

eBook

A Compact Guide to Retrieval Augmented Generation (RAG)

Definitions, components and basics for practitioners



Contents

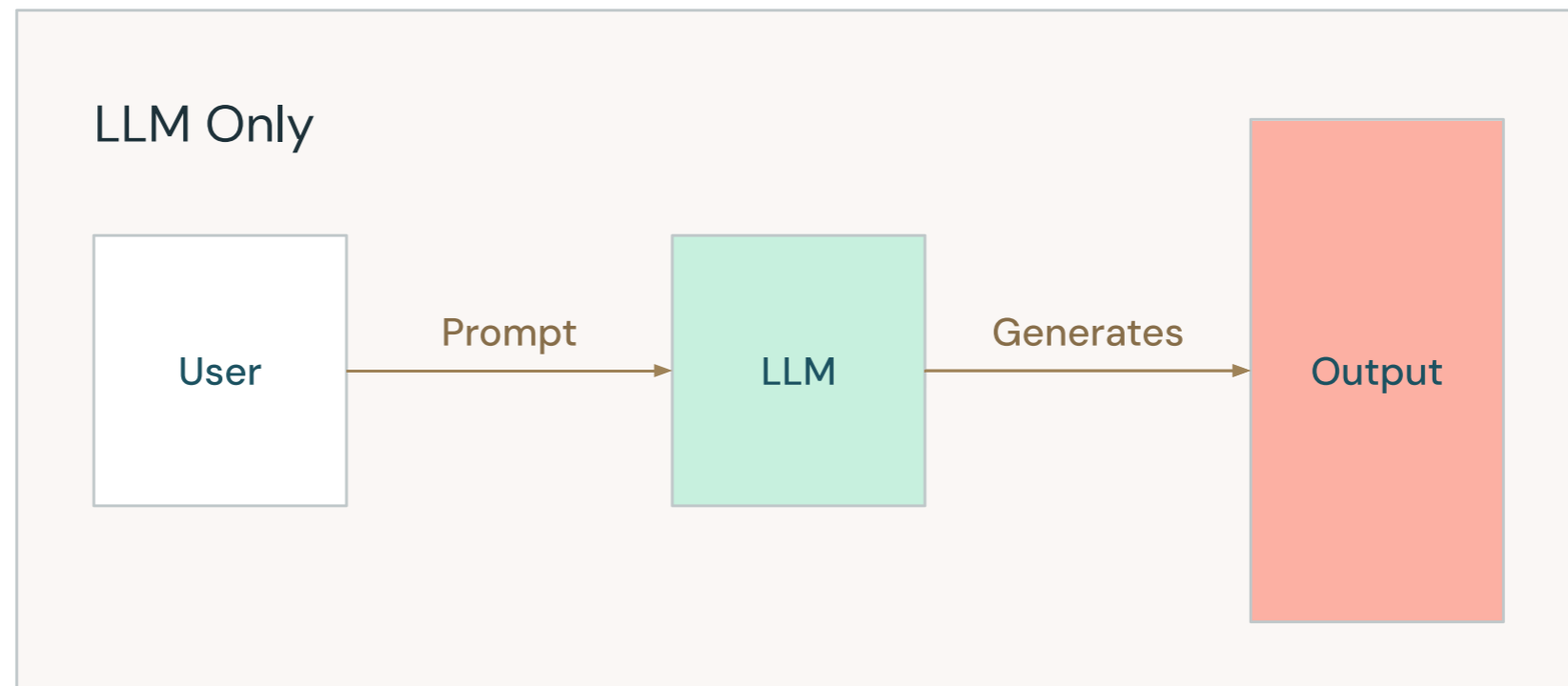
Introduction: Retrieval Augmented Generation (RAG) With Vector Search	3
LLMs and prompts.....	5
Vector Search and embedding models.....	6
Asking RAG About Databricks Asset Bundles (DABs)	8
Asking an LLM without RAG about DABs.....	8
Using RAG to give an LLM access to documentation about DABs.....	10
Addressing the Shortcomings of LLMs With RAG	11
RAG compared to LLM-only approaches.....	12
RAG Use Cases	14
Question-answering systems.....	14
Customer service.....	14
Content generation.....	14
Code assistance.....	14
RAG With Vector Search — Step by Step	15
Data preparation: Getting an external information source into a vector database.....	15
Retrieval: Getting relevant context.....	19
Augmentation: Adding context to the user’s prompt.....	23
Generation: Producing useful output with an LLM.....	27
Evaluation: Measuring RAG performance.....	30
Utilizing RAG With Other Modeling and Model Customization Methods	31
Prompt engineering.....	31
Fine-tuning.....	32
Pretraining.....	33
RAG.....	33
Combinations of methods.....	34
RAG on Databricks	35
Lakehouse architecture.....	35
Vector Search.....	35
Model serving.....	35
MLflow.....	36
Lakehouse Monitoring.....	36
Summary	37
GenAI training.....	37
Additional resources.....	37

Introduction

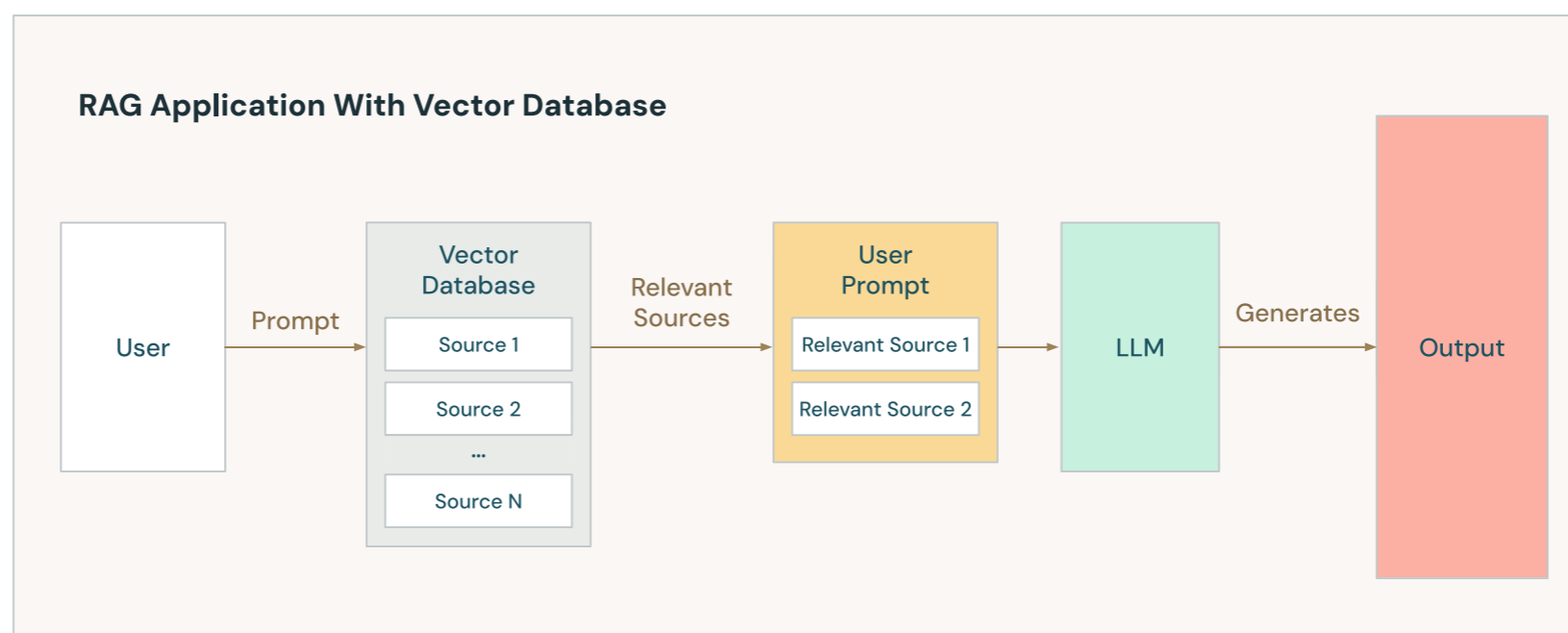
Retrieval Augmented Generation (RAG) With Vector Search

Retrieval augmented generation (RAG) is the process of combining a user's prompt with relevant external information to form a new, expanded prompt for a large language model (LLM) such as GPT-4 or Llama 2. The expanded prompt enables the LLM to provide more relevant, timely and accurate responses.

LLMs offer powerful language understanding and generation capabilities, but they aren't reliable information sources and they lack access to proprietary information or any other information they weren't trained on. They are also prone to so-called "hallucinations" — fabricating answers instead of acknowledging that they don't know the correct answer.



