← Learn

Retrieval Augmented Generation (RAG)

Zachary Proser

Aug 3, 2023

Share:

Core Components

Jump to section

What is retrieval augmented generation (RAG)?

Retrieval augmented generation (RAG) is an architecture that provides the most relevant and contextually-important proprietary, private or dynamic data to your Generative AI application's large language model (LLM) when it is performing tasks to enhance its accuracy and performance.

What is RAG in AI / LLM?

Retrieval Augmented Generation (RAG) in AI is a technique that leverages a database to fetch the most contextually relevant results that match the user's query at generation time.

Products built on top of Large Language Models (LLMs) such as OpenAI's ChatGPT and Anthropic's Claude are brilliant yet flawed. Powerful as they are, current LLMs suffer from several drawbacks:

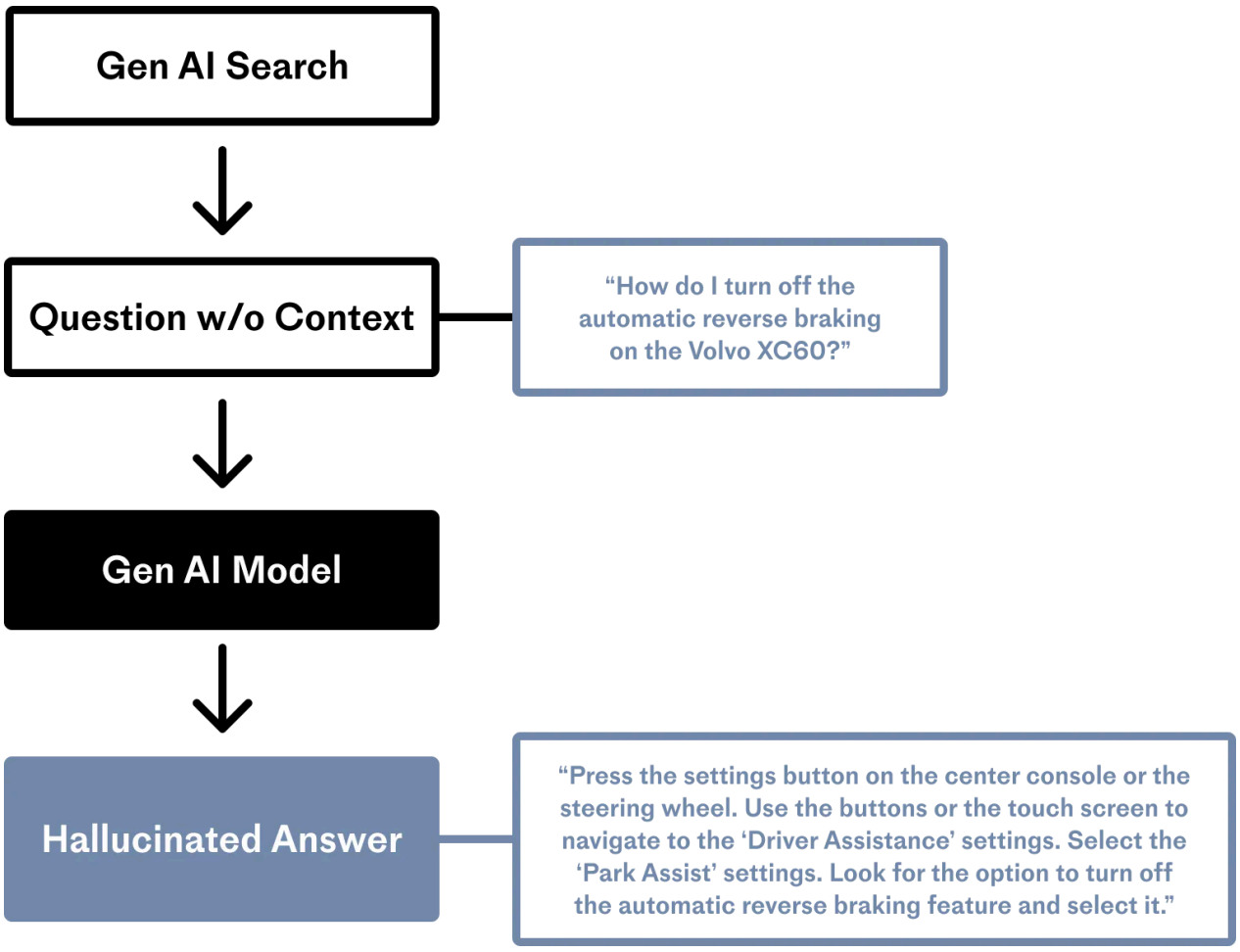
1. **They are static** - LLMs are “frozen in time” and lack up-to-date information. It is not feasible to update their gigantic training datasets.
2. **They lack domain-specific knowledge** - LLMs are trained for generalized tasks, meaning they do not know your company’s private data.
3. **They function as “black boxes”** - it’s not easy to understand which sources an LLM was considering when they arrived at their conclusions.
4. **They are inefficient and costly to produce** - Few organizations have the financial and human resources to produce and deploy foundation models.

