AWS Machine Learning Blog

Amazon Bedrock Knowledge Bases now supports advanced parsing, chunking, and query reformulation giving greater control of accuracy in RAG based applications

by Sandeep Singh, Chris Pecora, and Mani Khanuja | on 10 JUL 2024 | in Amazon Bedrock, Announcements, Artificial Intelligence, Generative AI | Permalink | Comments | Share

Amazon Bedrock Knowledge Bases is a fully managed service that helps you implement the entire Retrieval Augmented Generation (RAG) workflow from ingestion to retrieval and prompt augmentation without having to build custom integrations to data sources and manage data flows, pushing the boundaries for what you can do in your RAG workflows.

However, it's important to note that in RAG-based applications, when dealing with large or complex input text documents, such as PDFs or .txt files, querying the indexes might yield subpar results. For example, a document might have complex semantic relationships in its sections or tables that require more advanced chunking techniques to accurately represent this relationship, otherwise the retrieved chunks might not address the user query. To address these performance issues, several factors can be controlled. In this blog post, we will discuss new features in Amazon Bedrock Knowledge Bases can improve the accuracy of responses in applications that use RAG. These include advanced data chunking options, query decomposition, and CSV and PDF parsing improvements. These features empower you to further improve the accuracy of your RAG workflows with greater control and precision. In the next section, let's go over each of the features including their benefits.

Features for improving accuracy of RAG based applications

In this section we will go through the new features provided by Amazon Bedrock Knowledge Bases to improve the accuracy of generated responses to user query.

Advanced parsing

Advanced parsing is the process of analyzing and extracting meaningful information from unstructured or semi-structured documents. It involves breaking down the document into its constituent parts, such as text, tables, images, and metadata, and identifying the relationships between these elements.

Parsing documents is important for RAG applications because it enables the system to understand the structure and context of the information contained within the documents.

There are several techniques to parse or extract data from different document formats, one of which is using foundation models (FMs) to parse the data within the documents. It's most helpful when you have complex data within documents such as nested tables, text within images, graphical representations of text and so on, which hold important information.

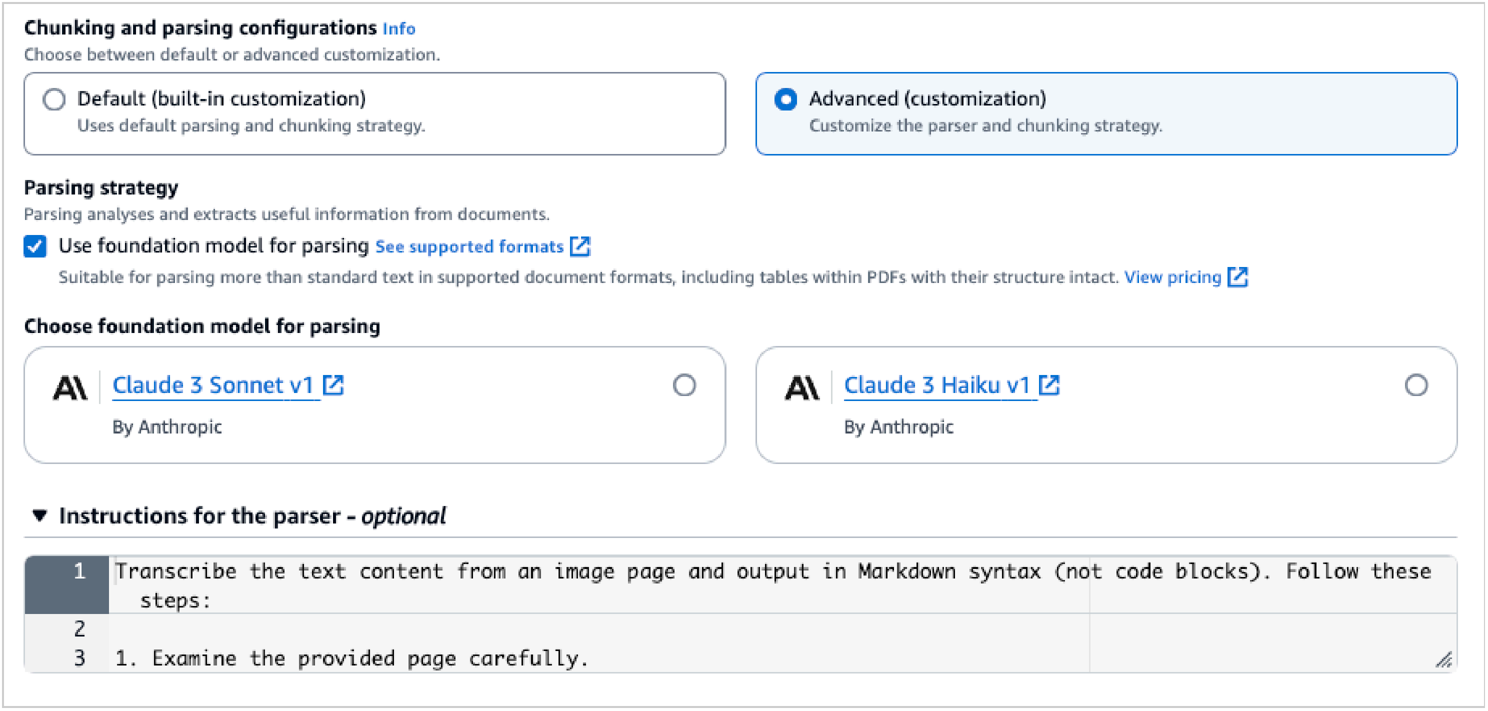
Using the advanced parsing option offers several benefits:

