WIKIPEDIA
The Free Encyclopedia

Retrieval-augmented generation

Retrieval-augmented generation (**RAG**) is a technique that enables large language models (LLMs) to retrieve and incorporate new information.[1] With RAG, LLMs do not respond to user queries until they refer to a specified set of documents. These documents supplement information from the LLM's pre-existing training data.[2] This allows LLMs to use domain-specific and/or updated information that is not available in the training data.[2] For example, this helps LLM-based chatbots access internal company data or generate responses based on authoritative sources.

RAG improves large language models (LLMs) by incorporating information retrieval before generating responses.[3] Unlike LLMs that rely on static training data, RAG pulls relevant text from databases, uploaded documents, or web sources.[1] According to *Ars Technica*, "RAG is a way of improving LLM performance, in essence by blending the LLM process with a web search or other document look-up process to help LLMs stick to the facts." This method helps reduce AI hallucinations,[3] which have caused chatbots to describe policies that don't exist, or recommend nonexistent legal cases to lawyers that are looking for citations to support their arguments.[4]

RAG also reduces the need to retrain LLMs with new data, saving on computational and financial costs.[1] Beyond efficiency gains, RAG also allows LLMs to include sources in their responses, so users can verify the cited sources. This provides greater transparency, as users can cross-check retrieved content to ensure accuracy and relevance.

The term RAG was first introduced in a 2020 research paper.[3]

RAG and LLM Limitations

LLMs can provide incorrect information. For example, when Google first demonstrated its LLM tool "Google Bard", the LLM provided incorrect information about the James Webb Space Telescope. This error contributed to a $100 billion decline in the company's stock value.[4] RAG is used to prevent these errors, but it does not solve all the problems. For example, LLMs can generate misinformation even when pulling from factually correct sources if they misinterpret the context. *MIT Technology Review* gives the example of an AI-generated response stating, "The United States has had one Muslim president, Barack Hussein Obama." The model retrieved this from an academic book rhetorically titled *Barack Hussein Obama: America's First Muslim President?* The LLM did not "know" or "understand" the context of the title, generating a false statement.[2]

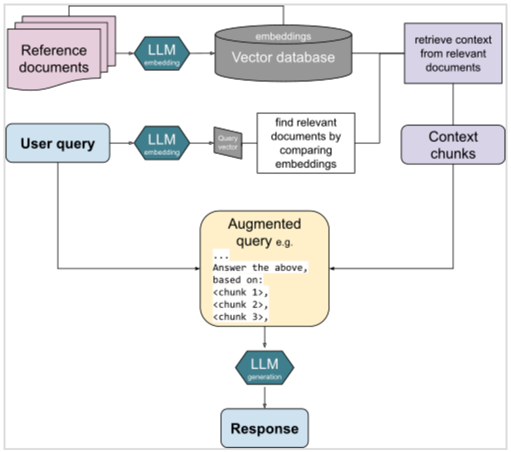
LLMs with RAG are programmed to prioritize new information. This technique has been called "prompt stuffing." Without prompt stuffing, the LLM's input is generated by a user; with prompt stuffing, additional relevant context is added to this input to guide the model's response. This approach provides the LLM with key information early in the prompt, encouraging it to prioritize the supplied data over pre-existing training knowledge.[5]

Process

Retrieval-augmented generation (RAG) enhances large language models (LLMs) by incorporating an information-retrieval mechanism that allows models to access and utilize additional data beyond their original training set. *Ars Technica* notes that "when new information becomes available, rather than having to retrain the model, all that's needed is to augment the model's external knowledge base with the updated information" ("augmentation").[4] IBM states that "in the generative phase, the LLM draws from the augmented prompt and its internal representation of its training data to synthesize an engaging answer tailored to the user in that instant".[1]

RAG key stages

Typically, the data to be referenced is converted into LLM embeddings, numerical representations in the form of a large vector space. RAG can be used on unstructured (usually text), semi-structured, or structured data (for example knowledge graphs). These embeddings are then stored in a vector database to allow for document retrieval.



Overview of RAG process, combining external documents and user input into an LLM prompt to get tailored output

Given a user query, a document retriever is first called to select the most relevant documents that will be used to augment the query.[2][3] This comparison can be done using a variety of methods, which depend in part on the type of indexing used.[1]

The model feeds this relevant retrieved information into the LLM via prompt engineering of the user's original query. Newer implementations (as of 2023) can also incorporate specific augmentation modules with abilities such as expanding queries into multiple domains and using memory and self-improvement to learn from previous retrievals.