Using RAG with an LLM helps to address some of these issues. Giving the LLM all the information it needs to answer a question enables it to provide answers about topics it was not trained on and reduces the likelihood of hallucinations. For example, an LLM trained on public data cannot answer any questions about a company's internal memos or project documents. It's likely to hallucinate if asked about such documents. A RAG application can supply all or parts of those documents to the LLM, giving it the context it needs to give correct and relevant answers.



RAG can work with a variety of data sources, including text, podcasts, videos, live search results and structured databases. In this document, we look at RAG over stored unstructured data such as PDFs, scraped web pages or code. This guide focuses on a RAG approach in which data is retrieved from vector databases using a process known as Vector Search.

We'll start with some brief background information on LLMs and prompts, which form the foundation for RAG.

LLMs and prompts

LLMs are a relatively new class of machine learning algorithms that can interpret, manipulate and generate text-based content. They're trained on massive text datasets from diverse sources, including books, text scraped from the internet and code repositories. During the training process, the model learns statistical relationships between words and phrases, enabling it to generate new text using the context of text it has already seen or generated.

LLMs are typically used via "prompting." A prompt is text that a user provides to an LLM and that the LLM responds to. Prompts can take many different forms. Some models are trained to complete text, so prompts to these models take the form of incomplete statements like "Jack and Jill went up the hill to..." which the model then continues. Other models expect questions or instructions as prompts. These models can answer questions such as, "What happened to Jack after Jack and Jill went up the hill?" RAG applications that enable users to ask questions about text generally use instruction-following and question-answering LLMs.

LLMs can typically handle prompts of at least several paragraphs in length. This is important for RAG. In RAG, the user's question or instruction is combined with some information retrieved from an external data source, forming the new, *augmented* prompt.

Next, let's discuss where the external information is retrieved from.



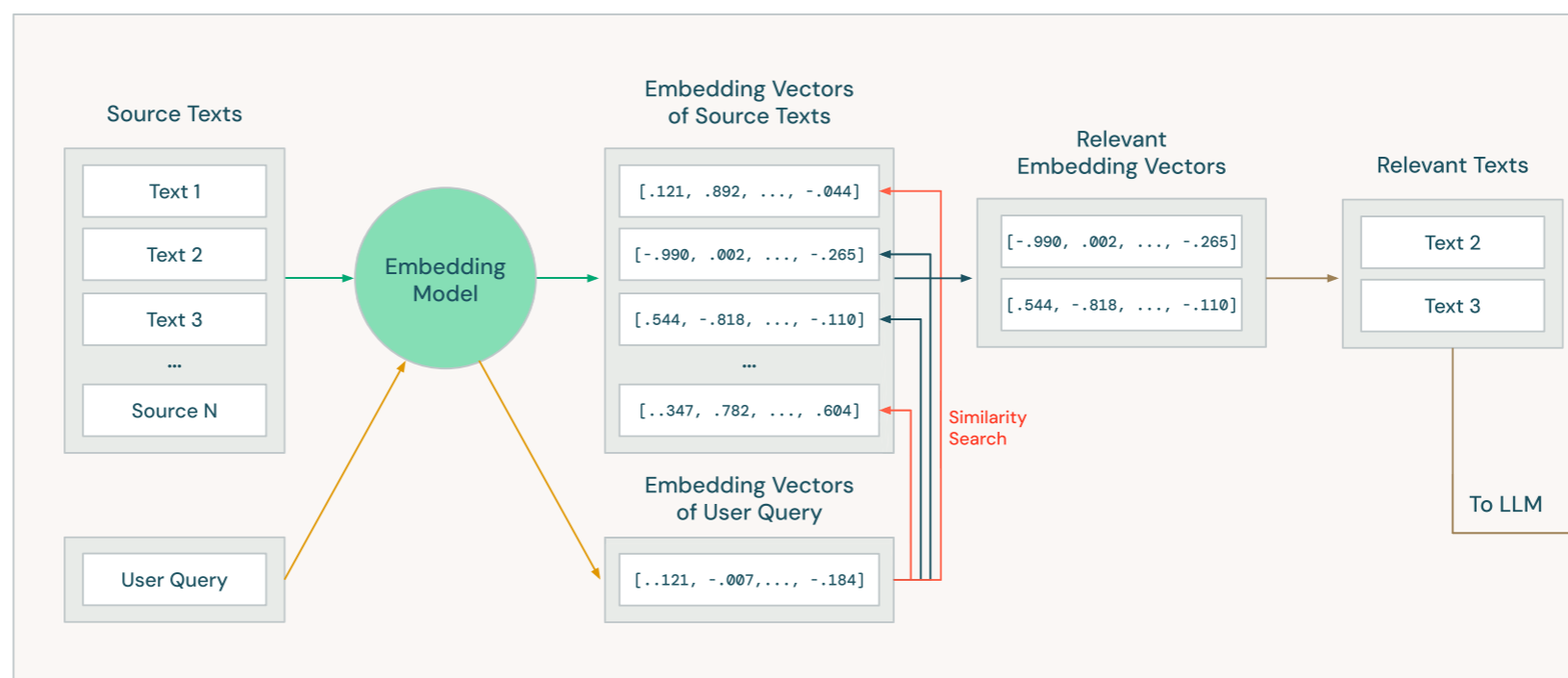
Vector Search and embedding models

An effective RAG application must be able to find information relevant to the user's prompt and supply it to the LLM. Selecting the most relevant texts from potentially millions of documents can be a significant challenge. To address this challenge, we use a technique called Vector Search to identify text relevant to the user's prompt.

In a RAG system with Vector Search, a special type of language model called an "embedding model" translates each text we want to search into a numeric vector, a series of numbers that encapsulates the text's meaning. The same model also converts the user's query to a comparable vector. This process makes it possible to mathematically compare the user's query to the text and identify those that are the most similar and the most relevant.

Crucially, these vectors represent the *meanings* of the text from which they are generated. Thus, when we search for relevant vectors, we are searching on the basis of meaning. This approach enables us to retrieve the text most relevant to the user's query and pass it to the LLM along with the user's original query.

We've talked a lot about embeddings encoding the meanings of texts. It's important to note that this is the meaning according to the model. The models are trained to capture meanings and relationships that humans care about, but the meaning captured by the embedding model may not be exactly what we want. This is one reason why it's essential to test and evaluate every component of a RAG application.



VECTOR DATABASES

The vectors generated by embedding models are often stored in a specialized *vector database*. Vector databases are optimized for storing and retrieving vector data efficiently. Like traditional databases, vector databases can be used to manage permissions, metadata and data integrity, ensuring secure and organized access to information. They also tend to include update mechanisms so newly added texts are indexed and ready to use quickly.

Asking RAG About Databricks Asset Bundles (DABs)

Suppose we want to ask some questions about **Databricks Asset Bundles** (DABs), a new Databricks feature that enables an infrastructure as code (IaC) approach to managing Databricks projects. We'll use the Llama 2 LLM, which doesn't have access to up-to-date information about DAB because it was trained before the DAB Public Preview was released.

All the following results are actual results generated by an LLM or by the vector database.

Asking an LLM without RAG about DABs

Let's see what happens if we ask the Llama 2-70B-chat LLM a couple questions about Databricks Asset Bundles without providing it with any documentation. In this case, the LLM is being used alone, not in the context of a RAG application.

```
1 User: What are databricks DABs?
2 Bot: Databricks DABs (Databricks Automation Building blocks) are a set of pre-built, reusable components that enable
3 users to automate and streamline their data engineering workflows on the Databricks platform. DABs provide a way to
4 encapsulate and reuse common data engineering patterns, such as data ingestion, transformation, and loading, across
5 different use cases and projects.
6
7 DABs are designed to be modular and can be combined to create more complex workflows. They provide a way to define
8 repeatable data engineering tasks that can be shared across different teams and projects, improving productivity and
reducing the risk of errors.
```

In this example, the model just guessed, stating incorrectly that DAB stands for "Databricks Automation Building blocks" and expanding on what those might be. Notably, it never indicated that it didn't know the answer or that it was making things up. This could represent a real issue in a customer support setting.

Let's try another question.

```
1 User: What version of the databricks CLI do I need in order to use Databricks Asset Bundles?
2 Bot: To use Databricks Asset Bundles, you need to have a Databricks CLI version that is greater than or equal to
3 1.10.0.
4 You can check your current version of the Databricks CLI by running the command `databricks --version` in your
5 terminal or command prompt.
6 If you have an older version of the Databricks CLI installed, you can upgrade to the latest version by running the
7 command `pip install --upgrade databricks-cli` (for Python 3) or `pip install --upgrade databricks-cli<=1.10.0` (for
8 Python 2).
9 Alternatively, you can download the latest version of the Databricks CLI from the Databricks website and install it
10 manually.
11 Once you have installed the required version of the Databricks CLI, you can use the `databricks assets` command to
12 manage Databricks Asset Bundles.
```

Databricks CLI version 1.10.0 doesn't exist and isn't the correct answer. Furthermore, there is no **databricks assets** command. In both cases, the model hallucinated an answer.



