Environment. Technology. Resources. Rezekne, Latvia
Proceedings of the 16th International Scientific and Practical Conference. 2025. Volume II, 345-351

Application of a Large Language Model for Air Pollution Analysis by Using a RAG System

Svetlomir Stankov
Faculty of Mathematics and Informatics
Sofia University "St. Kliment Ohridski"
Sofia, Bulgaria
sv_stankov@abv.bg

Desislava Velinova
Development of C4I Systems
Bulgarian Defence Institute "Prof. Tsvetan Lazarov"
Sofia, Bulgaria
velinova.desislava@gmail.com

***Abstract*—The paper examines the application of an LLM (Large Language Model) for analysing text information about air pollution using a RAG (Retrieval-Augmented Generation) system. It describes what RAG systems are and how they help increase the credibility of the text generated by LLM models (reducing their hallucinations) using information retrieval from reliable sources. An approach for the development of a RAG system suitable for answering queries on imported text documents by the user is presented. The realisation of this approach is the development of a RAG chatbot using the Python language, answering queries (questions) towards a database of uploaded text documents by the user. Some results from experiments on a selected database of scientific publications related to air pollution are presented.**

***Keywords*—Air pollution, Artificial Intelligence, LLM, RAG System**

I. Introduction

The area of Artificial Intelligence (AI) was developed enormously in recent years, and one of its most significant novelties was the introduction of Large Language Models (LLMs), which revolutionised text generation models with the ability to process and generate natural human-like texts. These models were made accessible to the general public both on the web and locally, on personal computers, and they gained massive popularity. They are being used for a variety of applications, such as chatbots and assistants.

Large Language Models (LLMs) are advanced machine learning algorithms that are designed to understand and generate human-like text (Fig. 1).

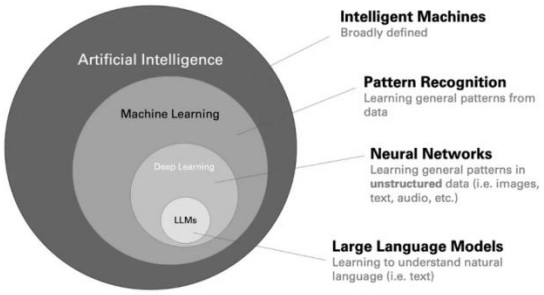

Fig. 1. Essence of Artificial Intelligence in layers [1].

These models are trained on massive datasets of text, which enables them to learn patterns, grammar, and vocabulary. As a result, LLMs can perform a wide range of tasks, including language translation, text summarization, question answering, and even creative writing. Some of the most well-known LLMs include GPT, BERT, Gemini, T5, DeepSeek, which have been developed by organizations such as OpenAI, Google, Microsoft, Hangzhou AI.

Their ability to understand and generate human-like text has made them valuable tools in fields such as customer service, content creation, and language translation. For instance, LLMs are used to power chatbots that can engage in natural conversations with customers, provide information, and even assist with problem-solving.

One of the distinctive and key strengths of LLMs is their ability to respond to unpredictable queries. Unlike other computer programs and models, which perform specific commands from a finite set of them or anticipate specific user inputs, LLMs can operate with natural human language and respond to unstructured commands, which revolutionized the artificial intelligence field [2].

Online ISSN 2256-070X
https://doi.org/10.17770/etr2025vol2.8603

A. The problem with LLMs – hallucinations

While LLMs are great at generating good-looking, human-like text, they can sometimes suffer from a phenomenon known as "hallucinations". Hallucinations occur when an LLM generates incorrect or nonsensical information that is not based on any factual evidence. This undermines LLMs' reliability and causes doubts about the accuracy of the information they provide. Mitigating these hallucinations is therefore a critical issue to be addressed both by the developers of the models and the creators of programs that use these LLMs.

