UDC: 004.89 https://doi.org/10.15407/jai2023.03.126

O. Nakhod
National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute", Ukraine
Peremohy ave., 37, Kyiv, 03056
alexey.nakhod@gmail.com
https://orcid.org/0009-0000-4350-6376

USING RETRIEVAL-AUGMENTED GENERATION TO ELEVATE LOW-CODE DEVELOPER SKILLS

Abstract. This article proposes applying retrieval-augmented generation (RAG) to improve the skills of low-code developers by augmenting large language models with up-to-date domain-specific knowledge. As low-code development requires combining multiple systems into a final product, developers must consult several sources of documentation and various articles, videos, and forum threads. Such a process may be time-consuming, prompting the use of an LLM for the authoritative answer. However, LLMs often lack knowledge of low-code platforms, leading to hallucinations and superficial responses. RAG utilizes the benefits of LLMs on relevant information, suggesting a presumption that it may be effectively applied in low-code development. Heterogeneous data sources concerning low-code systems are converted to a text representation, split into logical chunks, and stored in a vector database. During the exploitation of the model, cosine similarity is used to retrieve top-K documents and concatenate them with user query, using the produced text as a prompt to an LLM. The results support the hypothesis that RAG models outperform standard LLMs in knowledge retrieval in this domain.

Keywords: retrieval-augmented generation, low-code development, large language model, vector database, knowledge retrieval

Introduction

Low-code development refers to building software using specialized tools and solutions that streamline the development process. Such ease is facilitated by pre-built components that combine everyday programming tasks into distinct entities. These entities can then be extensively tested to ensure the high quality of developed software across all companies that utilize the low-code paradigm.

However, the providers of low-code tools often leave the customers with little guidance on using those tools. Admittedly, most low-code platforms host up-to-date documentation that covers the potential questions and describes the provided tools in great detail. Nevertheless, adopters of low-code solutions frequently encounter stagnation while solving their problems due to the inevitability of previously uncovered questions arising. There is also a possibility that the provided knowledge platform is hard to navigate or query for the required information. Finally, developers that utilize low-code platforms often try to integrate multiple solutions and thus must consult numerous sources simultaneously.

Perhaps the first resort the developers turn to when confronted with a problem is a search engine like Google. Unfortunately, even though search engines try to provide the most relevant information, they can overwhelm the user with the number of links and potential solutions. Additionally, many developing low-code platforms lack extensive forum threads, articles, and discussions concerning every possible issue that can arise while creating a product.

The next possible solution for a problem is a large language model (LLM) like ChatGPT that uses transformer [1] architecture. It utilizes the knowledge from the data it has been trained on to give a single answer to a prompt; thus, there is no longer the oversaturation of proposed solutions. Nevertheless, such models also have their limitations. First, they are trained on the data only up to a certain point in time, which causes the models to ignore current events. Moreover, they frequently "hallucinate" — make up the response without backing evidence. Finally, they provide no sources for their claims even

in the event of a correct answer, leaving the user in doubt of the legitimacy of the LLM's statements.

A potential solution can be found by combining the advantages of natural language processing from LLMs, vast domain knowledge of structured resources like product documentation, and practical search algorithms. One such approach is retrieval-augmented generation (RAG) [2], which aims to augment the processing done by a language model with external knowledge from a document database.

Related Work

The concept of RAG [2] has been introduced in the context of open-domain question answering, abstractive question answering, Jeopardy question generation, and fact verification. It has been shown that a unified architecture can achieve state-of-the-art performance across many tasks.

RAG has been applied in the software development process to gain a deeper understanding of the produced code [3]. Such insights were achieved by combining the benefits of Graph Neural Networks (GNNs) with a novel retrieval-augmented mechanism.

The applications of RAG in coding were extended to code generation via REDCODER [4] – a retrieval-augmented framework that uses state-of-the-art dense retrieval to provide context to a generative model.

The efficiency of the RAG pipeline has been considerably improved using Hierarchical Selection and Dense Knowledge Retrieval [5], reducing the computation time by a factor of 100 in a task-oriented dialog system.

