← Learn

# Retrieval-Augmented Generation (RAG)

Jenna Pederson

Jun 12, 2025

Core Components

Share:

Not only are foundation models stuck in the past, but they intentionally produce natural-sounding and varied responses. Both of these can lead to confidently inaccurate and irrelevant output. This behavior is known as “hallucination.”

In this article, we’ll explore the limitations of foundation models and how retrieval-augmented generation (RAG) can address these limitations so chat, search, and agentic workflows can all benefit.

## Limitations of foundation models

Products built on top of foundation models alone are brilliant yet flawed as foundation models have multiple limitations:

## Knowledge cutoffs

When you ask current models about recent events – like asking about last week's NBA basketball game or how to use features in the latest iPhone model - they may confidently provide outdated or completely fabricated information, the hallucinations we mentioned earlier.

Models are trained on massive datasets containing years of human knowledge and creative output from code repositories, books, websites, conversations, scientific papers, and more. But after a model is trained, this data is frozen at a specific point in time, the "cutoff". This cutoff creates a *knowledge gap*, leading them to generate plausible but incorrect responses when asked about recent developments.

## Lack depth in domain-specific knowledge

Foundation models have broad knowledge, but can lack depth in specialized domains. High quality datasets might not exist publicly for a domain, not necessarily because they are private, but because they are highly specialized. Consider a medical model that knows about anatomy, disease, and surgical techniques, but struggles with rare genetic conditions and cutting edge therapies. This data might exist publicly to be used during training, but it may not appear enough to train the model correctly. It also requires expert-level knowledge during the training process to contextualize the information.

This limitation can result in responses that are incomplete or irrelevant.

## Lack private or proprietary data

In the case of general-purpose, public models, the data (*your* data) does not exist publicly and is inaccessible during training. This means that models don't know the specifics of your business, whether that be internal company processes and policies, personnel data or email communications, or even the trade secrets of your company. And for good reason: if this data had been included in the training, anyone using the model would potentially gain access to your company's private and proprietary data.

Again, this limitation can result in incomplete or irrelevant responses, limiting the usefulness of the model for your custom business purpose.

### Loses trust

Models typically cannot cite their sources related to a specific response. Without citations or references, the user either has to trust the response or validate the claim themselves. Given that models are trained on vast amounts of public data, there is a chance that the generated response is the result of an unauthoritative source.

When inaccurate, irrelevant, and useless information is generated, users will lose trust in the model itself, even when this behavior is inherent in how foundation models work.

### Output generation is probabilistic

Hallucinations are often a symptom of the limitations just described. However, models are trained on a diverse set of data that can contain contradictions, errors, and ambiguous data (in addition to the correct data). Because of this, models assign probabilities to all possible continuations, even the wrong ones. Because of sampling randomness like temperature and top k combined with how a user constructs a prompt (maybe it's too vague or contains misleading context), models may choose the wrong continuation. The result is output that can contain hallucinations.