Using RAG to give an LLM access to documentation about DABs

What happens if we set up a RAG system that can search for relevant segments of the Databricks Asset Bundles docs and provide them to the model?

We set up a RAG application by extracting text from documentation about DABs, loading it into a vector database and setting up a retrieval system to pass relevant information to the LLM.

Now when we ask our question, the RAG application queries the vector database and retrieves the most relevant results, which it sends along with our question to the LLM. Let's ask the same questions as before and see how the results differ.

Here are the new results.

```
1 User: What are databricks DABs?
2 Bot: According to the provided context, Databricks Asset Bundles (DABs) are a tool for streamlining the development
3 of complex data, analytics, and machine learning (ML) projects on the Databricks platform. They provide a way to
4 manage complex projects during active development by offering CI/CD capabilities to software development workflows
5 in a single, concise, and declarative YAML syntax. DABs allow users to automate project tests, deployments, and
6 configuration management, which can reduce errors and promote software best practices across an organization through
7 templated projects.
```

```
1 User: What version of the databricks CLI do I need in order to use Databricks Asset Bundles?
2 Bot: Databricks CLI version 0.205 or higher is required for Databricks Asset Bundles. You can confirm the version by
3 running the command 'databricks --version' in your terminal. Note that version 0.205.2 or higher is required. If you
4 haven't installed the Databricks CLI yet, you can install it by following the instructions in the Install or update
5 the Databricks CLI documentation. Additionally, make sure that your remote Databricks workspaces are configured
6 correctly, specifically that the workspace files feature is enabled, which is the case by default for Databricks
7 Runtime version 11.2 or later.
```

In both cases, the model generated correct answers because it was given access to the information it needed.

Without RAG, the best the LLM could do was guess or admit it didn't know. With RAG, the LLM provided the correct answers.

Addressing the Shortcomings of LLMs With RAG

LLMs are capable of giving compelling and coherent answers to user prompts. They're often able to detect nuance, identify context and give the appearance of reasoning when answering. And because they're trained on vast amounts of data, they have access to enormous amounts of knowledge.

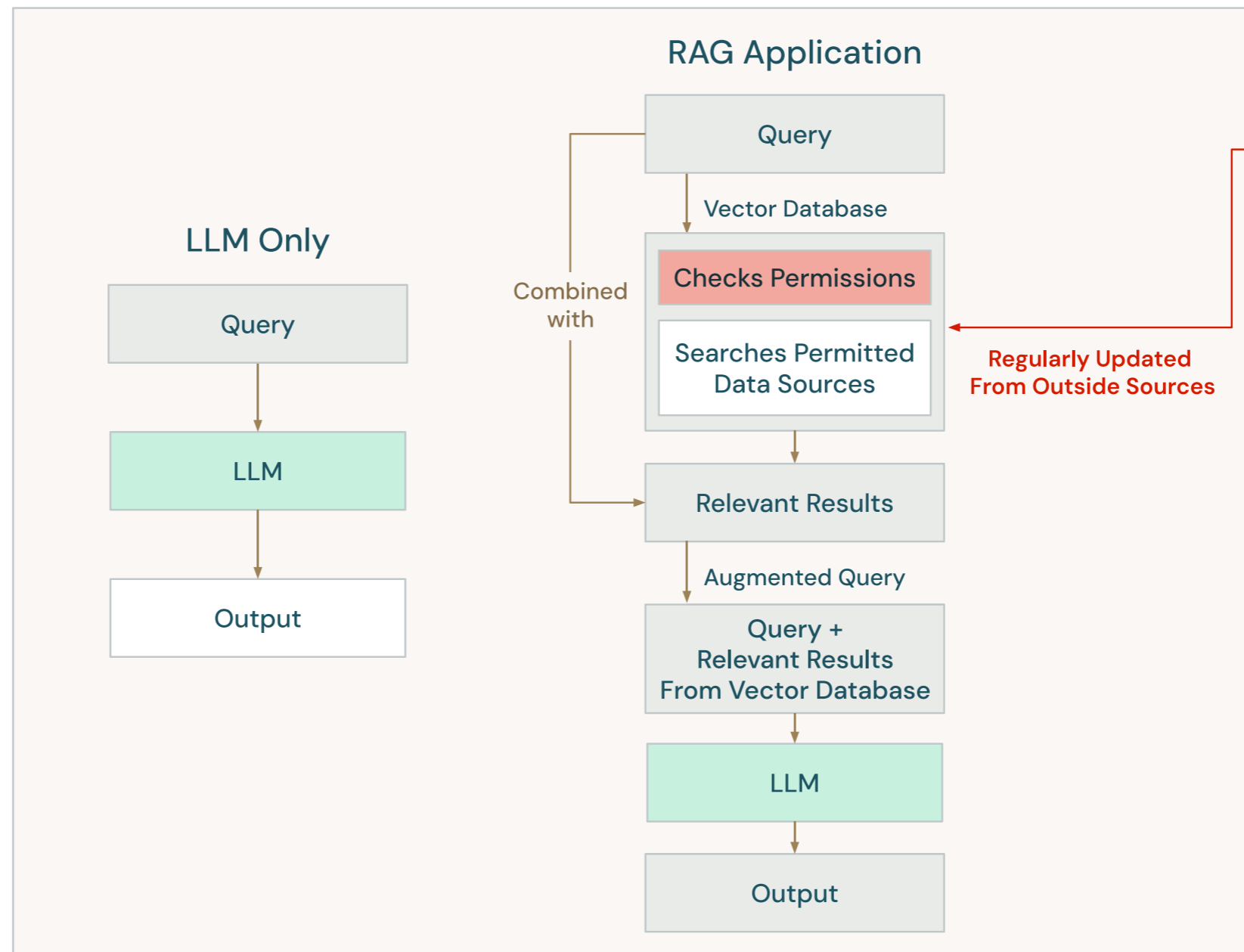
LLMs, however, are not *reliable* as knowledge sources. LLMs often respond with made-up answers, or hallucinations, rather than acknowledging that they don't know the correct answer. Furthermore, LLMs are fundamentally limited by their training data. An LLM alone doesn't know anything that happened after its training is completed, and it lacks access to proprietary information, such as company documents, that weren't publicly available during training.

We can try to mitigate these problems by explicitly giving the LLM the information it needs to address the user's prompt. This can be as simple as copying and pasting a couple of pages of reference documents and sending them, along with a question, to ChatGPT or another LLM. In so doing, we have *augmented* the original prompt (the question) with the information needed to answer the question accurately. The additional step of building a retrieval system, such as a vector database, allows us to automate this process and ensure that the model has the most relevant information without requiring the user to seek it out and add it to the prompt manually.

While implementing RAG with Vector Search involves the extra steps of data processing and managing the generated vectors (often with a vector database), it can help to address the limitations of using LLMs alone. RAG improves on LLM-only approaches by providing additional, specific context that the LLM can use when formulating an answer. RAG also has benefits compared to retrieval-only systems, as the LLM can process texts from multiple sources into a readable output tailored to the user's prompt.

RAG compared to LLM-only approaches

Let's explore some of the benefits of RAG in more detail.



- **RAG applications can incorporate proprietary data.** Most LLMs are trained on publicly available data. They don't have access to a company's internal documents or communications. RAG allows you to supply proprietary or domain-specific information such as internal memos, emails or design documents to the model.
- **RAG applications can access up-to-date information.** LLMs are generally trained at a particular point in time and then released for use. Older models lack updated information about the state of the world or a particular field or business. For example, if a new version of a software product is released after an LLM was trained, it won't be able to provide assistance specific to the new version. RAG provides a way to supply the model with up-to-date information.
- **RAG can enhance the accuracy of LLM responses.** LLMs alone can respond with incorrect or fabricated information (hallucinations). They aren't consistent or reliable information sources. An effective RAG system can retrieve relevant and correct references and supply them to the model, potentially reducing the occurrence of hallucinations. Outputs can include citations of original sources, allowing for human verification.
- **RAG enables fine-grained data access control.** LLMs alone cannot reliably provide different responses to different users based on security or permission considerations. RAG applications, on the other hand, can be designed to retrieve only documents that a user has permission to access. This can enable LLMs to securely reference confidential or personal data based on the access credentials of the system's user.

RAG Use Cases

RAG equips LLMs with context-specific information that LLMs alone either don't possess or may not be able to generate reliably when needed. This enables several different applications that would be difficult or impossible using only LLMs.