- **Improved accuracy**: FMs can better understand the context and meaning of the text, leading to more accurate information extraction and generation.
- **Adaptability**: Prompts for these parsers can be optimized on domain-specific data, enabling them to adapt to different industries or use cases.
- **Extracting entities**: It can be customized to extract entities based on your domain and use case.
- **Complex document elements**: It can understand and extract information represented in graphical or tabular format.

Parsing documents using FMs are particularly useful in scenarios where the documents to be parsed are complex, unstructured, or contain domain-specific terminology. It can handle ambiguities, interpret implicit information, and extract relevant details using their ability to understand semantic relationships, which is essential for generating accurate and relevant responses in RAG applications. These parsers might incur additional fees, see the pricing details before using this parser selection.

In Amazon Bedrock Knowledge Bases, we provide our customers the option to use FMs for parsing complex documents such as .pdf files with nested tables or text within images.

From the AWS Management Console for Amazon Bedrock, you can start creating a knowledge base by choosing **Create knowledge base**. In **Step 2: Configure data source**, select **Advanced (customization)** under **Chunking & parsing configuration**s, as shown in the following image. You can select one of the two models (Anthropic Claude 3 Sonnet or Haiku) currently available for parsing the documents.



Chunking and parsing configurations Info
Choose between default or advanced customization.

- ○ Default (built-in customization) — Uses default parsing and chunking strategy.
- ● Advanced (customization) — Customize the parser and chunking strategy.

Parsing strategy
Parsing analyses and extracts useful information from documents.

✓ Use foundation model for parsing See supported formats
Suitable for parsing more than standard text in supported document formats, including tables within PDFs with their structure intact. View pricing

Choose foundation model for parsing

- ○ Claude 3 Sonnet v1 — By Anthropic
- ○ Claude 3 Haiku v1 — By Anthropic

▼ **Instructions for the parser - *optional***

```
Transcribe the text content from an image page and output in Markdown syntax (not code blocks). Follow these
  steps:

1. Examine the provided page carefully.
```

If you want to customize the way the FM will parse your documents, you can optionally provide instructions based on your document structure, domain, or use case.

Based on your configuration, the ingestion process will parse and chunk documents, enhancing the overall response accuracy. We will now explore advanced data chunking options, namely semantic and hierarchical chunking which splits the documents into smaller units, organizes and store chunks in a vector store, which can improve the quality of chunks during retrieval.

Advanced data chunking options

The objective shouldn't be to chunk data merely for the sake of chunking, but rather to transform it into a format that facilitates anticipated tasks and enables efficient retrieval for future value extraction. Instead of inquiring, "How should I chunk my data?", the more pertinent question should be, "What is the most optimal approach to use to transform the data into a form the FM can use to accomplish the designated task?"[1]

To achieve this goal, we introduced two new data chunking options within Amazon Bedrock Knowledge Bases in addition to the fixed chunking, no chunking, and default chunking options:

- **Semantic chunking**: Segments your data based on its semantic meaning, helping to ensure that the related information stays together in logical chunks. By preserving contextual relationships, your RAG model can retrieve more relevant and coherent results.
- **Hierarchical chunking**: Organizes your data into a hierarchical structure, allowing for more granular and efficient retrieval based on the inherent relationships within your data.