There are different types of hallucinations, which include factual inaccuracies, nonsensical responses and contradictions. The first type (also known as fact-conflicting hallucinations) represents LLMs' tendency to sometimes generate incorrect or misleading information that is not based on any real facts. For instance, a user may ask for information about a non-existent person and the LLM may generate a story about him that, although well written, does not correspond to reality. The nonsensical responses (input-conflicting hallucinations) refer to the fact that LLMs at times respond with irrelevant information to a query, mainly due to not understanding the context correctly. There are also cases when LLMs may generate contradicting statements in a prolonged conversation or even in the same response in some instances (context-conflicting hallucinations). Studies have found a notable presence of self-contradictory responses in LLMs. For example, the contradiction rate of ChatGPT has been estimated to be 14.3% [4], [5], [6].

Hallucinations can happen for several reasons, including the model's training data containing inaccuracies and false information, the model's inability to fully understand the context of a given task, or the model's tendency to overgeneralize or make assumptions. The massive datasets used by companies that develop LLMs to train the models contribute to the problem with hallucinations because they consist of a variety of different resources on many different topics, and the verification of the factuality and unbiasedness of the information is challenging.

All in all, it could be said that LLMs are designed and trained in a way to identify patterns in texts and predict well the next word in a sequence, and the emphasis is on generating good-looking and plausible-sounding text. This makes the LLMs so good at generating human-like text but hinders the factuality and truthfulness of the information they provide, especially in cases where they lack sufficient information about a topic. More information can be found in [6].

B. Retrieval-Augmented Generation (RAG)

One of the most comprehensive solutions to improve the problem with hallucinations in LLMs for specific use cases and applications is the Retrieval-Augmented Generation (RAG) framework. There has been a lot of research on this methodology in recent years, and it has proved to be very effective at mitigating hallucinations in generative models by combining information retrieval techniques with the creativity of LLMs.

The main idea of RAG is to use pre-trained models to retrieve relevant documents from large databases and then use a generative model (LLM) to create answers based on the extracted information.

This approach is particularly useful in cases where the generative model does not have access to specific data, but can obtain it by retrieving it from suitable sources. RAG allows models to use information from various sources, thereby improving the accuracy and quality of the generated answers.

RAG consists of several key stages:

- Preprocessing and indexing: At the beginning, the documents in the database are preprocessed, and longer documents are broken into several smaller chunks so that it helps the retrieval process but also so that the relevant chunks fit the context size of the chosen LLM. The document chunks are then embedded by a chosen embedding model (it turns the chunk into a vector of high dimensions and is trained to capture the semantic meaning of a text), which makes them eligible for semantic search (they are compared to the embedding of the user query).
- Retrieval: A pre-trained embedding model (may be combined with a keyword-based algorithm) is used to retrieve relevant documents or passages from a database that are related to the user's question. In order to achieve this, the user query must be processed and embedded and then added to the search models to find the most relevant chunks of text (which are also preprocessed and embedded in the initial stage). This is a crucial stage that has a great impact on the performance of a RAG system.
- Augmentation: The top-ranked chunks (passages of text) retrieved by the retrieval stage are added to the prompt. The prompt at the end consists of instructions to the LLM (to read and gather information from the context when answering the query), the relevant context passages, and the user query for which an answer is expected.
- Generation: A generative model (LLM) is used to create an answer to the user's question on the basis of the context provided [7], [8], [10].

RAG is particularly effective in the context of tasks such as question answering, document summarization, and content creation based on information from multiple sources. This approach allows models to be more informative and accurate while still retaining their ability to generate creative and contextually relevant answers.

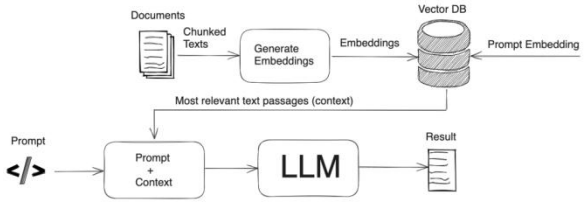

Fig. 2.Basic RAG pipeline [9]

II. MATERIAL AND METHODS

A. Approach for the realisation of RAG in a system which answers user queries in English and Bulgarian

As RAG systems are still in the beginning of their development and such systems with user interfaces for general use are not known to exist for Bulgarian language, an approach for developing such is proposed. Existing such systems (like the NVIDIA RAG Chatbot) mainly use LLMs that are intended to work in English language and do not perform efficiently enough for the Bulgarian language, as the information is translated between the languages, and this hinders the accuracy of the answers.