Question-answering systems

RAG becomes invaluable in use cases where the aim is to “talk to documents,” such as querying HR policies or accessing real-time financial reports. Using RAG, information can be dynamically retrieved and presented in a conversational manner to an end user. For instance, a large e-commerce company uses Databricks for an internal RAG application, enabling their HR team to query hundreds of employee policy documents.

Customer service

RAG systems can streamline the customer service process by providing support personnel with personalized and more informed responses to customer queries. This can enhance customer experience, reduce response times and increase resolution efficiency. We see this kind of “internal copilot” RAG application across many customers seeking to improve the efficiency and effectiveness of internal workers.

Content generation

In content creation scenarios, RAG can be used to draft communications, like sales emails, by integrating the most recent data and relevant context. This can ensure that customer outreach is both personalized and reflects the latest information. One Databricks customer is leveraging RAG to draft email responses to inbound sales emails, incorporating external product and customer information into responses.

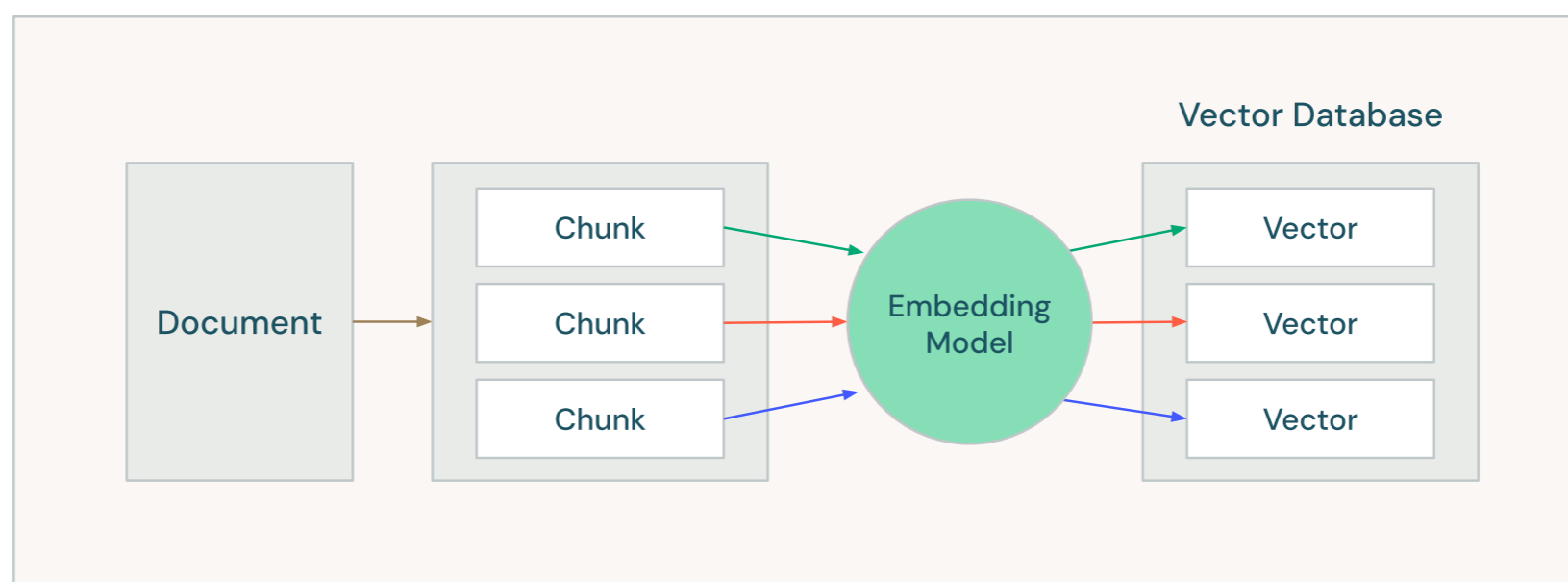
Code assistance

RAG can enhance code completion and code Q&A systems by intelligently searching and retrieving information from code bases, documentation and external libraries. This can result in improved code generation and more relevant responses compared to LLM-only code assistants.

RAG With Vector Search — Step by Step

RAG with Vector Search involves **retrieving** information using a vector database, **augmenting** the user's prompt with that information and **generating** a response based on the user's prompt and information retrieved using an LLM. In this section, we'll review each of these steps, focusing on the processes represented in a reasonably standard RAG system. Be aware that there are many different approaches to these steps, including some advanced techniques that may increase performance but add complexity.

Data preparation: Getting an external information source into a vector database



Before we can perform RAG with Vector Search, we need to get data — in this case, unstructured text data — into a vector database. There are many approaches to doing this, and it's essential to try different methods to determine which is the most effective for your use case.

Data preparation generally isn't a one-time task, because a vector database should be regularly updated to provide up-to-date and high-quality information. This is one of the key benefits of RAG — we can continuously update the vector database without needing to update the LLM weights over time.

A few core steps for preparing data for RAG include the following:

PARSING THE INPUT DOCUMENTS

The raw documents may not be in a format amenable to processing for RAG with Vector Search. Images may need to be converted to text; tables or images might require further processing and there may be extraneous text, such as page headers or page numbers, to clean up or remove. It is often necessary to parse the raw input documents and get them into a format — usually text — that will work with the rest of the RAG pipeline.

SPLITTING DOCUMENTS INTO CHUNKS

You typically don't want to retrieve entire books, web pages or articles in a RAG application. Instead, split the documents into smaller chunks so you can send more specific results to the LLM for context. "Documents" is a general term for referring to source texts, but you can think of documents as any kind of text.

Chunk size can affect the output quality of a RAG application. If the chunks are too small, they may not include enough context to address the user's query. If the chunks are too large, the LLM may fail to pull out the relevant details, focusing instead on other details included in the chunk.

There's no one-size-fits-all solution to choosing the best chunk size. It depends on the source documents, the LLM and the RAG application's goals. It's important to try different chunk sizes.

EMBEDDING THE TEXT CHUNKS

After splitting the source documents into manageable chunks, use an *embedding model* to convert each of those chunks into a high-dimensional numerical vector.

An embedding model is a special kind of language model that uses its knowledge of language to generate a numeric vector or a series of numbers, called an *embedding*, from a text. Embeddings encode the nuanced and context-specific meaning of each text in numeric form. A good embedding model will know that “raining cats and dogs” is a phrase about the weather, not a phrase about pets.

The true power of embeddings for RAG is that they can be mathematically compared to each other. We can measure how “similar” two embeddings are, which in this context, equates to how closely the meanings of their original texts are related. This will be especially useful later in the RAG process when we embed a user’s prompt, compare it to the embedded texts in the vector database and identify those we think are most likely to help the LLM provide a useful answer.

STORING AND INDEXING THE EMBEDDINGS

Embeddings are stored in a specialized kind of database known as a vector database, which is designed to efficiently store and search for vector data like embeddings. A vector database is a type of vector store — these terms are often used interchangeably — though “vector store” can refer to any type of vector storage solution, not just to databases. Vector databases often incorporate update mechanisms so newly added chunks can be searched and retrieved immediately. While such databases are not strictly necessary for RAG or Vector Search, they often meaningfully improve RAG performance and reliability.

Having a huge number of text chunks can result in slower retrieval speeds. A common approach to maintain performance is to index the embeddings with a vector index. A vector index is a mechanism, often part of a vector database, that uses various algorithms to organize and map vector embeddings in a way that optimizes search efficiency.

RECORDING METADATA

Capturing metadata along with text chunks allows us to filter results based on metadata (if applicable) and provide detailed references along with the results. A RAG application with metadata could, for instance, provide specific URLs or page numbers for the sources retrieved, or it could allow users to explicitly filter by date or source.