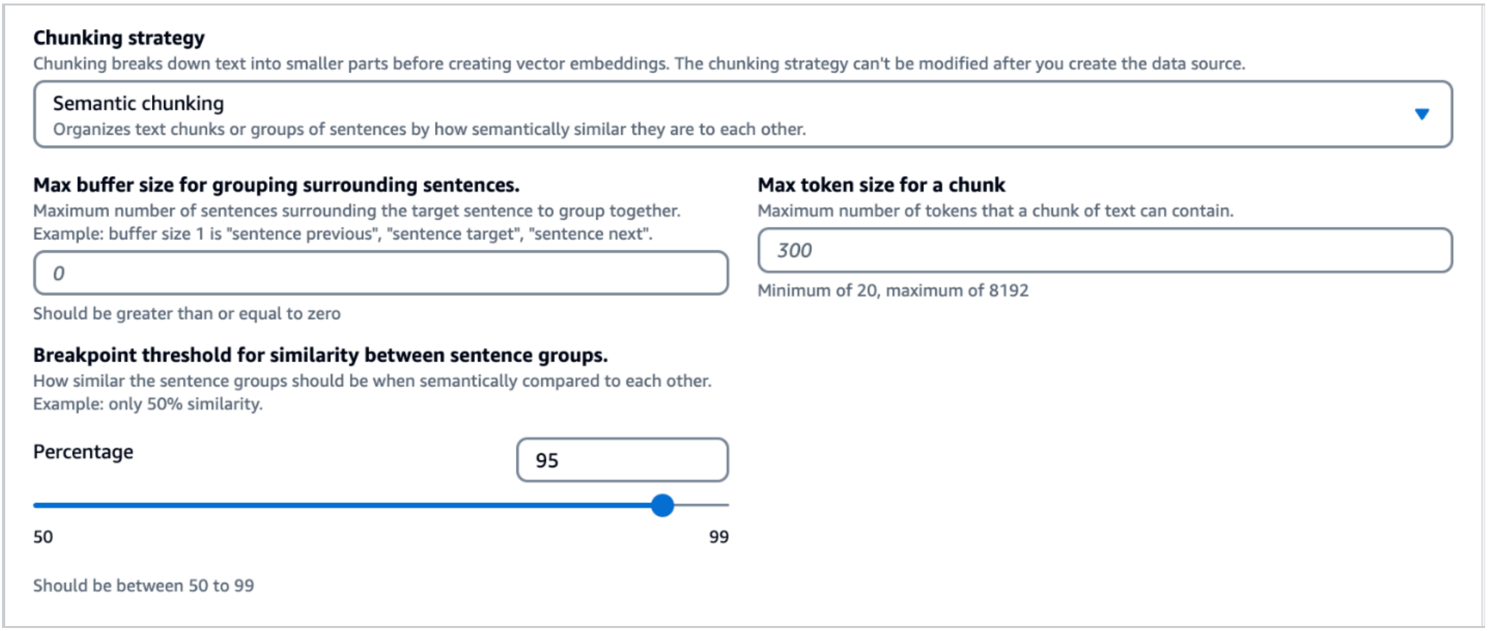
Let's do a deeper dive on each of these techniques.

Semantic chunking

Semantic chunking analyzes the relationships within a text and divides it into meaningful and complete chunks, which are derived based on the semantic similarity calculated by the embedding model. This approach preserves the information's integrity during retrieval, helping to ensure accurate and contextually appropriate results.

By focusing on the text's meaning and context, semantic chunking significantly improves the quality of retrieval. It should be used in scenarios where maintaining the semantic integrity of the text is crucial.

From the console, you can start creating a knowledge base by choosing **Create knowledge base**. In **Step 2: Configure data source**, select **Advanced (customization)** under the **Chunking & parsing configurations** and then select **Semantic chunking** from the **Chunking strategy** drop down list, as shown in the following image.



Chunking strategy
Chunking breaks down text into smaller parts before creating vector embeddings. The chunking strategy can't be modified after you create the data source.

Semantic chunking
Organizes text chunks or groups of sentences by how semantically similar they are to each other.

Max buffer size for grouping surrounding sentences.
Maximum number of sentences surrounding the target sentence to group together.
Example: buffer size 1 is "sentence previous", "sentence target", "sentence next".

0

Should be greater than or equal to zero

Max token size for a chunk
Maximum number of tokens that a chunk of text can contain.

300

Minimum of 20, maximum of 8192

Breakpoint threshold for similarity between sentence groups.
How similar the sentence groups should be when semantically compared to each other.
Example: only 50% similarity.

Percentage 95

50 99

Should be between 50 to 99

Details for the parameters that you need to configure.