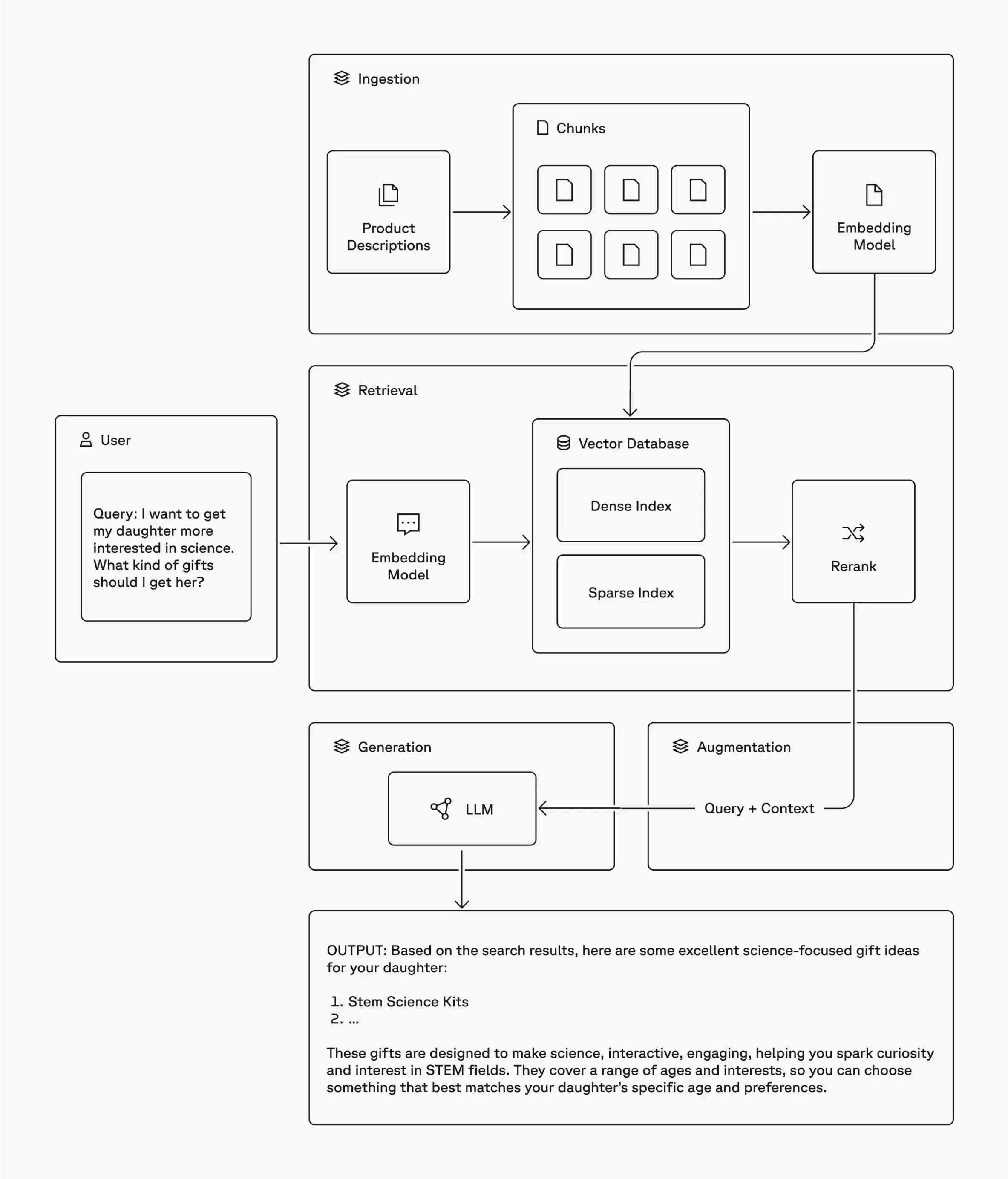
Additionally, models don't always distinguish between what they know vs what they don't know, sounding confident even when incomplete, inaccurate, or irrelevant. Hallucinations can produce unwanted behaviors and even be dangerous. For example, an inaccurate but highly convincing medical report could lead to life-threatening treatments or no treatment at all.

These foundation model limitations can impact your business bottom line and erode the trust of your users. Retrieval-augmented generation can address these limitations.

## What is Retrieval-Augmented Generation?

Retrieval-augmented generation, or RAG, is a technique that uses authoritative, external data to improve the accuracy, relevancy, and usefulness of a model's output. It does this through the following four core components, which we'll cover in more detail later in this article:

1. Ingestion: authoritative data like company proprietary data is loaded into a data source, like a Pinecone vector database
2. Retrieval: relevant data is retrieved from an external data source based on a user query
3. Augmentation: the retrieved data and the user query are combined into a prompt to provide the model with context for the generation step
4. Generation: the model generates output from the augmented prompt, using the context to drive a more accurate and relevant response.