EXAMPLE: PREPROCESSING AND EMBEDDING

In the example above, we started with two documents about Databricks Asset Bundles. To use them for RAG, we:

1. Split each document into chunks. For example, one of those chunks is:

```
1 Databricks Assets Bundles are an infrastructure-as-code (IaC) approach to managing your Databricks projects. Use them
2 when you want to manage complex projects where multiple contributors and automation are essential, and continuous
3 integration and deployment (CI/CD) are a requirement. Since bundles are defined and managed through YAML templates
4 and files you create and maintain alongside source code, they map well to scenarios where IaC is an appropriate
5 approach.\n\nSome ideal scenarios for bundles include:\n\nDevelop data, analytics, and ML projects in a team-
6 based environment. Bundles can help you organize and manage various source files efficiently. This ensures smooth
7 collaboration and streamlined processes.\n\nIterate on ML problems faster. Manage ML pipeline resources (such as
8 training and batch inference jobs) by using ML projects that follow production best practices from the beginning.'
```

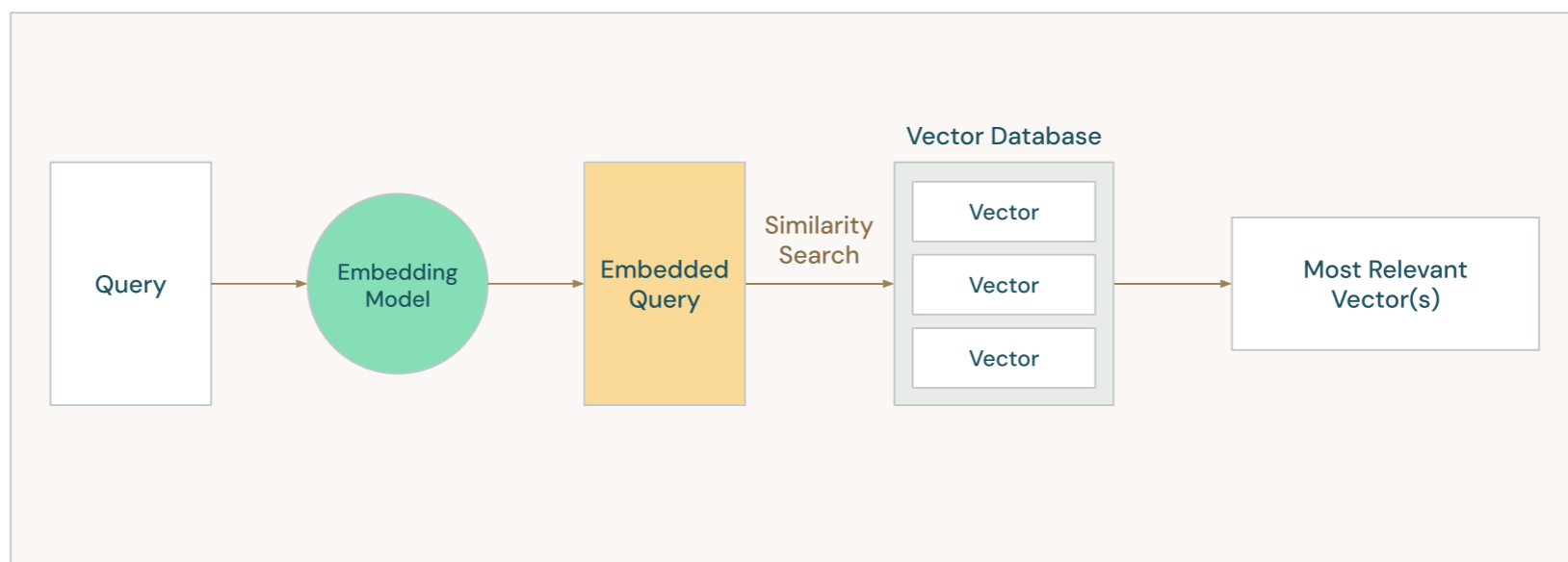
2. Embed the chunks. We use a general-purpose embedding model called `bge-large-en` to turn each chunk into a 1024-dimension numeric vector, which is basically a list of 1024 numbers. The chunk above is translated to:

```
1 [0.0209503173828125, 0.0172576904296875, -0.003314971923828125, -0.0025310516357421875, 0.00670623779296875,
2 -0.00506591796875, 0.0005450248718261719, -0.049896240234375, 0.00630950927734375, 0.0003032684326171875,
3 -0.001049041748046875, -0.0084991455078125, 0.031585693359375, -0.0621337890625, -0.009765625, 0.017669677734375,
4 -0.045623779296875, 0.0022907257080078125, -0.0736083984375, 0.0286102294921875, 0.01532745361328125, 0.0298919677734375,
5 -0.09027099609375, -0.0207977294921875, -0.048736572265625, 0.07818603515625, 0.0648193359375,
6 ...
7 -0.01076507568359375, 0.0123443603515625, -0.016693115234375, 0.0243377685546875, 0.0244903564453125]
```

3. Once both documents are split into chunks and embedded, we use **Databricks Vector Search** to store and index the embeddings. We also record some metadata — the document title and the document date — along with each text chunk and embedding. We'll come back to these vectors and show how they are used to retrieve relevant information in the next section, which focuses on retrieval.

At this phase, the data has been preprocessed and can now be queried. The next step is to retrieve the relevant information from the vector database.

Retrieval: Getting relevant context



After preprocessing our original documents, we have a vector database storing the text chunks, embeddings and metadata. With this in place, we can get to the first step in RAG: **retrieval**. In the retrieval step, the user provides a prompt, often a question, to the RAG application. The RAG application uses the prompt to query the database and identify the most relevant results, which can be used to **augment** the original prompt (the next step).

QUERYING THE VECTOR DATABASE

We can't directly match a user's input, which is usually plain text, with the records in our vector database. So, first we need to use the same embedding model that was used to embed the original text chunks to also embed the user's query. Once we have the embedded query, we can search the vector database to find the most similar records.

If the database contains only a small number of records, searching might involve calculating a **similarity score** for each record. For larger databases, we use vector indexes and specialized search algorithms — many of which use approximations to improve efficiency — to speed up the process.

Once the vector database has identified the most relevant results, the texts of those results can be combined with the user's prompt and sent to the LLM to generate the final response. Note that the embeddings are not "translated back" to text. Instead, the text chunks are stored with the embeddings or linked to them via database keys, so the chunks can simply be retrieved and sent to the next step in the RAG process.

We should decide how many results our RAG system should retrieve. This, like chunk size, is worth testing and can have a significant impact on the quality of the results. Retrieving too few records may mean missing some relevant information, while too many results may dilute the relevant information and make it more likely for the LLM to give irrelevant answers.

IMPROVING RETRIEVAL

The approach described above is often quite effective, but there are many more advanced techniques for improving retrieval, including:

- **Hybrid search:** This method blends traditional keyword search with Vector Search, which can improve retrieval accuracy
- **Reranking:** An additional model can be used to reorder the records initially returned by the similarity search, ensuring the most relevant results are prioritized
- **Summarized text comparison:** Some RAG applications don't compare the user's prompt directly to raw text embeddings. Instead, they use embeddings of summarized texts for a more efficient matching process.
- **Contextual chunk retrieval:** It's often beneficial to include chunks adjacent to the most relevant ones (e.g., the paragraphs preceding and following a retrieved chunk). This approach provides more complete context, which might aid the LLM in generating a useful response.
- **Prompt refinement:** Some RAG applications employ a language model to refine the user's original prompt, crafting a new query that better captures the user's intent for more effective searching in the vector database
- **Domain-specific tuning:** Utilizing embedding models that are fine-tuned for specific tasks or domains can enhance the accuracy and relevance of the retrieved information

These approaches are worth trying if testing reveals that the retrieval component of a RAG application is often failing to return the most relevant records from the vector database.

EXAMPLE: RETRIEVING THE INFORMATION

All the information we need about Databricks Asset Bundles is now available in the vector database. In the retrieval phase, we have to get that information out. To do so, we:

1. Embed the prompt. We use the same embedding model we used to embed the original document chunks to embed the query, and once again the result is a 1024-dimensional vector. So if we start with the prompt “What are Databricks Asset Bundles?” we end up with the embedding:

```
1 [0.006649017333984375, 0.029144287109375, 0.0001398324966430664, 0.00481414794921875, -0.006526947021484375,  
2 -0.00818634033203125, 0.029571533203125, -0.031982421875, 0.01082611083984375, 0.0025653839111328125, -0.013031005859375,  
3 0.01155853271484375, 0.054931640625, -0.049224853515625, -8.767843246459961e-05, 0.03131103515625, -0.022613525390625,  
4 0.0148162841796875, -0.052520751953125, 0.003780364990234375, 0.0279998779296875, 0.018585205078125, -0.081787109375,  
5 -0.030731201171875, -0.0236053466796875, 0.0357666015625, 0.03387451171875, 0.0335693359375,  
6 ...,  
7 0.0260467529296875, -0.0013675689697265625, 0.032318115234375, -0.002666473388671875, 0.0269012451171875, 0.0616455078125]
```

2. Use the embedding to search the vector database. We use the built-in `similarity_search` method of Databricks Vector Search to query the vector database with the embedded prompt. We specify that we want it to return the stored text and that we want the two most relevant results. From this, the database returns:

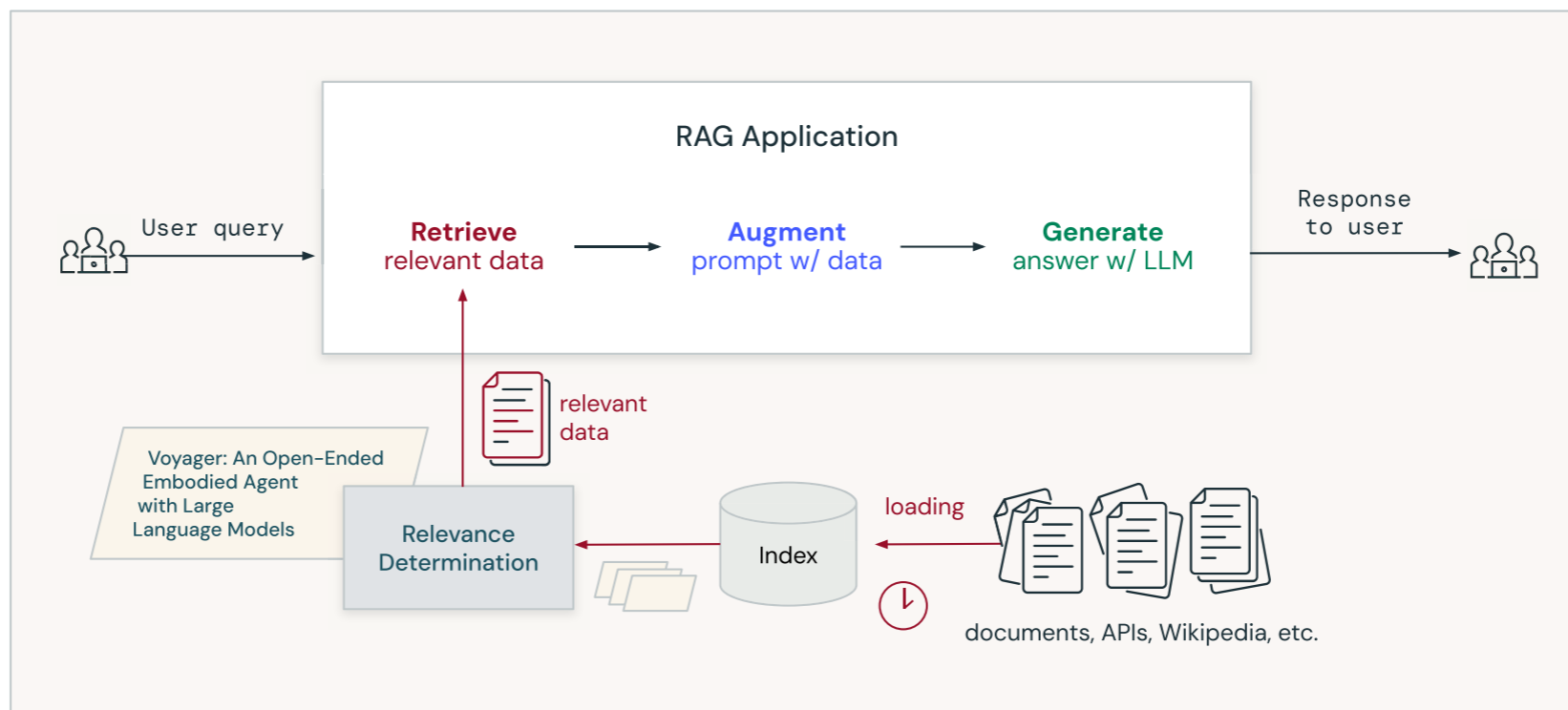
```

1      ['What are Databricks Asset Bundles?\nJanuary 08, 2024\n\nIn this article you will learn the basics of using
2      Databricks Asset Bundles, a new tool for streamlining the development of complex data, analytics, and ML projects for
3      the Databricks platform. Bundles make it easy to manage complex projects during active development by providing CI/CD
4      capabilities to your software development workflow in a single concise and declarative YAML syntax. By using bundles
5      to automate your project's tests, deployments, and configuration management you can reduce errors while promoting
6      software best practices across your organization as templated projects.\n\nPreview\n\nThis feature is in Public
7      Preview.\n\nBundles provide a way to include metadata alongside your project's source files to specify information
8      including:\n\nRequired cloud infrastructure and workspace configurations.\n\nUnit and integration tests.',

9      'Databricks Assets Bundles are an infrastructure-as-code (IaC) approach to managing your Databricks projects.
10     Use them when you want to manage complex projects where multiple contributors and automation are essential, and
11     continuous integration and deployment (CI/CD) are a requirement. Since bundles are defined and managed through
12     YAML templates and files you create and maintain alongside source code, they map well to scenarios where IaC is an
13     appropriate approach.\n\nSome ideal scenarios for bundles include:\n\nDevelop data, analytics, and ML projects in
14     a team-based environment. Bundles can help you organize and manage various source files efficiently. This ensures
15     smooth collaboration and streamlined processes.\n\nIterate on ML problems faster. Manage ML pipeline resources (such
16     as training and batch inference jobs) by using ML projects that follow production best practices from the beginning.']
    
```

Now what do we do with this information? This is where the *augmentation* part of RAG comes in.

Augmentation: Adding context to the user's prompt



The *retrieval* component of a RAG system queries the vector database with the user's prompt. It returns a selection of relevant texts and, in some cases, metadata. The texts are used to augment the original prompt.

AUGMENTING THE PROMPT WITH THE RETRIEVED CONTEXT

In its simplest form, "augmentation" means combining the user's original prompt with the retrieved texts. This equips the model with both the prompt and the context needed to address the prompt. In practice, the structure of the new prompt that combines the retrieved texts and the user's prompt can impact the quality of the final result.

For example, the final prompt usually includes an instruction on how to use the context:

```
1   ...
2   Based on the following context, answer the user's question. Context:
3   {context}
4   User question:
5   {user question}
6   ...
```

Depending on the model, putting the context first might be more or less effective than putting the user's question first. The phrasing might also be consequential. For example, you might want to phrase the instruction to emphasize that the model should generate its answer using *only* the retrieved context.

CONTEXT WINDOW: HOW MUCH CONTEXT SHOULD WE PROVIDE?

LLMs are limited by a “context window,” or the amount of text they can process to generate a response. Designing a RAG system involves ensuring that all retrieved texts and the user’s prompt fit within this window. Overloading the system with too many texts might lead to errors or lost context.

Some LLMs boast longer context windows, capable of handling texts as lengthy as short books. But this doesn’t mean adding more texts to the user’s prompt will always be beneficial. LLMs sometimes struggle to pay equal attention to all parts of a lengthy context. They typically focus more effectively on the beginning and end, potentially overlooking the middle content. This is known as the “lost in the middle” phenomenon. Hence, even with longer context windows, careful selection and arrangement of texts are crucial for augmenting prompts effectively.

EXAMPLE: PROMPT AUGMENTATION

Our question for the RAG application was “What are Databricks Asset Bundles?” and we retrieved the following two chunks for context:

```
1 ['What are Databricks Asset Bundles?\nJanuary 08, 2024\n\nIn this article you will learn the basics of using\n2 Databricks Asset Bundles, a new tool for streamlining the development of complex data, analytics, and ML projects for\n3 the Databricks platform. Bundles make it easy to manage complex projects during active development by providing CI/CD\n4 capabilities to your software development workflow in a single concise and declarative YAML syntax. By using bundles\n5 to automate your project's tests, deployments, and configuration management you can reduce errors while promoting\n6 software best practices across your organization as templated projects.\n\nPreview\n\nThis feature is in Public\n7 Preview.\n\nBundles provide a way to include metadata alongside your project's source files to specify information\n8 including:\n\nRequired cloud infrastructure and workspace configurations.\n\nUnit and integration tests.'],\n\n9 'Databricks Assets Bundles are an infrastructure-as-code (IaC) approach to managing your Databricks projects.\n10 Use them when you want to manage complex projects where multiple contributors and automation are essential, and\n11 continuous integration and deployment (CI/CD) are a requirement. Since bundles are defined and managed through\n12 YAML templates and files you create and maintain alongside source code, they map well to scenarios where IaC is an\n13 appropriate approach.\n\nSome ideal scenarios for bundles include:\n\nDevelop data, analytics, and ML projects in\n14 a team-based environment. Bundles can help you organize and manage various source files efficiently. This ensures\n15 smooth collaboration and streamlined processes.\n\nIterate on ML problems faster. Manage ML pipeline resources (such\n16 as training and batch inference jobs) by using ML projects that follow production best practices from the beginning.']
```