- **Max buffer size for grouping surrounding sentences:** The number of sentences to group together when evaluating semantic similarity. If you select a buffer size of 1, it will include the *sentence previous*, *sentence target*, and *sentence next* while grouping the sentences. Recommended value of this parameter is 1.
- **Max token size for a chunk:** The maximum number of tokens that a chunk of text can contain. It can be minimum of 20 up to a maximum of 8,192 based on the context length of the embeddings model. For example, if you're using the Cohere Embeddings model, the maximum size of a chunk can be 512. The recommended value of this parameter is 300.
- **Breakpoint threshold for similarity between sentence groups:** Specify (by a percentage threshold) how similar the groups of sentences should be when semantically compared to each other. It should be a value between 50 and 99. The recommended value of this parameter is 95.

Amazon Bedrock Knowledge Bases first divides documents into chunks based on the specified token size. Embeddings are created for each chunk, and similar chunks in the embedding space are combined based on the similarity threshold and buffer size, forming new chunks. Consequently, the chunk size can vary across chunks.

Although this method is more computationally intensive than fixed-size chunking, it can be beneficial for chunking documents where contextual boundaries aren't clear—for example, legal documents or technical manuals.[2]

Example:

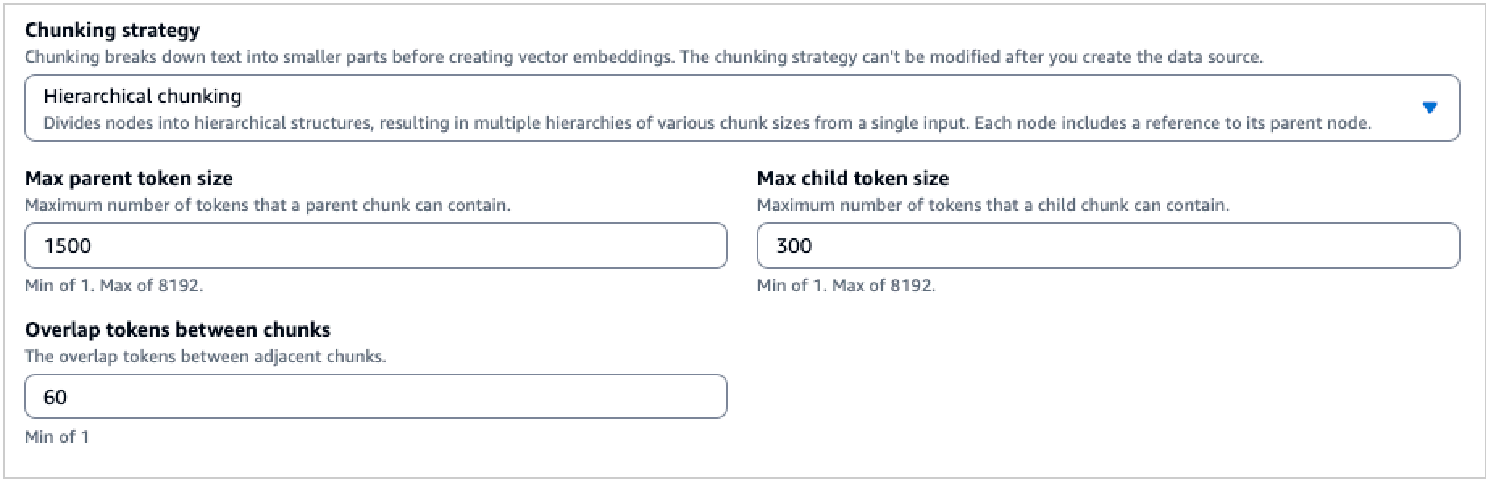
Consider a legal document discussing various clauses and sub-clauses. The contextual boundaries between these sections might not be obvious, making it challenging to determine appropriate chunk sizes. In such cases, the dynamic chunking approach can be advantageous, because it can automatically identify and group related content into coherent chunks based on the semantic similarity among neighboring sentences.

Now that you understand the concept of semantic chunking, including when to use it, let’s do a deeper dive into hierarchical chunking.

Hierarchical chunking

With hierarchical chunking, you can organize your data into a hierarchical structure, allowing for more granular and efficient retrieval based on the inherent relationships within your data. Organizing your data into a hierarchical structure enables your RAG workflow to efficiently navigate and retrieve information from complex, nested datasets.