OUTPUT: Based on the search results, here are some excellent science-focused gift ideas for your daughter:

1. Stem Science Kits
2. ...

These gifts are designed to make science, interactive, engaging, helping you spark curiosity and interest in STEM fields. They cover a range of ages and interests, so you can choose something that best matches your daughter's specific age and preferences.

Traditional RAG

By combining relevant data from an external data source with the user's query and providing it to the model as context for the generation step, the model will use it to generate a more accurate and relevant output.

RAG provides the following benefits:

1. Access to real-time data and proprietary or domain-specific data: bring in knowledge relevant to your situation - current events, news, social media, customer data, proprietary data
2. Builds trust: more relevant and accurate results are more likely to earn trust and source citations allow human review
3. More control: control over which sources are used, real-time data access, authorization to data, guardrails/safety/compliance, traceability/source citations, retrieval strategies, cost, tune each component independently of the others
4. Cost-effective compared to alternatives like training/re-training your own model, fine-tuning, or stuffing the context window: foundation models are costly to produce and require specialized knowledge to create, as is fine-tuning; the larger the context sent to the model, the higher the cost

## RAG in support of agentic workflows

But this traditional RAG approach is simple, often with a vector database and a one-shot prompt with context sent to the model to generate output. With the rise of AI agents, agents are now orchestrators of the core RAG components to:

- construct more effective queries
- access additional retrieval tools
- evaluate the accuracy and relevance of the retrieved context
- apply reasoning to validate retrieved information, to trust or discard it.

These operations can be performed by an agent or agents as part of a larger, iterative plan. Agents as orchestrators of RAG bring even more opportunities for review, revision of queries, reasoning or validation of context, allowing them to make better decisions, take more informed actions, and generate more accurate and relevant output.

Now that we've covered what RAG is, let's take a deeper dive into how it works.

# How does Retrieval-Augmented Generation work?

RAG brings accuracy and relevancy to LLM output by relying on authoritative data sources like proprietary, domain-specific data and real-time information. But before we dig into how it does that, let’s ask the questions: do you even need RAG and how will you know it’s working?

This is where ground truth evaluations come in. In order to properly deploy any application, you need to know when it's working. AI applications are no different, and so identifying a set of queries and their expected answers is critical to knowing if your application is working. Maintaining that evaluation set is also critical to knowing where to improve over time, and whether those improvements are working. RAG itself is just one optimization, but there are others like query rewriting, chunk expansion, knowledge graphs and more.

With a good baseline, you can move on to implementing the four main components of RAG:

## Ingestion

In simple traditional RAG, you’ll retrieve data from a vector database like Pinecone, using semantic search to find the true meaning of the user’s query and retrieve relevant information instead of simply matching keywords in the query. We’ll use Pinecone as an example here, but the concept applies to all vector databases.

But before we can retrieve the data, you have to ingest the data. Here are steps to get data into your database:

### *Chunk the data*

During the ingestion step, you’ll load your authoritative data as vectors into Pinecone. You may have structured or unstructured data in the form of text, PDFs, emails, internal wikis, or databases. After cleaning the data, you may need to chunk it by dividing each piece of data, or document, into smaller chunks. Depending on the kind of data you have, the types of queries your users have, and how the results will be used in your application, you’ll need to choose a chunking strategy.

### *Create vector embeddings*

Then, using an embedding model, you'll embed each chunk and load it into the vector database. The embedding model is a special type of LLM that converts the data chunk into a vector embedding, a numerical representation of the data's meaning. This allows computers to search for similar items based on the vector representation of the stored data.

### *Load data into a vector database*

Once you have vectors, you'll load them into a vector database. This ingestion step most likely happens offline, independently of your application and your user's workflow. However, if your data changes, for instance, product inventory is updated, you can update the index in real-time to provide up-to-date information to your users.

