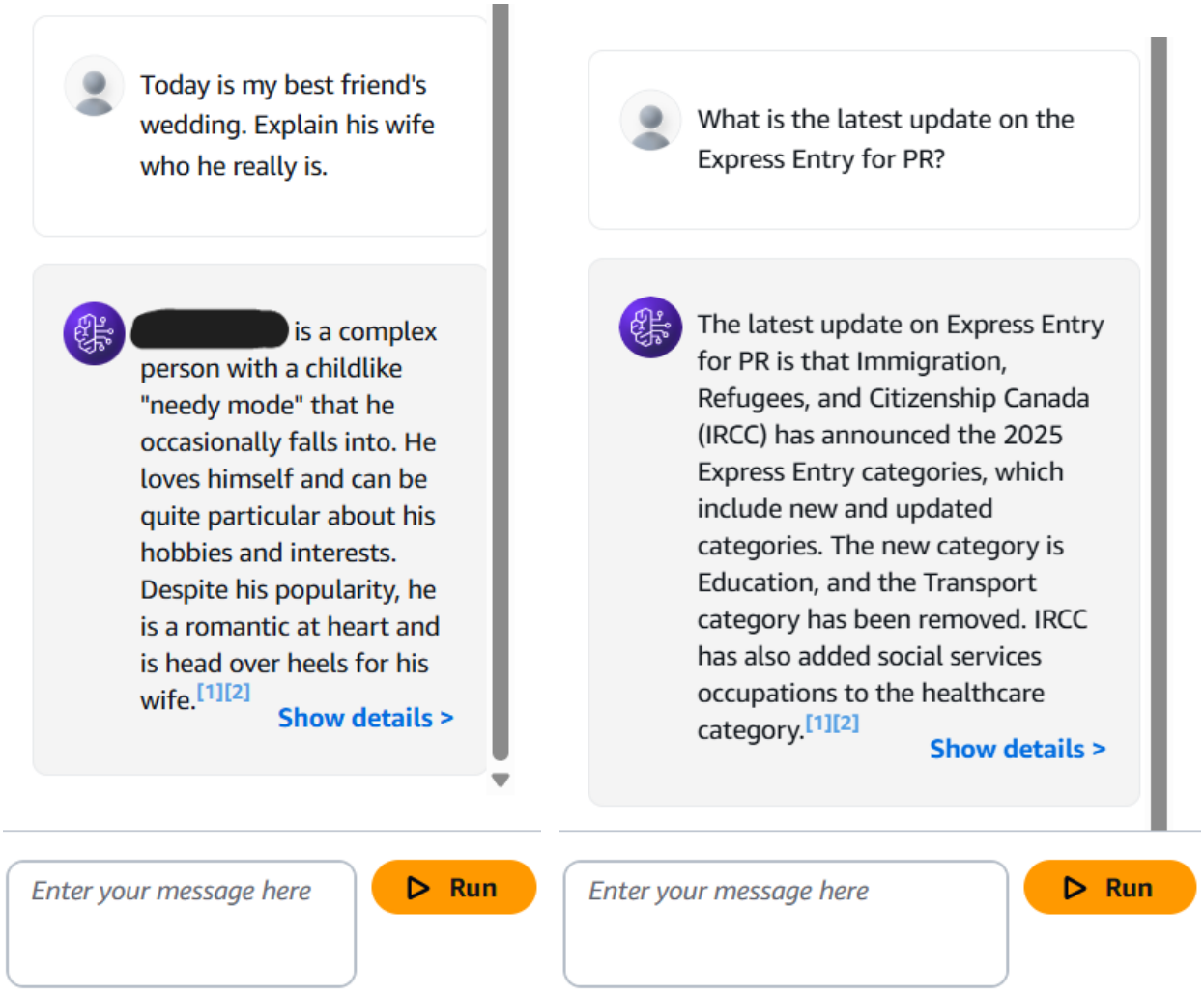
Set Up a RAG Chatbot in AWS Bedrock



Ryoichi (Rio) Homma

March 2025

Table of Contents

Introduction:

RAG (Retrieval Augmented Generation) chatbot is an AI-powered chatbot that enhances AI language models by incorporating documentation as a data source. This technique enables the chatbot to provide accurate and up-to-date responses, even if the required information is the latest or unavailable online. While non-RAG chatbots rely on pre-trained data and may struggle to offer current or specialized knowledge, the RAG chatbots can retrieve information from a specific knowledge base or external sources. This makes RAG chatbots particularly useful for applications, where up-to-date information is critical, such as legal or government updates, product releases, or personalized content.