From the console, start creating a knowledge base by choose **Create knowledge base**. **Configure data source**, select **Advanced (customization)** under the **Chunking & parsing configurations** and then select **Hierarchical chunking** from the **Chunking strategy** drop-down list, as shown in the following image.



Chunking strategy
Chunking breaks down text into smaller parts before creating vector embeddings. The chunking strategy can't be modified after you create the data source.

Hierarchical chunking
Divides nodes into hierarchical structures, resulting in multiple hierarchies of various chunk sizes from a single input. Each node includes a reference to its parent node.

Max parent token size
Maximum number of tokens that a parent chunk can contain.
1500
Min of 1. Max of 8192.

Max child token size
Maximum number of tokens that a child chunk can contain.
300
Min of 1. Max of 8192.

Overlap tokens between chunks
The overlap tokens between adjacent chunks.
60
Min of 1

The following are some parameters that you need to configure.

- **Max parent token size**: This is the maximum number of tokens that a parent chunk can contain. The value can range from 1 to 8,192 and is independent of the context length of the embeddings model because the parent chunk isn’t embedded. The recommended value of this parameter is 1,500.
- **Max child token size**: This is the maximum number of tokens that a child token can contain. The value can range from 1 to 8,192 based on the context length of the embeddings model. The recommended value of this parameter is 300.
- **Overlap tokens between chunks**: This is the percentage overlap between child chunks. Parent chunk overlap depends on the child token size and child percentage overlap that you specify. The recommended value for this parameter is 20 percent of the max child token size value.

After the documents are parsed, the first step is to chunk the documents based on the parent and child chunking size. The chunks are then organized into a hierarchical structure, where parent chunk (higher level) represents larger chunks (for example, documents or sections), and child chunks (lower level) represent smaller chunks (for example, paragraphs or sentences). The relationship between the parent and child chunks are maintained. This hierarchical structure allows for efficient retrieval and navigation of the corpus.

Some of the benefits include:

- **Efficient retrieval**: The hierarchical structure allows faster and more targeted retrieval of relevant information; first by performing semantic search on the child chunk and then returning the parent chunk during retrieval. By replacing the children chunks with the parent chunk, we provide large and comprehensive context to the FM.
- **Context preservation**: Organizing the corpus in a hierarchical manner helps preserve the contextual relationships between chunks, which can be beneficial for generating coherent and contextually relevant text.

Note: In hierarchical chunking, we return parent chunks and semantic search is performed on children chunks, therefore, you might see less number of search results returned as one parent can have multiple children.

Hierarchical chunking is best suited for complex documents that have a nested or hierarchical structure, such as technical manuals, legal documents, or academic papers with complex formatting and nested tables. You can combine the FM parsing discussed previously to parse the documents and select hierarchical chunking to improve the accuracy of generated responses.

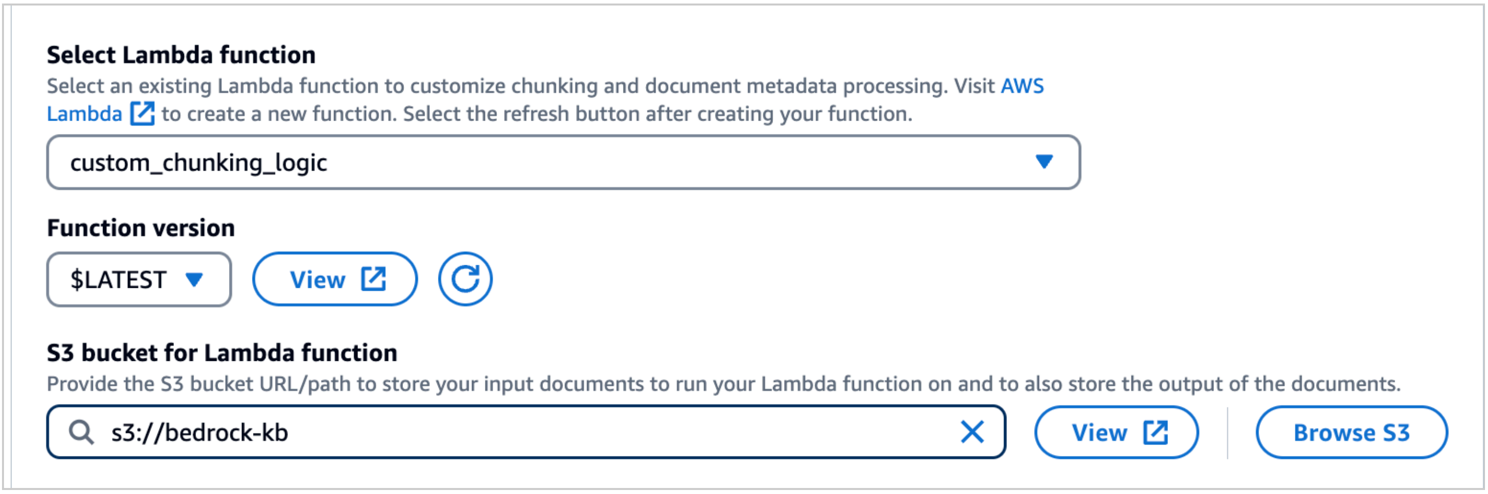
By organizing the document into a hierarchical structure during the chunking process, the model can better understand the relationships between different parts of the content, enabling it to provide more contextually relevant and coherent responses.