Now that your vector database contains the vector embeddings of your source data, the next step is retrieval.

## Retrieval

A simple approach to retrieval would use semantic search alone. But by using hybrid search, combining both semantic search (with dense vectors) and lexical search (with sparse vectors), you can improve the retrieval results even more. This becomes relevant when your users don't always use the same language to talk about a topic (semantic search) and they refer to internal, domain-specific language (lexical or keyword search) like acronyms, product names, or team names.

During retrieval, we'll create a vector embedding from the user's query to use for searching against the vectors in the database. In hybrid search, you'll query either a single hybrid index or both a dense and a sparse index. Then we combine and de-duplicate the results and use a reranking model to rerank them based on a unified relevance score, and return the most relevant matches.

## Augmentation

Now that you have the most relevant matches from the retrieval step, you'll create an augmented prompt with both the search results and the user's query to send to the LLM. This is where the magic happens.

An augmented prompt might look like this:

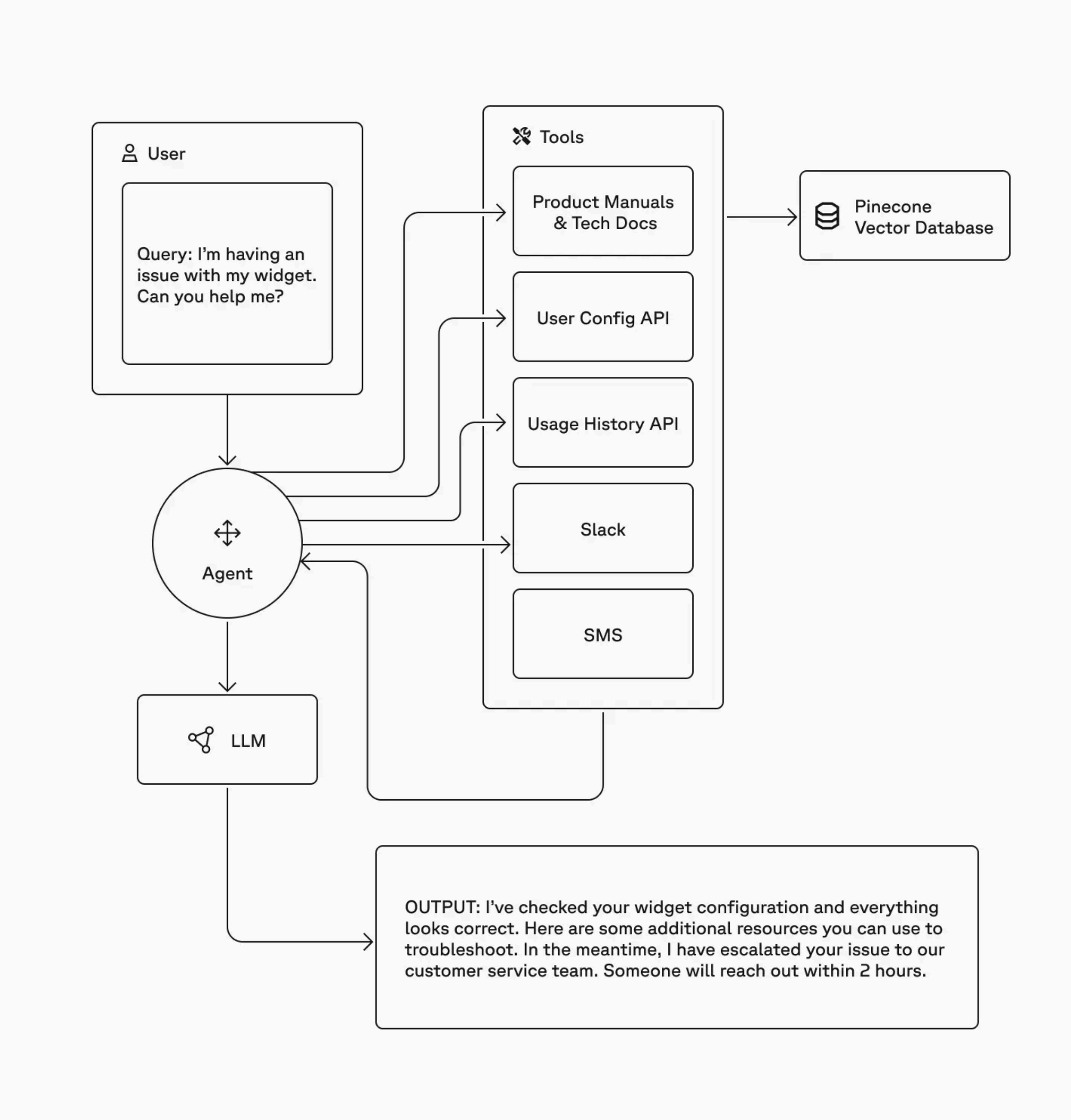
```
QUESTION:
<the user's question>

CONTEXT:
<the search results to use as context>

Using the CONTEXT provided, answer the QUESTION. Keep your answer grounded i
```