Unfortunately, these issues affect the accuracy of GenAI applications leveraging LLMs. For any business application with requirements more demanding than your average chatbot demo, LLMs used “out of the box” with no modifications aside from prompting will perform poorly at context-dependent tasks, such as helping customers book their next flight.

This post will examine why Retrieval Augmented Generation is the preferred method for addressing these drawbacks, provide a deep dive into how RAG works, and explain why most companies use RAG to boost their GenAI applications’ performance.

LLMs are “stuck” at a particular time, but RAG can bring them into the present.

ChatGPT’s training data “cutoff point” was September 2021. If you ask ChatGPT about something that occurred last month, it will not only fail to answer your question factually; it will likely dream up a very convincing-sounding outright lie. We commonly refer to this behavior as “hallucination.”



The LLM lacks domain-specific information about the Volvo XC60 in the above example. Although the LLM has no idea how to turn off reverse braking for that car

model, it performs its generative task to the best of its ability anyway, producing an answer that sounds grammatically solid - but is unfortunately flatly incorrect.

The reason LLMs like ChatGPT feel so bright is that they've seen an immense amount of human creative output - entire companies' worth of open source code, libraries worth of books, lifetimes of conversations, scientific datasets, etc., but, critically, this core training data is static.

After OpenAI finished training the foundation model (FM) behind ChatGPT in 2021, it received no additional updated information about the world; it remains oblivious to significant world events, weather systems, new laws, the outcomes of sporting events, and, as demonstrated in the image above, the operating procedures for the latest automobiles.

RAG provides up-to-date information about the world and domain-specific data to your GenAI applications.

Retrieval Augmented Generation means fetching up-to-date or context-specific data from an external database and making it available to an LLM when asking it to generate a response, solving this problem. You can store proprietary business data or information about the world and have your application fetch it for the LLM at generation time, reducing the likelihood of hallucinations. The result is a noticeable boost in the performance and accuracy of your GenAI application.

LLMs lack context from private data - leading to hallucinations when asked domain or company-specific questions.

The second major drawback of current LLMs is that, although their base corpus of knowledge is impressive, they do not know the specifics of your business, your requirements, your customer base, or the context your application is running in - such as your e-commerce store.

RAG addresses this second issue by providing extra context and factual information to your GenAI application's LLM at generation time: anything from customer records to paragraphs of dialogue in a play, to product specifications and current stock, to audio

such as voice or songs. The LLM uses this provided content to generate an informed answer.

Retrieval Augmented Generation allows GenAI to cite its sources and improves auditability.

In addition to addressing the recency and domain-specific data issues, RAG also allows GenAI applications to provide their sources, much like research papers will provide citations for where they obtained an essential piece of data used in their findings.