Used Tools and Concepts:

The AWS services I used in this project were **Amazon Bedrock**, **S3**, and **OpenSearch Serverless**. Key concepts include storing data in S3, creating a Knowledge Base, requesting access to AI models, how chatbot generates responses using AI models and Knowledge Base, and utilizing vector stores for efficient retrieval.

Objectives:

Use Case 1 – Personalized Q&A Chatbot

Since my best friend recently got engaged, I wanted to give them something unique, meaningful, and a little humorous gift. Then, I decided to build an application integrated with an RAG chatbot that answers questions about him, allowing his fiancée to learn more about him in a fun way. This not only makes them laugh but also helps them start their new life together with a deeper understanding.

Use Case 2 – Automated Immigration Updates

Whenever the Government of Canada announces new rules or updates regarding temporary residents and Permanent Residency (PR), I usually spend 15 to 20 minutes searching through multiple documents and sources to verify the information’s accuracy. I realized that the RAG chatbot could streamline this process by providing reliable updates directly from trusted sources if I could successfully integrate with public web links in the Knowledge Base. This would significantly reduce my search time while ensuring access to accurate and up-to-date information.

Project Challenges:

The most challenging part of this project was creating the Knowledge Base since I encountered several errors related to permission policies. I struggled to determine which permission policies to add and how to configure them in the IAM console to grant the necessary service access in the IAM account. Additionally, setting up the data source for the Web Crawler was difficult – my initial attempts to sync the data either failed or never completed successfully. By researching solutions through the AWS blog, I was able to resolve these issues. Overcoming these challenges provided valuable insights into AWS service integration and permission management.

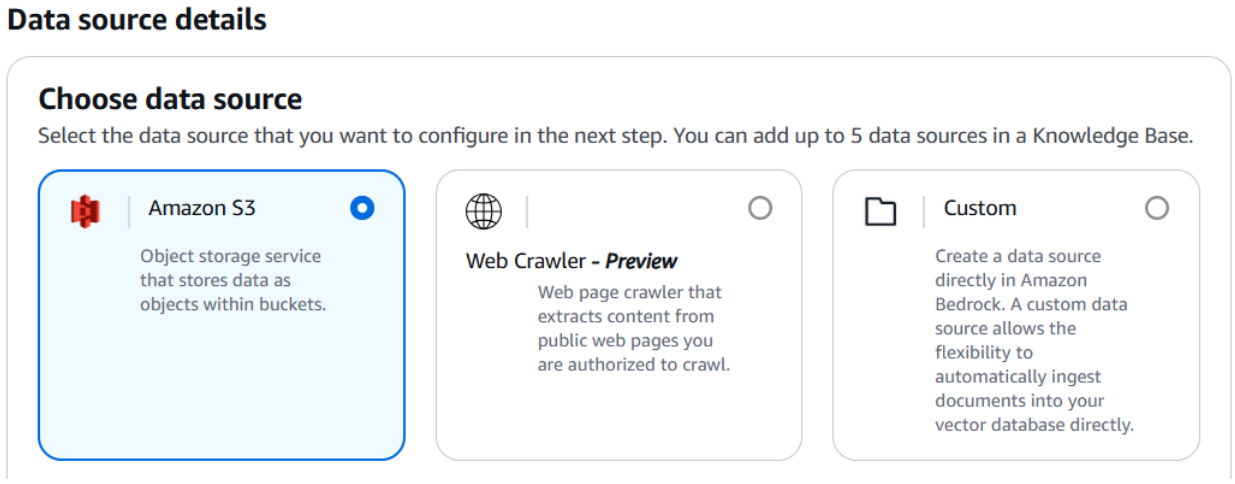
Amazon Bedrock:

Amazon Bedrock is one of the AWS services that simplifies the development of generative AI applications. It acts as an AI model marketplace, where developers and engineers can find, use, and test AI models from different providers like Meta. In this project, I used Amazon Bedrock to create the Knowledge Base, enabling the chatbot to retrieve relevant information efficiently.