We use these and some more general instructions to construct the final prompt we send to the model:

```
1 You are a helpful assistant. Answer the user's question. If context is provided, you must answer based only on the
2 context. If no context is provided, answer based on your knowledge. If you don't know the answer, say you don't know.
3 Be concise.

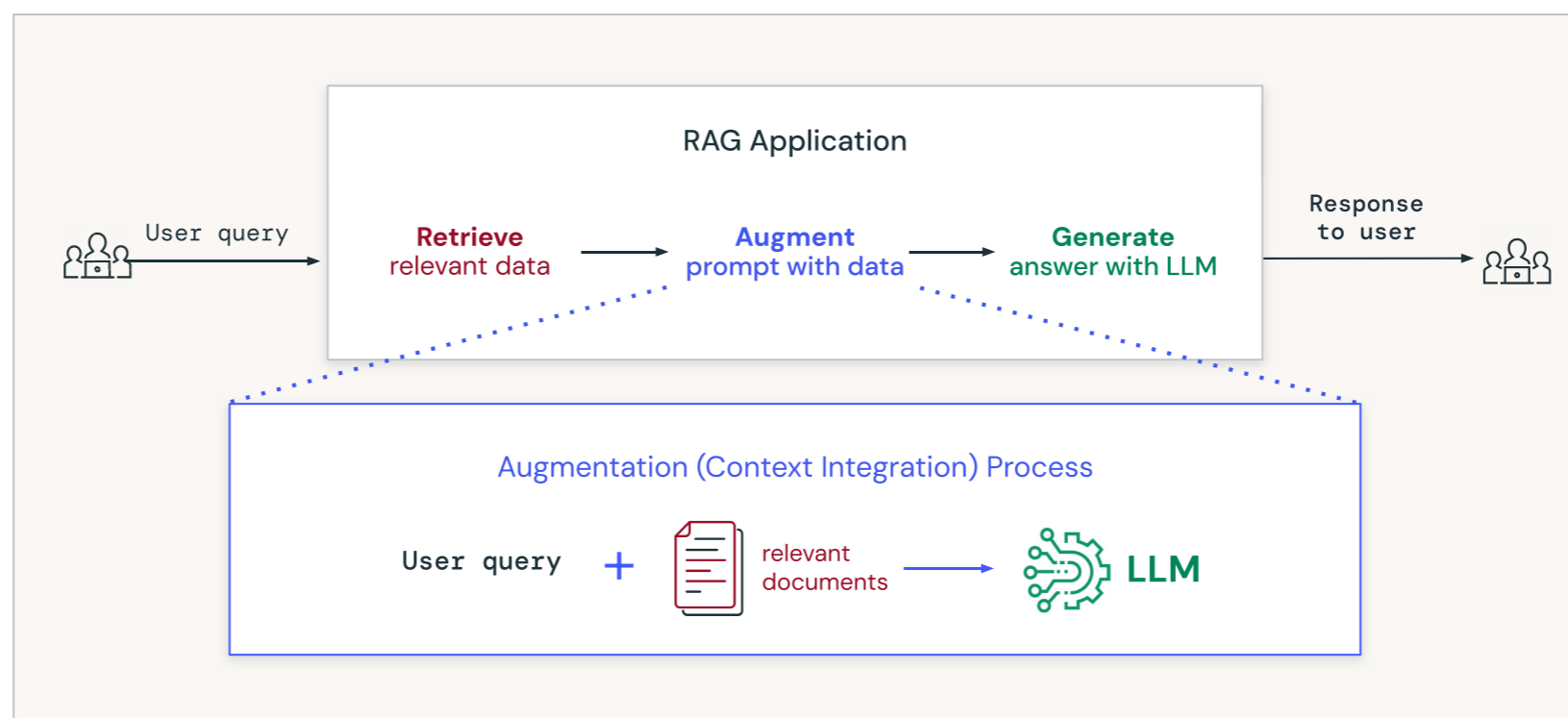
4 Answer the question based on the provided context. Context:

5 What are Databricks Asset Bundles?
6 January 08, 2024
7 In this article you will learn the basics of using Databricks
8 Asset Bundles, a new tool for streamlining the development of complex data, analytics, and ML projects for the
9 Databricks platform. Bundles make it easy to manage complex projects during active development by providing CI/CD
10 capabilities to your software development workflow in a single concise and declarative YAML syntax. By using bundles
11 to automate your project's tests, deployments, and configuration management you can reduce errors while promoting
12 software best practices across your organization as templated projects.
13 Preview
14 This feature is in Public
15 Preview.
16 Bundles provide a way to include metadata alongside your project's source files to specify information
17 including:
18 Required cloud infrastructure and workspace configurations.
19 Unit and integration tests.

20 Databricks Assets Bundles are an infrastructure-as-code (IaC) approach to managing your Databricks projects. Use them
21 when you want to manage complex projects where multiple contributors and automation are essential, and continuous
22 integration and deployment (CI/CD) are a requirement. Since bundles are defined and managed through YAML templates
23 and files you create and maintain alongside source code, they map well to scenarios where IaC is an appropriate
24 approach.
25 Some ideal scenarios for bundles include:
26 Develop data, analytics, and ML projects in a team-
27 based environment. Bundles can help you organize and manage various source files efficiently. This ensures smooth
28 collaboration and streamlined processes.
29 Iterate on ML problems faster. Manage ML pipeline resources (such as
30 training and batch inference jobs) by using ML projects that follow production best practices from the beginning.
```

We've now *augmented* the original prompt with the context needed to address it and with instructions on how to use the context. The last step is to pass this along to the LLM.

Generation: Producing useful output with an LLM



After the *retrieval* and *augmentation* steps, we have a prompt and a set of retrieved texts, formatted with instructions on how to use the texts to answer the prompt. In the *generation* step of RAG, we send the augmented prompt to an LLM, and the LLM responds with an answer.

At this stage, many approaches can be used to customize the final output and adjust the end user's experience with the RAG system.

PROMPT ENGINEERING

As noted in the previous section, we can include instructions on how the model should use the retrieved context (e.g., “answer only based on the provided context”). We can provide similar instructions to the LLM to guide its output tone and structure. We might, for example, specify that the answers should be very polite or should be phrased in a short and direct manner. We could also instruct the model to refuse to answer questions about a competitor’s products.

PREPROCESSING AND POSTPROCESSING

Some LLMs are quite good at following instructions, but none are completely reliable. Some programmatic preprocessing or postprocessing can be used to structure the output in a certain way or to guarantee that certain types of responses are or are not generated. For example, we could append a list of links to the most relevant context sources at the end of each response, or we could automatically respond with a polite refusal to answer if specific words or phrases are included in the user’s prompt.

CONVERSATION HISTORY

Another important consideration is whether the RAG system should have a memory. In other words, can it engage in multi-turn conversations in which it remembers earlier questions and responses? A conversational AI system can give users the opportunity to refine their questions and ask follow-ups, but it adds some complexity. Different models and frameworks use different approaches to storing conversation histories, and including conversation histories may require more careful management of context length.



EXAMPLE: RAG OUTPUT

After we generate our augmented prompt, it is sent to the LLM. This is the response:

```
...
1 User: What are Databricks Asset Bundles?

2 Bot: Databricks Asset Bundles are a way to manage complex data, analytics, and machine learning projects on the
3 Databricks platform using an infrastructure-as-code (IaC) approach. They provide a way to include metadata alongside
4 project source files to specify information such as required cloud infrastructure and workspace configurations, unit
5 and integration tests, and CI/CD capabilities. Bundles are defined and managed through YAML templates and files,
6 making it easier to manage complex projects with multiple contributors and automation, and promote software best
7 practices across an organization. They are particularly useful in scenarios such as team-based development, iterating
8 on machine learning problems, and managing ML pipeline resources.
```

One way to enhance the generation step of RAG is to add multi-turn conversation ability. This allows us to ask a follow-up question that refers back to the original question and answer:

```
...
1 User: What version of the CLI do I need in order to use them?

2 Bot: According to the provided context, you need Databricks CLI version 0.205.2 or higher to use Databricks Asset
3 Bundles. You can check the version of the Databricks CLI you are currently using by running the command `databricks
4 --version` in your terminal. If you don't have the required version, you can install or update the Databricks CLI
5 following the instructions provided in the article.
```

Evaluation: Measuring RAG performance

Because a RAG application has many adjustable variables that may affect the retrieval or generation quality, it's important to have ways to measure its performance.

RAG evaluation is an area of active research and experimentation. It's one of the most challenging parts of setting up a RAG application, and there's no universal solution. As we've discussed, RAG comprises multiple steps, and it's often helpful to evaluate these steps separately. A great LLM won't be able to compensate for a poor retrieval pipeline, and the best retrieval system can't overcome the limitations of a weak model.

At its core, RAG evaluations involve generating prompts, identifying the relevant records that should be retrieved to address each prompt and generating good answers to those questions. Running the evaluations means passing each evaluation prompt to the RAG application and comparing the desired retrievals and responses to the actual retrievals and responses.

RAG evaluations often rely on other LLMs to judge response quality. For example, RAG responses are often evaluated on their "faithfulness" to the provided context. A judge LLM examines the context and the end response from a RAG application and provides a rating on how true the response is to the context.