Imagine a GenAI application serving the legal industry by helping lawyers prepare arguments. The GenAI application will ultimately present its recommendations as final outputs. Still, RAG enables it to provide citations of the legal precedents, local laws, and the evidence it used when arriving at its proposals.

RAG makes the inner workings of GenAI applications easier to audit and understand. It allows end users to jump straight into the same source documents the LLM used when creating its answers.

Why is RAG the preferred approach from a cost-efficacy perspective?

There are three main options for improving the performance of your GenAI application other than RAG. Let's examine them to understand why RAG remains the main path most companies take today.

Create your own foundation model

OpenAI's Sam Altman estimated it cost around $100 million to train the foundation model behind ChatGPT.

Not every company or model will require such a significant investment, but ChatGPT's price tag underscores that cost is a real challenge in producing sophisticated models with today's techniques.

In addition to raw compute costs, you'll also face the scarce talent issue: you need specialized teams of machine learning PhDs, top-notch systems engineers, and highly skilled operations folks to tackle the many technical challenges of producing such a model and every other AI company in the world is angling for the same rare talent.

Another challenge is obtaining, sanitizing, and labeling the datasets required to produce a capable foundation model. For example, suppose you're a legal discovery company considering training your model to answer questions about legal documents. In that case, you'll also need legal experts to spend many hours labeling training data.

Even if you have access to sufficient capital, can assemble the right team, obtain and label adequate datasets and overcome the many technical hurdles to hosting your model in production, there's no guarantee of success. The industry has seen several ambitious AI startups come and go, and we expect to see more failures.

Fine-tuning: adapting a foundation model to your domain's data.

Fine-tuning is the process of retraining a foundation model on new data. It can certainly be cheaper than building a foundation model from scratch. Still, this approach suffers from many of the same downsides of creating a foundation model: you need rare and deep expertise and sufficient data, and the costs and technical complexity of hosting your model in production don't go away.

Fine-tuning is not a practical approach now that LLMs are pairable with vector databases for context retrieval. Some LLM providers, such as OpenAI, no longer support fine-tuning for their latest-generation models.

Fine-tuning is an outdated method of improving LLM outputs. It required recurring, costly, and time-intensive labeling work by subject-matter experts and constant monitoring for quality drift, undesirable deviations in the accuracy of a model due to a lack of regular updates, or changes in data distribution.

If your data changes over time, even a fine-tuned model's accuracy can drop, requiring more costly and time-intensive data labeling, constant quality monitoring, and repeated fine-tuning.

Imagine updating your model every time you sell a car so your GenAI application has the most recent inventory data.

Prompt engineering is insufficient for reducing hallucinations.

Prompt engineering means testing and tweaking the instructions you provide your model to attempt to coax it to do what you want.

It's also the cheapest option to improve the accuracy of your GenAI application because you can quickly update the instructions provided to your GenAI application's LLM with a few code changes.