Now that you understand the concepts for semantic and hierarchical chunking, in case you want to have more flexibility, you can use a Lambda function for adding custom processing logic to chunks such as metadata processing or defining your custom logic for chunking. In the next section, we discuss custom processing using Lambda function provided by Amazon Bedrock Knowledge Bases.

Custom processing using Lambda functions

For those seeking more control and flexibility, Amazon Bedrock Knowledge Bases now offers the ability to define custom processing logic using AWS Lambda functions. Using Lambda functions, you can customize the chunking process to align with the unique requirements of your RAG application. Furthermore, you can extend it beyond chunking, because Lambda can also be used to streamline metadata processing, which can help unlock additional avenues for efficiency and precision.

You can begin by writing a Lambda function with your custom chunking logic or use a chunking methodology provided by your favorite open source framework such as LangChain and LLamaIndex. Make sure to create the Lambda layer for the specific open source framework. After writing and testing the Lambda function, you can start creating a knowledge base by choosing **Create knowledge base**, in **Step 2: Configure data source**, select **Advanced (customization)** under the **Chunking & parsing configurations** and then select corresponding lambda function from **Select Lambda function** drop down, as shown in the following image:


Select Lambda function

Select an existing Lambda function to customize chunking and document metadata processing. Visit AWS Lambda to create a new function. Select the refresh button after creating your function.

custom_chunking_logic

Function version

$LATEST

S3 bucket for Lambda function

Provide the S3 bucket URL/path to store your input documents to run your Lambda function on and to also store the output of the documents.

s3://bedrock-kb

From the drop down, you can select a Lambda function created in the same AWS Region, including the verified version of the Lambda function. Next, you will provide the Amazon Simple Storage Service (Amazon S3) path where you want to store the input documents to run your Lambda function on and to store the output of the documents.

So far, we have discussed advanced parsing using FMs and advanced data chunking options to improve the quality of your search results and accuracy of the generated responses. In the next section, we will discuss some optimizations that have been added to Amazon Bedrock Knowledge Bases to improve the accuracy of parsing .csv files.

Metadata customization for .csv files

Amazon Bedrock Knowledge Bases now offers an enhanced .csv file processing feature that separates content and metadata. This update streamlines the ingestion process by allowing you to designate specific columns as content fields and others as metadata fields. Consequently, it reduces the number of required files and enables more efficient data management, especially for large .csv file datasets. Moreover, the metadata customization feature introduces a dynamic approach to storing additional metadata alongside data chunks from .csv files. This contrasts with the current static process of maintaining metadata.

This customization capability unlocks new possibilities for data cleaning, normalization, and enrichment processes, enabling augmentation of your data. To use the metadata customization feature, you need to provide metadata files alongside the source .csv files, with the same name as the source data file and a `<filename>.csv.metadata.json` suffix. This metadata file specifies the content and metadata fields of the source .csv file. Here's an example of the metadata file content:

```
{
    "metadataAttributes": {
        "docSpecificMetadata1": "docSpecificMetadataVal1",
        "docSpecificMetadata2": "docSpecificMetadataVal2"
    },
    "documentStructureConfiguration": {
        "type": "RECORD_BASED_STRUCTURE_METADATA",
        "recordBasedStructureMetadata": {
            "contentFields": [
                {
                    "fieldName": "String"
                }
            ],
            "metadataFieldsSpecification": {
                "fieldsToInclude": [
                    {
                         "fieldName": "String"
                    }
                ],
```

Use the following steps to experiment with the .csv file improvement feature:

1. Upload the .csv file and corresponding `<filename>.csv.metadata.json` file in the same Amazon S3 prefi
2. Create a knowledge base using either the console or the Amazon Bedrock SDK.
3. Start ingestion using either the console or the SDK.
4. Retrieve API and RetrieveAndGenerate API can be used to query the structured .csv file data using either the console or the SDK.

