-1.3b-instruct," 2023. Model accessed from HuggingFace, https://huggingface.co/deepseek-ai/deepseek-coder-1.3b-instruct.

[19] MetaAI, "llama3-70b-8192," 2024. Model accessed from MetaAI, https://llama.meta.com/llama3/.

[20] neulab, "Conala," 2022. data retrieved from HuggingFace, https://huggingface.co/datasets/neulab/conala.