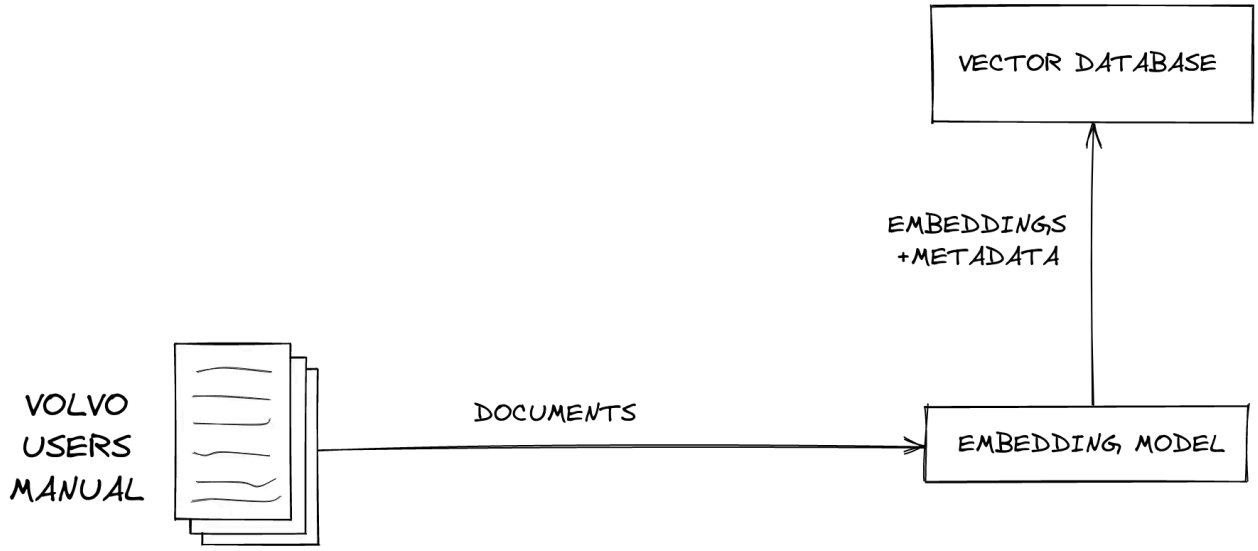
It refines the responses your LLMs return but cannot provide them with any new or dynamic context, so your GenAI application will still lack up-to-date context and be susceptible to hallucination.

Let's now take a deeper dive into how Retrieval Augmented Generation works.

We've discussed how RAG passes additional relevant content from your domain-specific database to an LLM at generation time, alongside the original prompt or question, through a "context window".

An LLM's context window is its field of vision at a given moment. RAG is like holding up a cue card containing the critical points for your LLM to see, helping it produce more accurate responses incorporating essential data.

To understand RAG, we must first understand semantic search, which attempts to find the true meaning of the user's query and retrieve relevant information instead of simply matching keywords in the user's query. Semantic search aims to deliver results that better fit the user's intent, not just their exact words.



Creating a vector database from your domain-specific proprietary data using an embedding model.

This diagram shows how you make a vector database from your domain-specific, proprietary data. To create your vector database, you convert your data into vectors by running it through an embedding model.

An embedding model is a type of LLM that converts data into vectors: arrays, or groups, of numbers. In the above example, we're converting user manuals containing the ground truth for operating the latest Volvo vehicle, but your data could be text, images, video, or audio.

The most important thing to understand is that a vector represents the meaning of the input text, the same way another human would understand the essence if you spoke the text aloud. We convert our data to vectors so that computers can search for semantically similar items based on the numerical representation of the stored data.

Next, you put the vectors into a vector database, like Pinecone. Pinecone's vector database can search billions of items for similar matches in under a second.

Remember that you can create vectors, ingest the vectors into the database, and update the index in real-time, solving the recency problem for the LLMs in your GenAI applications.

For example, you can write code that automatically creates vectors for your latest product offering and then upserts them in your index each time you launch a new product. Your company's support chatbot application can then use RAG to retrieve up-