Query reformulation

Often, input queries can be complex with many questions and complex relationships. With such complex prompts, the resulting query embeddings might have some semantic dilution, resulting in retrieved chunks that might not address such a multi-faceted query resulting in reduced accuracy along with a less than desirable response from your RAG application.

Now with query reformulation supported by Amazon Bedrock Knowledge Bases, we can take a complex input query and break it into multiple sub-queries. These sub-queries will then separately go through their own retrieval steps to find relevant chunks. In this process, the subqueries having less semantic complexity might find more targeted chunks. These chunks will then be pooled and ranked together before passing them to the FM to generate a response.

Example: Consider the following complex query to a financial document for the fictitious company Octank asking about multiple unrelated topics:

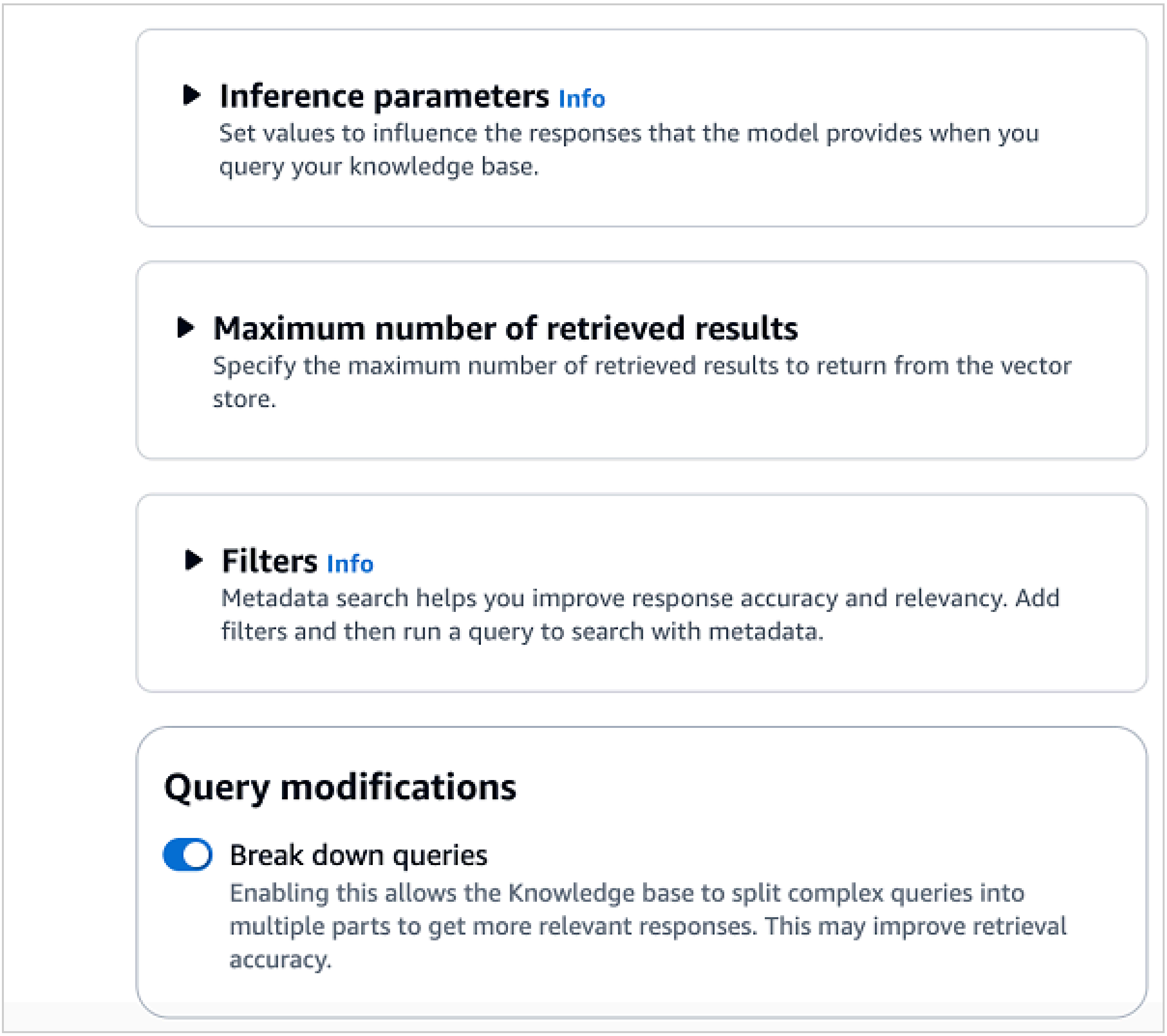
"Where is the Octank company waterfront building located and how does the whistleblower scandal hurt the company and its image?"