Several methods have been proposed to adapt RAG to heterogeneous knowledge [6]. One such approach is homogenizing knowledge from different domains to unified unstructured text. We can also utilize graph databases to facilitate multi-hop reasoning over heterogeneous structured sources.

Another promising modification is a forward-looking active retrieval-augmented generation (FLARE) [7], which retrieves the documents based not only on the user's input but also on the model's prediction of the following statement and regenerates the sentence in the event of low-confidence tokens.

It has also been demonstrated that it is not necessary to modify the model architecture to incorporate external knowledge for the RAG process [8]. A simple concatenation of the retrieved documents to the input (in-context model) produces satisfactory results in language modeling and open-domain question answering.

The following metrics have been suggested to evaluate the performance of different LLMs in RAG: noise robustness, negative rejection, information integration, and counterfactual robustness [9]. It was shown that the introduction of RAG has posed several challenges, even for state-of-the-art LLMs.

The advantages of using an external knowledge source in RAG models and storing knowledge in parametric models were combined into a key-value memory [10] to improve the accuracy and execution time of question answering.

The widespread interest in RAG has led to dedicated discussions on recent developments in this area [11]. They attract an audience interested in natural language generation and information retrieval, covering the topics of dialogue response generation, machine translation, text style transfer, etc.

RAG has been utilized in multimodal models, resulting in a Multimodal Retrieval-Augmented Transformer (MuRAG), which employs external non-parametric memory to improve language generation [12]. MuRAG has achieved state-of-the-art accuracy on WebQA and Multimodal QA.

Other sources consider using RAG in multimodal models as a way to integrate knowledge in a more scalable and modular way [13]. It has been shown that the resulting model outperforms DALL-E and CM3 on image and caption generation tasks while requiring much less compute resources for training.

Particular attention has been given to constructing a valuable representation of the documents in the database. One such structure is a knowledge graph, which can be built using the accomplishments in a related task – slot filling [14]. An approach called KGI has been

shown to improve dense passage retrieval in the KILT benchmark.

Suggested Method

Let us examine the RAG model in greater detail. The process of RAG [2] is split into multiple steps. First, a query is received, and the model retrieves the best matches from the database using dense passage retrieval (DPR) [15]. Next, each match is used as context to an LLM [16], which produces a distribution for every output token. Finally, the distributions are marginalized based on the contribution of each document. This process is illustrated in Fig. 1.

RAG can provide low-code developers with relevant, up-to-date, evidence-driven responses. Consider the following example that demonstrates the advantages of RAG.

A developer might need to combine two low-code platforms, Caspio (a database solution) and Power Automate (an automation solution). A potential question may be: "How do you get Caspio tables in Power Automate?" Such an experiment at the time of writing using *gpt-3.5-turbo* produces a response with numerous flaws.