For this reason, the proposed approach is based on the utilization of the recently developed and introduced LLM for Bulgarian language, BgGPT-Gemma-2, developed by INSAIT [11], [12], which was built on top of Google's Gemma 2 family of models with various additional improvements, including continuous pre-training on around 100 billion tokens in Bulgarian and a novel instruction-fine tuning.

The essence of the proposed approach here is to provide the user with the ability to upload his own documents and use RAG on them so that valuable knowledge and insights in the documents texts are gained by asking queries to a chatbot while the hallucinations in the LLM are reduced.

The proposed approach utilizes several techniques at different parts of the system workflow.

The first part of the RAG process is the preprocessing of each document and splitting the longer documents into chunks. For this part of the process, it is suggested to use sentence-based chunking for general text documents and use cases, as it captures the context of the full sentences and not just a part of them because it represents a natural and logical division of the text. It is proposed the chunking size to be 20 sentences with an overlap of 2 sentences between the chunks to maintain the connection between sequential chunks. Smaller documents (less than the chosen chunking size) should not be split. The formed chunks are then embedded into high-dimensional vectors used for performing semantic search in the retrieval process. It is proposed to use the embedding model BGE-M3 by BAAI, as it is multilingual and is able to process inputs of different granularities – from short sentences to long documents up to 8192 tokens [17]. That functionality is essential for RAG systems, as user queries are generally short but have to be compared to large documents to find the most relevant ones. The embeddings are proposed to be stored in a vector database like FAISS to improve the semantic search speed, which is essential for systems with large document databases. In order to improve the retrieval of relevant chunks, the performing of a keyword search along with the semantic (vector) search is proposed. For the keyword search, the BM25 algorithm is proposed, which is implemented in the bm25s package in Python as well as in other packages like rank-bm25. For the keyword search to be more efficient, a lemmatisation of the chunks is needed. The package simplemma in Python can be used, as it also offers lemmatisation in multiple languages and also has a language detector useful for RAG systems that are intended to work with multiple languages.

In order to combine the results from the keyword and the semantic (vector) search, a ranking algorithm is needed. The reciprocal rank fusion (RRF) algorithm is proposed to be used, as it is a simple and fast algorithm that shows remarkable results, better than even its more complex competitors [18].

RAG Fusion uses the LLM to generate similar queries to the original one by making small modifications so that a wider view of the topic is obtained for improving the retrieval of relevant chunks to the user query. These similar queries, along with the original one, are then given to the retrieval process to find relevant contexts for each of the generated queries (plus the original user query). The results are combined by the RRF algorithm.

The HyDE retrieval method is a zero-shot learning technique; the LLM is used to provide an answer without relevant context, as when RAG is not used. This answer (a hypothetical document) is then embedded by the embedding model, and vector search as well as keyword search is executed, finding relevant chunks to the generated answer. Even if some details provided in the generated answer are inaccurate, the method tends to capture relevant patterns which improve the retrieval process [15]. Moreover, the generated answer has a lot more words and sentences than a single-line user query, for example, and can capture deeper semantic

relationships between the words, which helps the search process, as the chunks also contain a lot of information and are longer in length.

The generated subqueries method uses the LLM to detect whether the query is a complex query and if the user has turned on the generated sub-queries option, another function breaks the user query into several self-sufficient simple queries. These are used to improve the retrieval process, as sometimes the query may ask for information about different topics which may hinder the retrieval process. Moreover, it improves the generation of the answers for complex queries, as the sub-queries are numbered and given to the LLM, which ensures that every distinct question in the query is answered and is not ignored. The retrieved relevant text chunks for each subquery are combined with the RRF algorithm to get the final top relevant chunks.