Utilizing RAG With Other Modeling and Model Customization Methods

At this point, you should have a good idea of what RAG is and how it compares to using an LLM alone. However, RAG isn't the only approach to customizing LLMs or equipping them with new information. In this section, we explain how RAG fits into the broader context of LLM customization approaches.

All of the following approaches involve tradeoffs between cost, complexity and expressive power. "Cost" refers simply to the financial cost of setting up and using a given model or system. "Complexity" means the intricacy or technical difficulty, which may be reflected in the time, effort and expertise required. And "expressiveness" refers to the model's or system's ability to generate diverse, meaningful and useful responses tailored to your specific needs.

These methods are not mutually exclusive and should be used in combination to maximize task- or domain-specific performance.

Prompt engineering

Prompt engineering is the process of designing prompts or prompt templates that guide a model's outputs toward a desired result. It's typically the least complex approach and entails the lowest up-front costs because it doesn't involve changing the model's weights or working with any external data systems.

The cost associated with prompt engineering will vary. Large and highly capable models are often required in order to understand and follow complex prompts. These models often entail higher serving costs or per-token costs than smaller, less-capable models. That said, prompt engineering doesn't come with the high up-front costs of training a model or setting up the infrastructure for a production RAG system.

The expressiveness obtainable via prompt engineering is fundamentally limited by the underlying model. Prompt engineering offers a good alternative to RAG in cases where there's no need for proprietary or recent knowledge.

Fine-tuning

Fine-tuning is the process of adapting a pretrained generative model to a new domain or task by training all or some of the model's weights (or, in the case of adapter methods, new weights) on new data. The primary goal of fine-tuning is to enhance the model's expressiveness and accuracy in handling domain-specific queries or tasks. For example, a language model might be fine-tuned to follow instructions based on a large dataset of instruction and response data. Or a model could be fine-tuned on a collection of medical texts in order to better understand medical jargon.

The cost and complexity of this process can vary greatly, depending on factors such as the size of the model, the quantity and specificity of the training data and the nature of the task. Fine-tuning can sometimes be used to reduce costs. A smaller model fine-tuned on a specific task can replace a larger and more expensive generalist model.

FINE-TUNING AND RAG

While RAG excels in enhancing a model's responses with additional, relevant information, it doesn't fundamentally change the model's behavior or linguistic style. Any limitations or quirks of the base model will still be present in a RAG system, while fine-tuning can durably change the model's behavior in ways that are less constrained by the base model's behavior.

On the other hand, fine-tuning doesn't include a straightforward mechanism for rapidly updating the model with new information, and it may not be as reliable as RAG for generating relevant responses, even if the model was fine-tuned on relevant data.

Pretraining

Pretraining is the process of training an LLM from scratch. This is the highest-cost and highest-complexity approach, but it offers the greatest potential control over the model's expressiveness.

Pretraining a model gives you control over all the data that goes into it. This might mean including proprietary data not available to off-the-shelf models or excluding data from sources not deemed trustworthy, reliable or legally acceptable in a given business context. For example, you may decide not to include Reddit data when pretraining a model to give legal or financial advice.

Consider pretraining when it is essential to understand and control all the data a model is trained on or when you need a domain-specific model that meets certain evaluation or performance requirements not available in existing models.

RAG

RAG is more complex than prompt engineering alone. It requires setting up a retrieval system (i.e., a vector database) and integrating the retrieved context with the prompt.

If access to external information is the goal, RAG offers many benefits:

- Ability to add and remove data sources without changing the model
- Control over who has permission to access certain data sources
- Flexibility to compare different LLMs without needing to train them on new data

The cost and complexity of a RAG system depends on the choice of model and on the scale and structure of the retrieval system. As with any database, a vector database that guarantees low-latency retrieval over a vast number of records will be more costly than a higher-latency, smaller-scale system. But a RAG system alone will not entail the up-front cost and complexity of pretraining or fine-tuning a model.

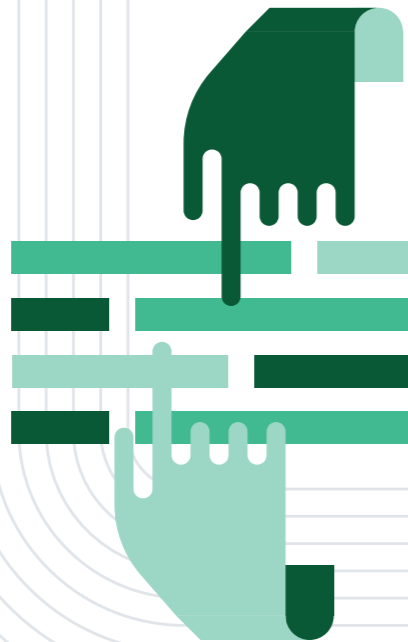
The expressiveness of a RAG system, though limited by the choice of LLM, can still be quite high given its access to contextually relevant external data sources.

Combinations of methods

These methods can and often should be used together. RAG and prompt engineering are already inseparable — merging the user's prompt with the external data sources is a form of prompt engineering.

Using a custom pretrained or fine-tuned model in a RAG system can improve the RAG system by offering fine-grained control over the model's data, the response tone and structure and the aptitude with domain-specific language.

In general, it's good practice to start with less costly, less complex methods and evaluate their performance. Moving to more complex methods or combinations of methods is a good option when the simpler methods prove inadequate for their tasks.



RAG on Databricks

While it's possible to set up a quick RAG demo on your laptop in a few minutes, a production-ready RAG system requires careful orchestration of several different components in a reliable, scalable and secure manner. Databricks offers an end-to-end RAG solution combining data management and governance with a vector database, model serving and other tools for managing and monitoring AI processes.

Lakehouse architecture

RAG applications in Databricks are built on **lakehouse architecture**. The lakehouse centralizes the management of structured data, unstructured data and AI assets under a common governance scheme, Unity Catalog. Organizations can build cloud-agnostic RAG systems on proprietary data with sophisticated security, lineage tracking and monitoring.

Vector Search

Databricks Vector Search enables you to create an auto-updating vector database from Delta tables, managed via Databricks Unity Catalog and searchable using a simple API. Databricks Vector Search scales automatically to handle different numbers of documents and queries.

Model serving

Databricks Model Serving provides a number of different ways to host and use LLMs and embedding models for RAG. Databricks Model Serving supports custom models via MLflow; governance of external models such as SaaS models from OpenAI, Anthropic or Google; and state-of-the-art open source models from the **Databricks Foundation Model APIs**, which offer both pay-per-token and provisioned throughput offerings.

MLflow

MLflow is an open source end-to-end model lifecycle management platform. It includes a variety of tools useful for implementing and improving RAG systems, including an evaluation framework and a prompt engineering UI.

Lakehouse Monitoring

Databricks Lakehouse Monitoring provides a centralized monitoring solution for both model and data monitoring. It allows you to keep track of various statistical properties for all your data sources as well as to monitor the performance of your served models.



Summary

Whether you're looking to disrupt traditional industries, enhance creative endeavors or solve complex problems in novel ways, the potential applications of generative AI are limited only by your imagination and willingness to experiment. Remember, every significant advancement in this field began with a simple idea and the courage to explore it further.

For those seeking more knowledge or who are simply curious about the latest developments in the realm of generative AI, we've provided some resources on training, demos and product information.

GenAI training

- **Generative AI engineer learning pathway:** Take self-paced, on-demand and instructor-led courses on generative AI
- **Free LLM course (edX):** An in-depth course to learn GenAI and LLMs inside and out
- **GenAI webinar:** Learn how to take control of your GenAI app performance, privacy and cost, and drive value with generative AI

Additional resources

- **The Big Book of MLOps:** A deep dive into the architectures and technologies behind MLOps — including LLMs and GenAI
- **Mosaic AI:** Product page covering the features of Mosaic AI within Databricks
- **The Big Book of Generative AI:** Best practices for building production-quality GenAI applications

Build Production-Quality GenAI Applications — See How

Create high-quality generative AI applications and ensure your output is accurate, governed and safe. See why over 10,000 organizations worldwide rely on Databricks for all their workloads from BI to AI — test-drive the full Databricks Platform free for 14 days.

Try Databricks free

Take Generative AI Fundamentals On-Demand Training

About Databricks

Databricks is the data and AI company. More than 10,000 organizations worldwide — including Comcast, Condé Nast, Grammarly and over 50% of the Fortune 500 — rely on the Databricks Data Intelligence Platform to unify and democratize data, analytics and AI. Databricks is headquartered in San Francisco, with offices around the globe, and was founded by the original creators of Lakehouse, Apache Spark™, Delta Lake and MLflow. To learn more, follow Databricks on [LinkedIn](#), [X](#) and [Facebook](#).