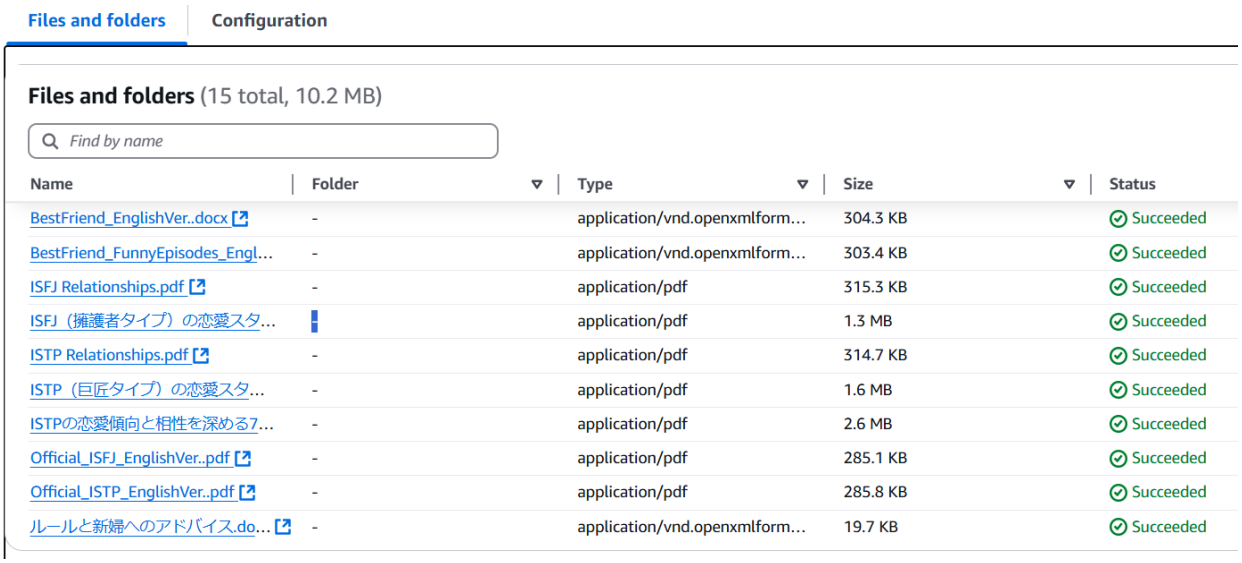
Data Source (Amazon S3 and Web Crawler):

Use Case 1 – Store Documentation in S3

The Knowledge Base is linked to Amazon S3, which serves as the storage source for the Knowledge Base's row documentation. Amazon S3 is AWS's scalable storage service, where developers and engineers can store various types of objects such as documents, videos, and audio within the same bucket.