The most important aspect of the RAG systems is the retrieval process. It needs to find relevant context to a query from the database of text documents. For this purpose, along with the keyword search and the semantic search, the use of several different advanced techniques that utilize the LLM is proposed – RAG Fusion, HyDE approach, generated sub-queries, LLM filtering of the retrieved contexts and modification of the original user query based on the chat conversation if the query is not classified as a standalone one (it is connected with a previous interaction with the chatbot).

In order to improve the retrieval process for specific documents and use cases, one or more of the abovementioned methods may be used. It is suggested to use both HyDE and RAG Fusion methods, as they generally bring the best retrieval results when used together. When used together, the results from these methods are ranked by the RRF algorithm to obtain the final relevant context chunks, which are then given to the LLM (or filtered by the LLM before that if this option is chosen). Furthermore, a reranking model could be used to rerank the top 20 context chunks by the RRF algorithm ranking, based on their relevancy to the user query. The top reranked context chunks are then provided to the LLM (or additionally filtered by the LLM before that if this option is chosen).

For the last part of the RAG pipeline, the generation phase where an LLM is used to generate the answer, the most important point is the prompt engineering or the instructions given to the LLM in the prompt. Here it is suggested to set a clear instruction to the LLM that it needs to use factual information from the provided context by the retrieval process to answer the query and to not be reluctant to say that it cannot answer the question if it does not have enough information on the matter.

III. Results and discussion

A. Application of the developed RAG system in ecology - for extracting information from air pollution documents.

Based on the approach proposed above, a RAG system was developed, using the Python programming language, with the application of the BgGPT-Gemma-2 LLM. The quantized GGUF models are used with llama.cpp for Python so that the LLM can be loaded on the GPU (using the CUDA toolkit of NVIDIA) of normal personal computers with limited available VRAM. The system was tested on the version "BgGPT-Gemma-2-9B-IT-v1.0.Q6_K.gguf" but has the ability to download other quantized versions of the LLM based on the VRAM available on the user's personal computer. The generative model can work with both English and Bulgarian languages.

The developed RAG system works for the Bulgarian and English languages and also has a user interface. The system gives users the opportunity to get more reliable information from the LLM and the ability to see the resources used by the LLM for answering their query.

An important feature of the developed system is that it gives users the ability to import their own text documents and files in various formats (PDF, TXT, DOC, DOCX and JSON) and ask queries about them. The system is designed to be used locally, on a personal computer, so the files will not be uploaded on any servers. The feature provides the user with the opportunity to take advantage of the LLM's strengths in text generation on his own text files in a more reliable way.

In the case when the user has not imported any files, the system has a default database of thousands of Bulgarian articles from Bulgarian Wikipedia [13] and focus news website [14]. They are used in order to help mitigate the LLM hallucinations and give relevant resources accompanying the answer to the query. It is an optional database that can be used or not based on the user preference.

B. Testing of the developed RAG system for air pollution documents, experimentations and results

The developed RAG system can be used in various areas, as it is made to work with any text documents. The system was tested for extracting information from around 40 research papers on the topic of air pollution in English. They were imported into the system as a database for the RAG. The documents are PDF files containing various research papers in the field of air pollution. The files were processed normally following the RAG pipeline described in this paper. In order to assess the presented system, around 70 queries were asked on topics that are mainly discussed in the chosen articles.