Finally, the LLM can generate output based on both the query and the retrieved documents.[2][6] Some models incorporate extra steps to improve output, such as the re-ranking of retrieved information, context selection, and fine-tuning.

Improvements

Improvements to the basic process above can be applied at different stages in the RAG flow.

Encoder

These methods focus on the encoding of text as either dense or sparse vectors. Sparse vectors, which encode the identity of a word, are typically dictionary-length and contain mostly zeros. Dense vectors, which encode meaning, are more compact and contain fewer zeros. Various enhancements can improve the way similarities are calculated in the vector stores (databases).[7]

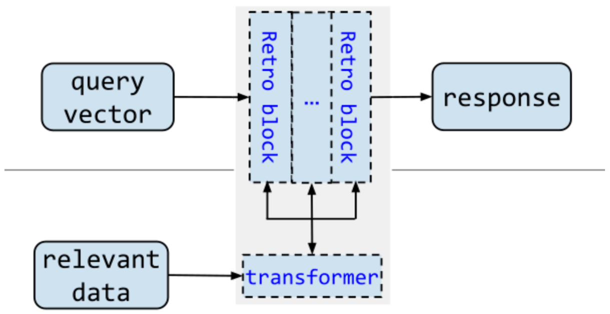
- Performance improves by optimizing how vector similarities are calculated. Dot products enhance similarity scoring, while approximate nearest neighbor (ANN) searches improve retrieval efficiency over K-nearest neighbors (KNN) searches.[8]
- Accuracy may be improved with Late Interactions, which allow the system to compare words more precisely after retrieval. This helps refine document ranking and improve search relevance.[9]
- Hybrid vector approaches may be used to combine dense vector representations with sparse one-hot vectors, taking advantage of the computational efficiency of sparse dot products over dense vector operations.[7]
- Other retrieval techniques focus on improving accuracy by refining how documents are selected. Some retrieval methods combine sparse representations, such as SPLADE, with query expansion strategies to improve search accuracy and recall.[10]

Retriever-centric methods

These methods aim to enhance the quality of document retrieval in vector databases:

- Pre-training the retriever using the *Inverse Cloze Task* (ICT), a technique that helps the model learn retrieval patterns by predicting masked text within documents.[11]
- Supervised retriever optimization aligns retrieval probabilities with the generator model’s likelihood distribution. This involves retrieving the top-k vectors for a given prompt, scoring the generated response’s perplexity, and minimizing KL divergence between the retriever’s selections and the model’s likelihoods to refine retrieval.[12]
- Reranking techniques can refine retriever performance by prioritizing the most relevant retrieved documents during training.[13]

Language model



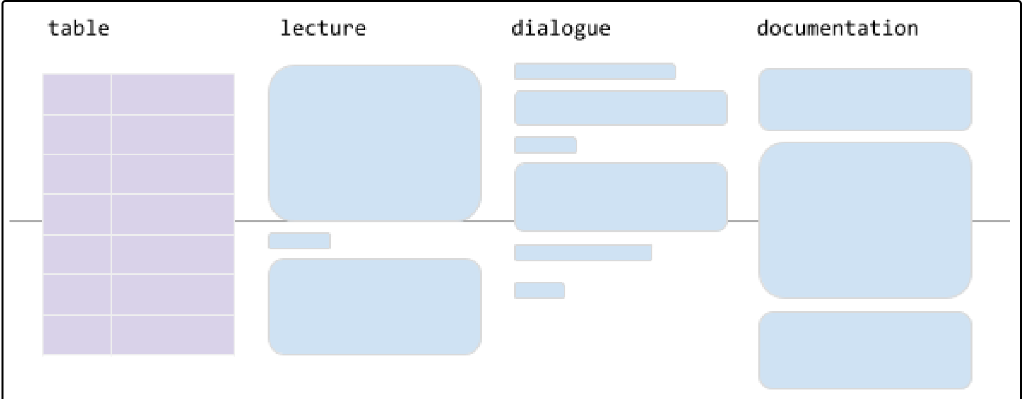
Retro language model for RAG. Each Retro block consists of Attention, Chunked Cross Attention, and Feed Forward layers. Black-lettered boxes show data being changed, and blue lettering shows the algorithm performing the changes.