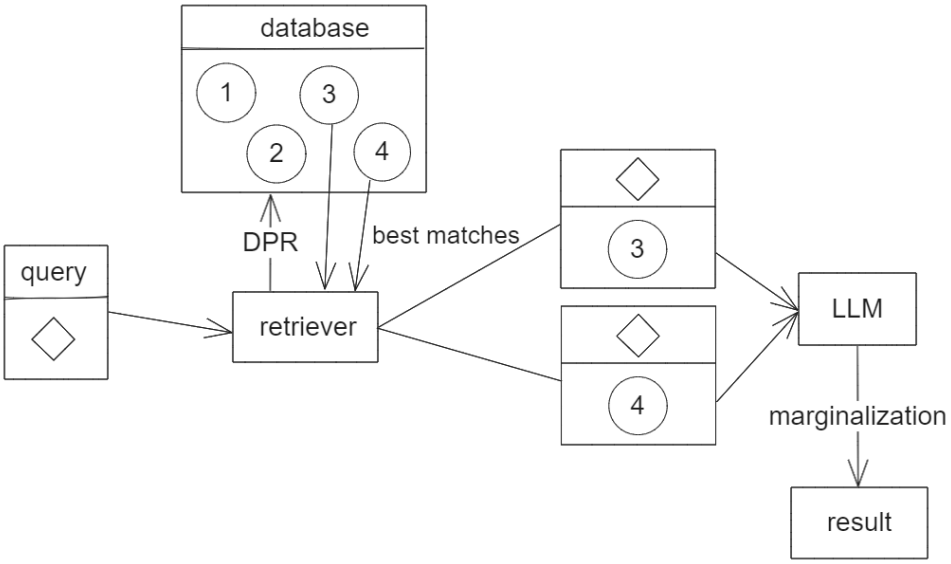

Fig. 1. Illustration of RAG algorithm

For instance, the model provides an outdated Caspio API endpoint ("The base URL usually follows the format: `https://caspio.com/rest/v1/tables/{table_name}`"), does not elaborate on specific steps to get a Caspio API key ("You may need to include an API key or credentials in the headers to authenticate."), and fails to mention detailed actions to create a corresponding flow in Power Automate ("Open the Power Automate platform and create a new flow").

To eliminate the shortcomings of the standard model, the author proposes to construct a RAG model in the following way.

First, a retrieval structure is chosen. Word embeddings [17] provide the advantage of quick queries to find phrases with similar meanings, so that is the approach taken in this article. OpenAI Embeddings API provides *text-embedding-ada-002* for creating the embeddings, while Pinecone API stores them in a database with subsequent querying.

Second, a generation model is established. OpenAI Chat Completion API gives access to state-of-the-art *gpt-3.5-turbo*, justifying its selection for this task.

Third, official Caspio and Power Automate documentation data is downloaded and split into chunks, each representing a logical piece of information. An embedding algorithm processes the data segments, and the resulting embeddings are inserted into a vector database. One segment is depicted in Fig. 2. *Score* represents the cosine similarity between the queried and the stored vector on a scale from -1 if vectors are pointing in opposite directions to 1 when vectors are pointing in the same direction. The vector is in an n-dimensional space and is represented by

values. *Text* is a key in the *metadata* dictionary.

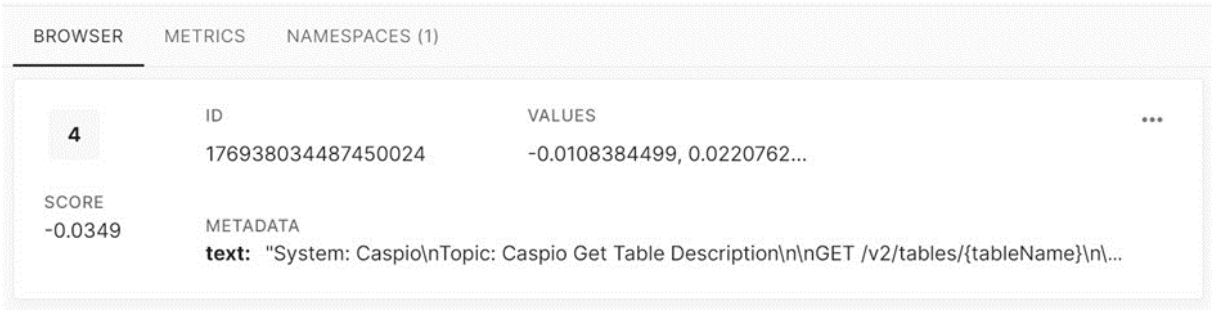

Fig. 2. An example of a typical vector stored in Pinecone

When a *query* is received, its embedding is computed, and the cosine similarity [17] between the user's question vector and each database vector is calculated using the following formula:

$$cosine\ similarity = \cos\theta = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

An example of two such vectors is presented in Fig. 3.

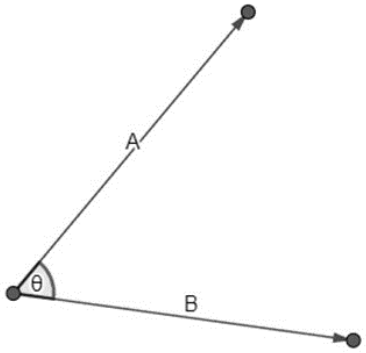

Fig. 3. An illustration of two vectors, denoted as A and B, with an angle θ included between them

The top five scoring vectors are returned as containing the most relevant text. These vectors form the *context* for the subsequent input to an LLM in the form of "Use all the information: {*context*} and answer this user's query in detail: {*query*}." The LLM generates the corresponding response.