In order to evaluate the system, the REMi evaluation model was chosen, classified by the developers as the first open-source RAG evaluation model [3]. The model is based on the RAG Triad evaluation approach, which is based on three metrics to evaluate RAG systems:

- answer relevance: assesses how relevant the generated answer is to the query
- context relevance: assesses the relevance of the retrieved context to the query
- groundedness: assesses the degree to which the generated answer is grounded in the provided retrieved context (how much the answer supports the information in the context)

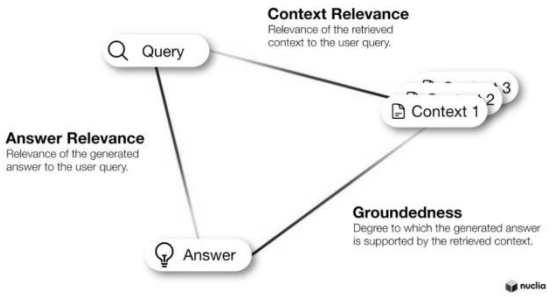

Fig. 3. Diagram illustrating the RAG Triad evaluation metrics [16].

The model REMi used to evaluate the RAG system is a relatively small (around 15 GB) and efficient LLM adapted on top of Mistral AI's Mistral 7B v0.3 Instruct. It was fine-tuned specifically for evaluation with the RAG Triad metrics. The REMi model gives scores from 0 to 5 on each of the metrics while also providing an additional reasoning for the answer relevance score. It is provided by the nuclia-eval package in Python. All the model needs to do the evaluation is the query, the retrieved relevant contexts and the generated answer from the system. The model provides an opportunity to assess how the RAG system works without having ground truths and golden datasets and makes it easier to detect early potential problems.

The model gives one score for the answer relevance, while for the context relevance and the groundedness, it gives scores for each retrieved context for the query in the form of a list. For the metrics to be observed with a single value each, an aggregation is needed along the context pieces for a single query and then across the multiple queries. The proposed aggregation by the developers of the evaluation model is to get the maximum score for each of the contexts retrieved for the query and then average along all queries [3]. This method is also used to evaluate the implemented RAG system.

As was already mentioned, around 70 queries were asked to the RAG system in order to evaluate its performance on the dataset of around 40 air pollution research papers. In this section we will take a look at some of these experiments and the obtained results.

C. Analysis of the results

As a whole, the results obtained from the evaluation model are pretty positive – from around 70 queries asked, one answer was given a 0 score for answer relevance to the query, 2 answers were given a 3 score, 23 were given a 4 score and the rest 43 answers were given the maximum relevancy score. The average score for answer relevance across all queries is, therefore, 4.5 or 90%.

For the context relevance metric, the average score across the queries for the top 3 chunks retrieved is 3.95 or 79%, which is very high considering this result is obtained for all 3 top chunks used for these experiments. The aggregation proposed by the creators of the evaluation model REMi [3] uses just the maximum relevance score of the retrieved contexts for a query, and then an average across all queries is performed. Using this approach, the average context relevance score across the queries is 4.81 or 96%, which shows that for almost every query asked in the experimentation process, a chunk with the highest context relevance score is retrieved. This is a very positive assessment of the retrieval process in the RAG system.

Speaking of which, the most effective approach is found to be using both HyDE and RAG Fusion methods, which may hinder slightly the speed of the answer generation but improves the retrieval process and the overall results. When both HyDE and RAG Fusion are used, the average results using the maximum score for a query regarding the context relevance and groundedness scores are the following: answer relevance of 4.6 (92%), context relevance of 4.97 (99%) and groundedness of 4.4 (88%). It can be seen that combining both techniques improve the scores and is better than the general results at all 3 metrics. If we calculate the average score across all 3 top retrieved chunks for a query and just the one with the best score, the context relevance average is 4.26 (85%) and the groundedness is 2.4 (48%). This means that in most cases all 3 of the retrieved chunks have very high relevance scores to the query, confirming the theory that using both HyDE and RAG Fusion improves the overall RAG process.

Otherwise, the groundedness general average score is 3.89 (78%). If we take a look across all scores for the top 3 chunks retrieved, the score is 2.08 (42%), which is not a problem because it is across all top 3 chunks retrieved for a query, and it is hard and impractical in many cases for the LLM to use all 3 retrieved chunks for the generation of the answer (provided all 3 are actually relevant to the query and can be used for the answer). This shows that the system tends to use mainly 1 and sometimes 2 of the retrieved chunks to answer a query. These results also show that a better retrieval process (when using both HyDE and RAG Fusion) also improves the groundedness of the answers in the contexts.