By redesigning the language model with the retriever in mind, a 25-time smaller network can get comparable perplexity as its much larger counterparts.[14] Because it is trained from scratch, this method (Retro) incurs the high cost of training runs that the original RAG scheme avoided. The hypothesis is that by giving domain knowledge during training, Retro needs less focus on the domain and can devote its smaller weight resources only to language semantics. The redesigned language model is shown here.

It has been reported that Retro is not reproducible, so modifications were made to make it so. The more reproducible version is called Retro++ and includes in-context RAG.[15]

Chunking

Chunking involves various strategies for breaking up the data into vectors so the retriever can find details in it.



Different data styles have patterns that correct chunking can take advantage of.

Three types of chunking strategies are:

- Fixed length with overlap. This is fast and easy. Overlapping consecutive chunks helps to maintain semantic context across chunks.
- Syntax-based chunks can break the document up into sentences. Libraries such as spaCy or NLTK can also help.
- File format-based chunking. Certain file types have natural chunks built in, and it's best to respect them. For example, code files are best chunked and vectorized as whole functions or classes. HTML files should leave <table> or base64 encoded <img> elements intact. Similar considerations should be taken for pdf files. Libraries such as Unstructured or Langchain can assist with this method.

Hybrid search

Sometimes vector database searches can miss key facts needed to answer a user's question. One way to mitigate this is to do a traditional text search, add those results to the text chunks linked to the retrieved vectors from the vector search, and feed the combined hybrid text into the language model for generation.

Evaluation and benchmarks

RAG systems are commonly evaluated using benchmarks designed to test retrievability, retrieval accuracy and generative quality. Popular datasets include BEIR, a suite of information retrieval tasks across diverse domains, and Natural Questions or Google QA for open-domain QA.

Challenges

RAG does not prevent hallucinations in LLMs. According to *Ars Technica*, "It is not a direct solution because the LLM can still hallucinate around the source material in its response."[4]

While RAG improves the accuracy of large language models (LLMs), it does not eliminate all challenges. One limitation is that while RAG reduces the need for frequent model retraining, it does not remove it entirely. Additionally, LLMs may struggle to recognize when they lack sufficient information to provide a reliable response. Without specific training, models may generate answers even when they should indicate uncertainty. According to IBM, this issue can arise when the model lacks the ability to assess its own knowledge limitations.[1]

RAG systems may retrieve factually correct but misleading sources, leading to errors in interpretation. In some cases, an LLM may extract statements from a source without considering its context, resulting in an incorrect conclusion. Additionally, when faced with conflicting information RAG models may struggle to determine which source is accurate. The worst case outcome of this limitation is that the model may combine details from multiple sources producing responses that merge outdated and updated information in a misleading manner. According to the *MIT Technology Review*, these issues occur because RAG systems may misinterpret the data they retrieve.[2]

References

1. "What is retrieval-augmented generation?" (https://research.ibm.com/blog/retrieval-augmented-generation-RAG). *IBM*. 22 August 2023. Retrieved 7 March 2025.
2. "Why Google's AI Overviews gets things wrong" (https://www.technologyreview.com/2024/05/31/1093019/why-are-googles-ai-overviews-results-so-bad/). *MIT Technology Review*. 31 May 2024. Retrieved 7 March 2025.
3. Kiela Douwe, Lewis Patrick, Perez Ethan, Piktus Aleksandra, Petroni Fabio, Karpukhin Vladimir, Goyal Naman, Küttler Heinrich, Lewis Mike, Yih Wen-Tau, Rocktäschel Tim, Riedel Sebastian (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks* (https://dl.acm.org/doi/abs/10.5555/3495724.3496517). pp. 9459–9474. arXiv:2005.11401 (https://arxiv.org/abs/2005.11401). ISBN 978-1-7138-2954-6.
4. "Can a technology called RAG keep AI models from making stuff up?" (https://arstechnica.com/ai/2024/06/can-a-technology-called-rag-keep-ai-models-from-making-stuff-up/). *Ars Technica*. 6 June 2024. Retrieved 7 March 2025.
5. "Mitigating LLM hallucinations in text summarisation" (https://www.bbc.co.uk/rd/articles/2024-06-mitigating-llm-hallucinations-in-text-summarisation). *BBC*. 20 June 2024. Retrieved 7 March 2025.