By sending both the search results and the user’s question as context to the LLM, you are encouraging it to use the more accurate and relevant info from the search results during the next generation step.

## Generation

Using the augmented prompt, the LLM now has access to the most pertinent and grounding facts from your vector database so your application can provide an accurate answer for your user, reducing the likelihood of hallucination.

But RAG is no longer simply about searching for the right piece of information to inform a model response. With agentic RAG, it's about deciding which questions to ask, which tools to use, when to use them, and then aggregating results to ground answers.


User
Query: I'm having an issue with my widget. Can you help me?
Tools
Product Manuals & Tech Docs
User Config API
Usage History API
Slack
SMS
Pinecone Vector Database
Agent
LLM
OUTPUT: I've checked your widget configuration and everything looks correct. Here are some additional resources you can use to troubleshoot. In the meantime, I have escalated your issue to our customer service team. Someone will reach out within 2 hours.

Agentic RAG

In this simple version, the LLM is the agent and decides which retrieval tools to use and when, and how to query those tools.

## Wrapping up

Retrieval-augmented generation has evolved from a buzzword to an indispensable foundation for AI applications. It blends the broad capabilities of foundation models with your company's authoritative and proprietary knowledge. With AI agents handling more complex use cases, from those supporting professionals servicing complex manufacturing equipment to delivering domain-specific agents at scale, RAG is not just relevant in 2025. It's critical for building accurate, relevant, and responsible AI applications that go beyond information retrieval. As AI agents become more autonomous and handle more complex workflows, they'll need to ground their reasoning in your private and domain-specific data through RAG. The question is no longer whether to implement RAG, but how to architect it most effectively for your unique use case and data requirements.

Want to dig into a RAG code example? Create a free Pinecone account and check out our example notebooks to implement retrieval-augmented generation with Pinecone or get started with Pinecone Assistant, to build production-grade chat and agent-based applications quickly.

Share:

𝕏 in Y

Was this article helpful?

Yes No

RECOMMENDED FOR YOU

## Further Reading

Learn Jun 25, 2025

### Beyond the hype: Why RAG remains essential for modern AI

Jenna Pederson

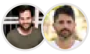
Learn Oct 25, 2024

### Building a reliable, curated, and accurate RAG system with Cleanlab and Pinecone

Matt Turk

Engineering Jan 16, 2024

### RAG makes LLMs better and equal


Amnon, Roy, Ilai, Nathan, Amir

Pinecone

Product Resources Company

Vector Database

Assistant

Documentation

Pricing

Security

Integrations

Community Forum

Learning Center

Blog

Customer Case Studies

Status

What is a Vector DB?

What is RAG?

About

Partners

Careers

Newsroom

Contact

Legal

Customer Terms

Website Terms

Privacy

Cookies

Cookie Preferences