These results are a promising assessment of the developed RAG system based on the proposed approach in the paper, which shows that the pipeline implemented in the system can be very effective in generating factual answers when a dataset of quality documents is provided. These results, however, are based very much on the queries asked, as they have to be relevant to the documents provided; otherwise, the scores on the metrics will drop. Therefore, providing relevant documents to the topic of interest is essential for the user to improve his experience with such RAG systems. The dataset used for these

experimentations was with documents in English, but the system can perform equally well also with Bulgarian documents, which was the initial and primary objective of this system. This was confirmed by the experiments that were conducted with Bulgarian research papers on the topic of air pollution, as the results from them were very similar to the results obtained from the experiments with the English research papers.

Further experiments were conducted to compare the different techniques for the context retrieval process using repetition of the asked queries as well as new queries for each technique (around 45 queries were asked for each technique) in order to obtain more factual results. Bear in mind, however, that although there were a number of cases where the same query was asked for all possible combinations of techniques, there were also unique queries for some of the techniques. It was not the same set of queries asked for all combinations of the used techniques.

The obtained results from these experiments confirmed once again that using both RAG Fusion and HyDE provides the best overall results. The results from combining the two techniques showed around 2% less average answer relevance than the other techniques (around 90% for combining both RAG Fusion and HyDE, while the techniques alone show results around 92%, and the usage of none of these techniques also shows 92%); but significant improvement in the retrieval phase – the max context relevance average was 99% (while using only RAG Fusion was 97%, only HyDE was 95% and using none of the techniques was around 93%); the max groundedness scores average was around 90% (while using only RAG Fusion was 67%, only HyDE was 82% and using none of the techniques was also 82%). If we take a look at the context relevance average scores (not taking just the score for the most relevant context for a query but the average of the scores for the 3 retrieved contexts), combining both techniques showed a result of 84% (while only RAG Fusion was 78%, only HyDE was 81% and using none of the techniques was 75%). The groundedness average score for combining the two techniques was 47% (while using only RAG Fusion was 33%, only HyDE was 43% and using none of the techniques was around 42.5%).

If we compare HyDE and RAG Fusion based on the obtained results from these experimentations, then we can see that in answer relevance they provide pretty much the same results; in max context relevance average, RAG Fusion has a 2% advantage, while in context relevance average, HyDE has a 3% advantage. In the max groundedness average, HyDE provides significantly better results with around a 15% advantage over RAG Fusion, while in the groundedness average, it has an advantage of 10%. Based on the significant difference of over 10% in the groundedness scores, HyDE has an advantage over RAG Fusion, according to the conducted experiments. In the other two metrics, however, the two techniques are pretty similar.

IV. Conclusions

In the presented paper, an approach for creating a RAG system that enables the user to import his own documents to ask queries on them is proposed. A realization of the suggested approach is also presented in the form of a RAG system that uses the new BgGPT model suitable for the Bulgarian and English languages. The developed system was tested on a set of research papers in the field of ecology and, more specifically, air pollution. The results showed that the system gives factually based answers and mitigates the LLM hallucinations.

RAG systems developed on the basis of the proposed approach can be implemented in various different fields and can be used to improve the searching of information in big textual documents and serve as a useful and powerful tool to gain knowledge and information on various topics.

Acknowledgements

This work was supported by the NSP "Security and defence" program, which has received funding from the Ministry of Education and Science of the Republic of Bulgaria under the grant agreement no. Д01-74/19.05.2022.

References

[1] A.Vina, "From Code to Conversation: How Does an LLM Work?",November 2024.[Online]. Available: https://www.ultralytics.com/de/blog/from-code-to-conversation-how-does-an-llm-work [Accessed: Jan. 30, 2025].