6. Lewis, Patrick; Perez, Ethan; Piktus, Aleksandra; Petroni, Fabio; Karpukhin, Vladimir; Goyal, Naman; Küttler, Heinrich; Lewis, Mike; Yih, Wen-tau; Rocktäschel, Tim; Riedel, Sebastian; Kiela, Douwe (2020). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" (https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html). *Advances in Neural Information Processing Systems*. **33**. Curran Associates, Inc.: 9459–9474. arXiv:2005.11401 (https://arxiv.org/abs/2005.11401).
7. Luan, Yi; Eisenstein, Jacob; Toutanova, Kristina; Collins, Michael (26 April 2021). "Sparse, Dense, and Attentional Representations for Text Retrieval" (https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00369/100684/Sparse-Dense-and-Attentional-Representations-for). *Transactions of the Association for Computational Linguistics*. **9**: 329–345. arXiv:2005.00181 (https://arxiv.org/abs/2005.00181). doi:10.1162/tacl_a_00369 (https://doi.org/10.1162%2Ftacl_a_00369). Retrieved 15 March 2025.
8. "Information retrieval" (https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/rag/rag-information-retrieval). *Microsoft*. 10 January 2025. Retrieved 15 March 2025.
9. Khattab, Omar; Zaharia, Matei (2020). "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT" (https://dl.acm.org/doi/10.1145/3397271.3401075). *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. pp. 39–48. doi:10.1145/3397271.3401075 (https://doi.org/10.1145%2F3397271.3401075). ISBN 978-1-4503-8016-4.
10. Wang, Yup; Conroy, John M.; Molino, Neil; Yang, Julia; Green, Mike (2024). "Laboratory for Analytic Sciences in TREC 2024 Retrieval Augmented Generation Track" (https://trec.nist.gov/pubs/trec33/index.html). *NIST TREC 2024*. Retrieved 15 March 2025.
11. Lee, Kenton; Chang, Ming-Wei; Toutanova, Kristina (2019). " "Latent Retrieval for Weakly Supervised Open Domain Question Answering" " (https://aclanthology.org/P19-1612.pdf) (PDF).
12. Shi, Weijia; Min, Sewon; Yasunaga, Michihiro; Seo, Minjoon; James, Rich; Lewis, Mike; Zettlemoyer, Luke; Yih, Wen-tau (June 2024). "REPLUG: Retrieval-Augmented Black-Box Language Models" (https://aclanthology.org/2024.naacl-long.463/). *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. pp. 8371–8384. arXiv:2301.12652 (https://arxiv.org/abs/2301.12652). doi:10.18653/v1/2024.naacl-long.463 (https://doi.org/10.18653%2Fv1%2F2024.naacl-long.463). Retrieved 16 March 2025.
13. Ram, Ori; Levine, Yoav; Dalmedigos, Itay; Muhlgay, Dor; Shashua, Amnon; Leyton-Brown, Kevin; Shoham, Yoav (2023). "In-Context Retrieval-Augmented Language Models" (https://aclanthology.org/2023.tacl-1.75/). *Transactions of the Association for Computational Linguistics*. **11**: 1316–1331. arXiv:2302.00083 (https://arxiv.org/abs/2302.00083). doi:10.1162/tacl_a_00605 (https://doi.org/10.1162%2Ftacl_a_00605). Retrieved 16 March 2025.
14. Borgeaud, Sebastian; Mensch, Arthur (2021). "Improving language models by retrieving from trillions of tokens" (https://proceedings.mlr.press/v162/borgeaud22a/borgeaud22a.pdf) (PDF).
15. Wang, Boxin; Ping, Wei; Xu, Peng; McAfee, Lawrence; Liu, Zihan; Shoeybi, Mohammad; Dong, Yi; Kuchaiev, Oleksii; Li, Bo; Xiao, Chaowei; Anandkumar, Anima; Catanzaro, Bryan (2023). "Shall We Pretrain Autoregressive Language Models with Retrieval? A Comprehensive Study" (https://aclanthology.org/2023.emnlp-main.482/). *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. pp. 7763–7786. doi:10.18653/v1/2023.emnlp-main.482 (https://doi.org/10.18653%2Fv1%2F2023.emnlp-main.482).