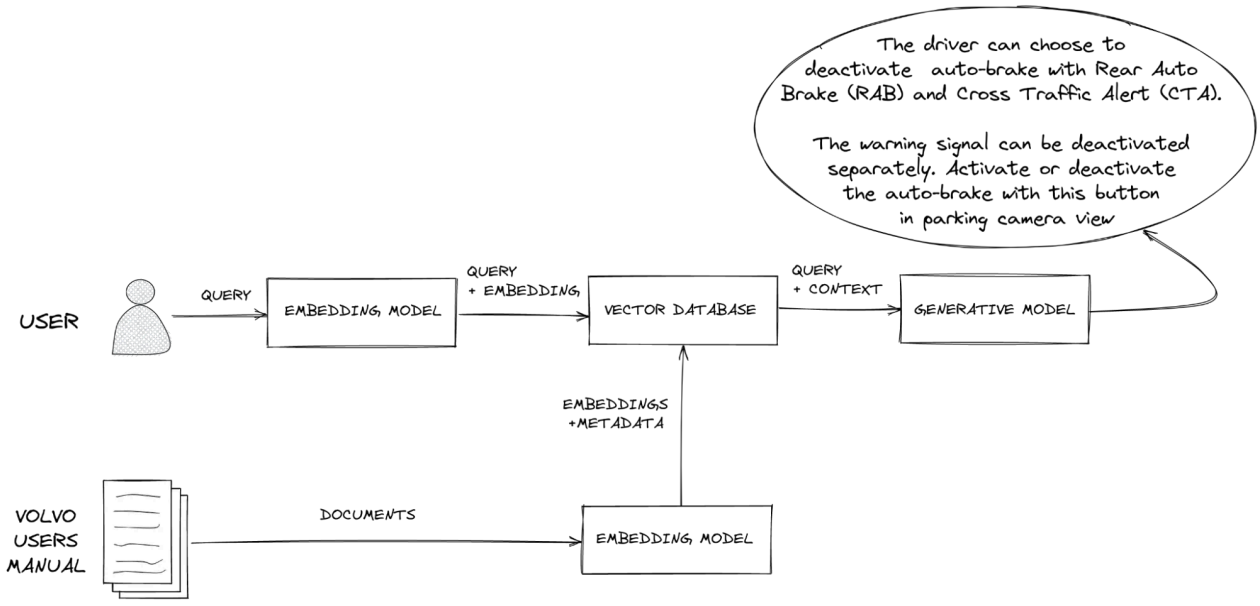
to-date information about product availability and data about the current customer it's chatting with.

Vector databases allow you to query data using natural language, which is ideal for chat interfaces.

Now that your vector database contains numerical representations of your target data, you can perform a semantic search. Vector databases shine in semantic search use cases because end users form queries with ambiguous natural language.

Semantic search works by converting the user's query into embeddings and using a vector database to search for similar entries.

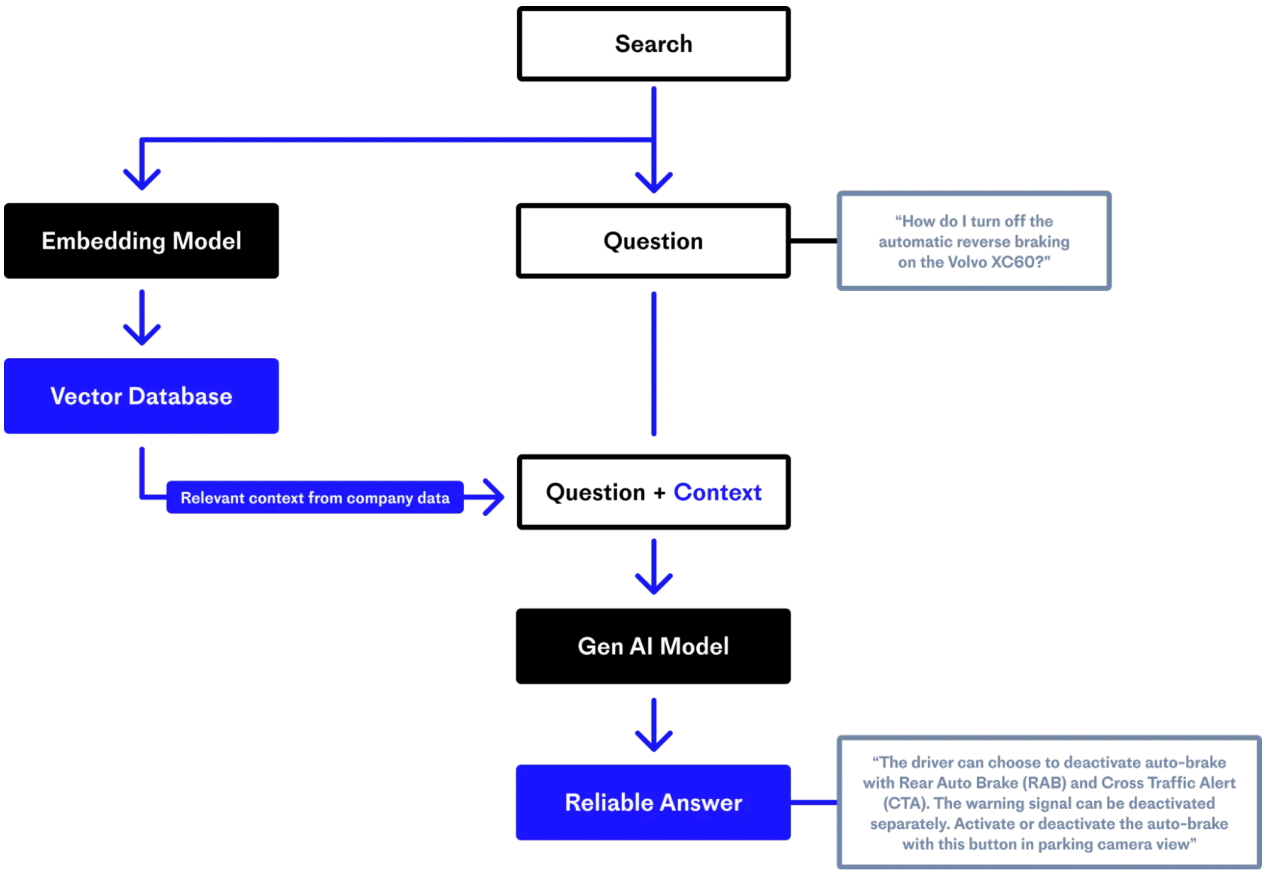
Let's see what happens to our retrieval augmented generation example when we are able to inject context dynamically:



Retrieval Augmented Generation (RAG) uses semantic search to retrieve relevant and timely context that LLMs use to produce more accurate responses.

You originally converted your proprietary data into embeddings. When the user issues a query or question, you translate their natural language search terms into embeddings.

You send these embeddings to the vector database. The database performs a “nearest neighbor” search, finding the vectors that most closely resemble the user’s intent. When the vector database returns the relevant results, your application provides them to the LLM via its context window, prompting it to perform its generative task.



Retrieval Augmented Generation reduces the likelihood of hallucinations by providing domain-specific information through an LLM's context window.

Since the LLM now has access to the most pertinent and grounding facts from your vector database, your rag application can provide an accurate answer for your user. RAG reduces the likelihood of hallucination.

Vector databases can support even more advanced search functionality

Semantic search is powerful, but it’s posble to go even further. For example, Pinecone’s vector database supports hybrid search functionality, a retrieval system that considers the query's semantics and keywords.

RAG is the most cost-effective, easy to implement, and lowest-risk path to higher performance for GenAI applications.

Semantic search and Retrieval Augmented Generation provide more relevant GenAI responses, translating to a superior experience for end-users. Unlike building your foundation model, fine-tuning an existing model, or solely performing prompt engineering, RAG simultaneously addresses recency and context-specific issues cost-effectively and with lower risk than alternative approaches.

Its primary purpose is to provide context-sensitive, detailed answers to questions that require access to private data to answer correctly.

Pinecone enables you to integrate RAG within minutes. Check out our examples repository on GitHub for runnable examples, such as this RAG Jupyter Notebook.

Share:

Further Reading

LangGraph and Research Agents

Build Privacy-aware AI software using Pinecone

The Practitioner's Guide To E5

Pinecone

Product

- Overview
- Documentation
- Integrations
- Trust and Security

Solutions

- Customers
- RAG
- Semantic Search
- Multi-Modal Search
- Candidate Generation
- Classification

Resources

- Learning Center
- Community
- Pinecone Blog
- Support Center
- System Status
- What is a Vector Database?
- What is Retrieval Augmented Generation (RAG)?

Company

- About
- Partners
- Careers
- Newsroom
- Contact

Legal

- Customer Terms
- Website Terms
- Privacy
- Cookies
- Cookie Preferences