[2] Cloudflare, "What is a large language model (LLM)?". [Online]. Available: https://www.cloudflare.com/learning/ai/what-is-large-language-model/ [Accessed: Jan. 30, 2025].

[3] C. Onielfa, "Introducing REMi, the first ever open-source RAG evaluation model", August 2024. [Online]. Available: https://nuclia.com/developers/remi-open-source-rag-evaluation-model [Accessed: Jan. 30, 2025].

[4] Nexla, "LLM Hallucination-Types, Causes, and Solution".[Online]. Available: https://nexla.com/ai-infrastructure/llm-hallucination/ [Accessed: Jan. 30, 2025].

[5] N. Muendler, J. He, S.Jenko, M. Vechev, "Self-Contradictory Hallucinations of LLMs: Evaluation, Detection and Mitigation", March 2024.[Online]. Available: https://arxiv.org/pdf/2305.15852 [Accessed: Feb. 07, 2025].

[6] A. Choudhury, "Best Strategies to minimize Hallucinations in LLMs: A Comprehensive Guide", November 2023. [Online]. Available: https://www.turing.com/resources/minimize-llm-hallucinations-strategy [Accessed: Feb. 07, 2025].

[7] H. Chawre, "What Is Retrieval-Augmented Generation (RAG) in LLMs?".[Online]. Available: https://www.turing.com/resources/understanding-retrieval-augmented-generation-rag [Accessed: Feb. 07, 2025].

[8] Wikipedia, "Retrieval-augmented generation".[Online]. Available: https://en.wikipedia.org/wiki/Retrieval-augmented_generation [Accessed: Feb. 07, 2025].

[9] K. Safjan, "Understanding Retrieval-Augmented Generation (RAG) empowering LLMs".[Online]. Available: https://safjan.com/understanding-retrieval-augmented-generation-rag-empowering-llms/ [Accessed: Feb. 07, 2025].

[10] P. Menon, "How RAG Works: A Detailed Explanation of its Components and Steps", January 2024.[Online]. Available: https://www.linkedin.com/pulse/how-rag-works-detailed-

explanation-its-components-steps-pradeep-menon-ws7sc [Accessed: Feb. 07, 2025].

[11] BgGPT-Gemma-2, [Online]. Available: https://huggingface.co/collections/INSAIT-Institute/bggpt-gemma-2-673b972fe9902749ac90f6fe [Accessed: Feb. 07, 2025].

[12] INSAIT, "INSAIT releases new AI models, setting a standart for open national language models", November 2024.[Online]. Available: https://insait.ai/insait-releases-new-ai-models-setting-a-standard-for-open-national-language-models/ [Accessed: Feb. 15, 2025].

[13] Dataset Card for Wikipedia, [Online]. Available: https://huggingface.co/datasets/legacy-datasets/wikipedia [Accessed: Feb. 07, 2025].

[14] FOCUS News, [Online]. Available: https://www.focus-news.net/[Accessed: Feb. 15, 2025].

[15] H. Sajid, "Improving Information Retrievala nd RAG with Hypothetical Document Embeddings (HyDE)", July 2024.[Online]. Available: https://zilliz.com/learn/improve-rag- and-information-retrieval-with-hyde-hypothetical-document-embeddings [Accessed: Feb. 15, 2025].

[16] Z. Rackauckas, "RAG-Fusion: A new take on Retrieval-Augmented Generation", February 2024.[Online]. Available: https://arxiv.org/pdf/2402.03367 [Accessed: Feb. 20, 2025].

[17] BGE-M3, [Online]. Available: https://huggingface.co/BAAI/bge-m3 [Accessed: Feb.15, 2025].

[18] G. Cormack, C. Clarke, S. Buettcher, Reciprocal Rank Fusion outperforms Condorcet and individual Rank Learning Methods.[Online]. Available: https://plg.uwaterloo.ca/~gvcormac/cormacksigir09-rrf.pdf [Accessed: Feb. 20, 2025].