We can decompose the query into multiple subqueries:

1. Where is the Octank Waterfront building located?
2. What is the whistleblower scandal involving Octank?
3. How did the whistleblower scandal affect Octank's reputation and public image?

Now, we have more targeted questions that might help retrieve chunks from the knowledge base from more semantically relevant sections of the documents without some of the semantic dilution that can occur from embedding multiple asks in a single complex query.

Query reformulation can be enabled in the console after creating a knowledge base by going to **Test Knowledge Base Configurations** and turning on **Break down queries** under **Query modifications**.



▶ **Inference parameters** Info
Set values to influence the responses that the model provides when you query your knowledge base.

▶ **Maximum number of retrieved results**
Specify the maximum number of retrieved results to return from the vector store.

▶ **Filters** Info
Metadata search helps you improve response accuracy and relevancy. Add filters and then run a query to search with metadata.

Query modifications

Break down queries (on)
Enabling this allows the Knowledge base to split complex queries into multiple parts to get more relevant responses. This may improve retrieval accuracy.

Query reformulation can also be enabled during runtime using the RetrieveAndGenerateAPI by adding an additional element to the `KnowledgeBaseConfiguration` as follows:

```
    "orchestrationConfiguration": {
        "queryTransformationConfiguration": {
        "type": "QUERY_DECOMPOSITION"
    }
}
```

Query reformulation is another tool that might help increase accuracy for complex queries that you might encounter in production, giving you another way to optimize for the unique interactions your users might have with your application.

Conclusion

With the introduction of these advanced features, Amazon Bedrock Knowledge Bases solidifies its position as a powerful and versatile solution for implementing RAG workflows. Whether you're dealing with complex queries, unstructured data formats, or intricate data organizations, Amazon Bedrock Knowledge Bases empowers you with the tools and capabilities to unlock the full potential of your knowledge base.

By using advanced data chunking options, query decomposition, and .csv file processing, you have greater control over the accuracy and customization of your retrieval processes. These features not only help improve the quality of your knowledge base, but also can facilitate more efficient and effective decision-making, enabling your organization to stay ahead in the ever-evolving world of data-driven insights.

Embrace the power of Amazon Bedrock Knowledge Bases and unlock new possibilities in your retrieval and knowledge management endeavors. Stay tuned for more exciting updates and features from the Amazon Bedrock team as they continue to push the boundaries of what's possible in the realm of knowledge bases and information retrieval.

For more detailed information, code samples, and implementation guides, see the Amazon Bedrock documentation and AWS blog posts.

For additional resources, see:

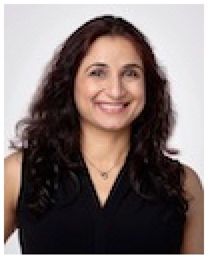
- Amazon Bedrock Knowledge Bases
- Use RAG to improve responses in generative AI application
- Amazon Bedrock Knowledge Base – Samples for building RAG workflows

References:

[1] LlamaIndex: Chunking Strategies for Large Language Models. Part — 1
[2] How to Choose the Right Chunking Strategy for Your LLM Application

About the authors

Sandeep Singh is a Senior Generative AI Data Scientist at Amazon Web Services, helping businesses innovate with generative AI. He specializes in Generative AI, Artificial Intelligence, Machine Learning, and System Design. He is passionate about developing state-of-the-art AI/ML-powered solutions to solve complex business problems for diverse industries, optimizing efficiency and scalability.



Mani Khanuja is a Tech Lead – Generative AI Specialists, author of the book Applied Machine Learning and High Performance Computing on AWS, and a member of the Board of Directors for Women in Manufacturing Education Foundation Board. She leads machine learning projects in various domains such as computer vision, natural language processing, and generative AI. She speaks at internal and external conferences such AWS re:Invent, Women in Manufacturing West, YouTube webinars, and GHC 23. In her free time, she likes to go for long runs along the beach.

Chris Pecora is a Generative AI Data Scientist at Amazon Web Services. He is passionate about building innovative products and solutions while also focused on customer-obsessed science. When not running experiments and keeping up with the latest developments in generative AI, he loves spending time with his kids.

Like Share

Comments

Log in to comment