Results

The resulting output of the model is up-to-date, relevant, and detailed, providing sufficient information to answer the user's query. In the example considered, RAG model:

- elaborates on the necessary actions to get a Caspio API token ("Here are the steps: … This token is generated using the Client ID/Secret pair that can be found on your Caspio REST API profile page… Body:grant_type=client_credentials&client_id=<Your_Client_ID>&client_secret=<Your_Client_Secret>…");
- provides the correct endpoint ("/v2/tables/{tableName}");
- gives advice on handling the response in Power Automate ("Parse the response: The response will be in JSON format…").

Conclusion

In summary, using RAG has demonstrated considerable improvement in

output compared to a standard LLM. Such a solution will allow low-code developers to solve the arising issues during product development and free their time for further iteration. Although this work was conducted with low-code developers in mind, the results can apply to people without a technical background or professionals who use proprietary knowledge systems and require effortless search and summarization capabilities. The model may be enhanced by using a different way of storing and retrieving information, perhaps in a graph database [18], on which further research efforts will be focused.

References

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.
2. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, *33*, 9459-9474.
3. Liu, S., Chen, Y., Xie, X., Siow, J., & Liu, Y. (2020). Retrieval-augmented generation for code summarization via hybrid gnn. *arXiv preprint arXiv:2006.05405*.
4. Parvez, M. R., Ahmad, W. U., Chakraborty, S., Ray, B., & Chang, K. W. (2021). Retrieval augmented code generation and summarization. *arXiv preprint arXiv:2108.11601*.
5. Thulke, D., Daheim, N., Dugast, C., & Ney, H. (2021). Efficient retrieval augmented generation from unstructured knowledge for task-oriented dialog. *arXiv preprint arXiv:2102.04643*.
6. Yu, W. (2022, July). Retrieval-augmented generation across heterogeneous knowledge. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Student Research Workshop* (pp.52-58).
7. Jiang, Z., Xu, F. F., Gao, L., Sun, Z., Liu, Q., Dwivedi-Yu, J., ... & Neubig, G. (2023). Active retrieval augmented generation. *arXiv preprint arXiv:2305.06983*.
8. Ram, O., Levine, Y., Dalmedigos, I., Muhlgay, D., Shashua, A., Leyton-Brown, K., & Shoham, Y. (2023). In-context retrieval-augmented language models. *arXiv preprint arXiv:2302.00083*.
9. Chen, J., Lin, H., Han, X., & Sun, L. (2023). Benchmarking Large Language Models in Retrieval-Augmented Generation. *arXiv preprint arXiv:2309.01431*.
10. Wu, Y., Zhao, Y., Hu, B., Minervini, P., Stenetorp, P., & Riedel, S. (2022). An efficient memory-augmented transformer for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2210.16773*.
11. Cai, D., Wang, Y., Liu, L., & Shi, S. (2022, July). Recent advances in retrieval-augmented text generation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 3417-3419).
12. Chen, W., Hu, H., Chen, X., Verga, P., & Cohen, W. W. (2022). Murag: Multimodal retrieval-augmented generator for open question answering over images and text. *arXiv preprint arXiv:2210.02928*.
13. Yasunaga, M., Aghajanyan, A., Shi, W., James, R., Leskovec, J., Liang, P., ... & Yih, W. T. (2023). Retrieval-augmented multimodal language modeling.
14. Glass, M., Rossiello, G., Chowdhury, M. F. M., & Gliozzo, A. (2021). Robust retrieval augmented generation for zero-shot slot filling. *arXiv preprint arXiv:2108.13934*.
15. Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., ... & Yih, W. T. (2020). Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.
16. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877-1901.
17. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.
18. Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G. D., Gutierrez, C., ... & Zimmermann, A. (2021). Knowledge graphs. *ACM Computing Surveys (Csur)*, *54*(4), 1-37.

The article has been sent to the editors 12.10.23.
After processing 22.10.23.
Submitted for printing 30.11.23.