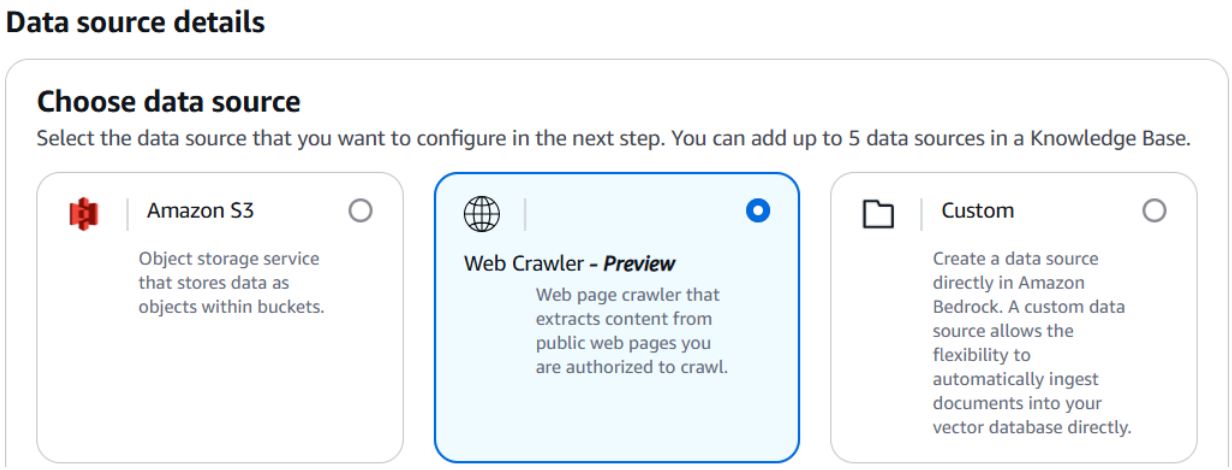


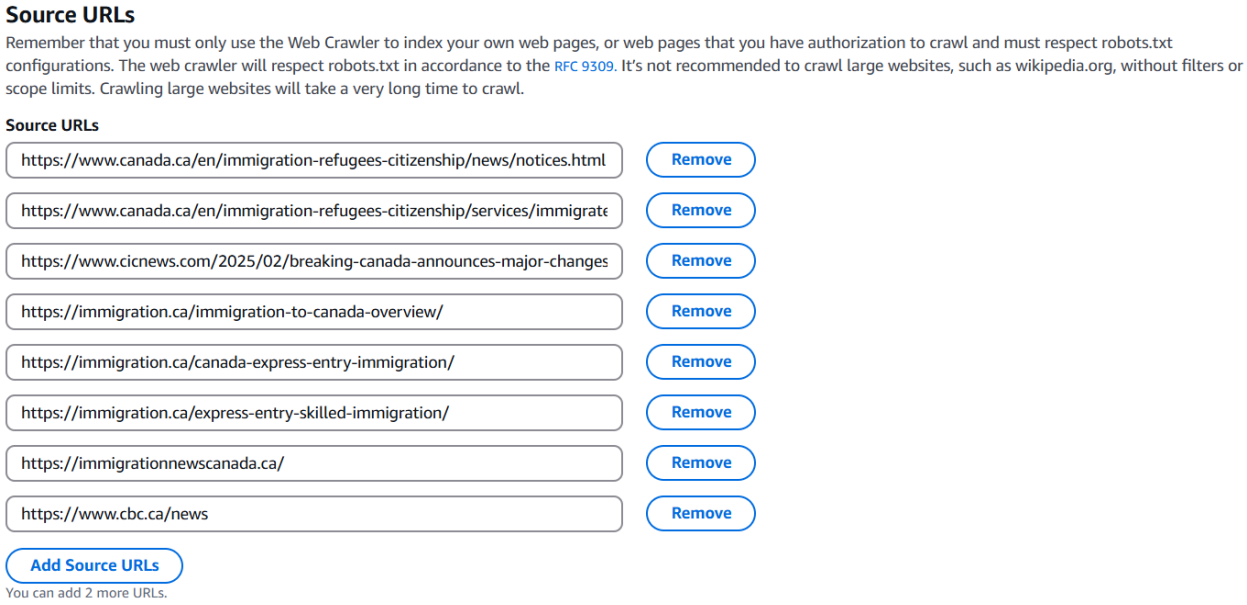
In the S3 bucket, I uploaded the documentation that makes up the AI chatbot's knowledge. Initially, the setup did not work, but I figured out that Amazon Bedrock is a regional service soon after. This means that both the S3 bucket and the Knowledge Base must be in the same AWS region for proper integration. After ensuring they were aligned, the connection functioned as expected.



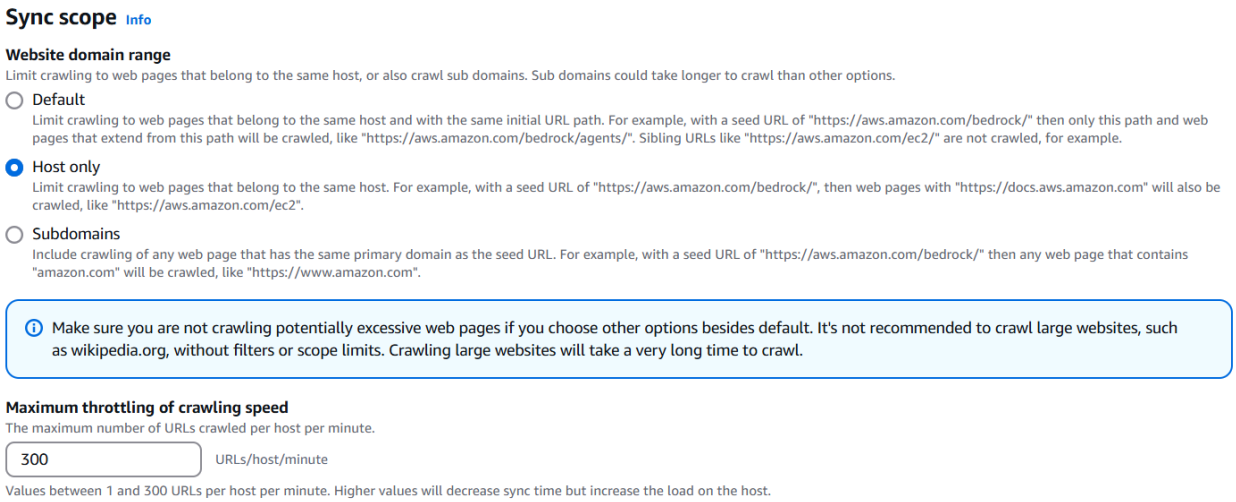
Use Case 2 – Add Source URLs

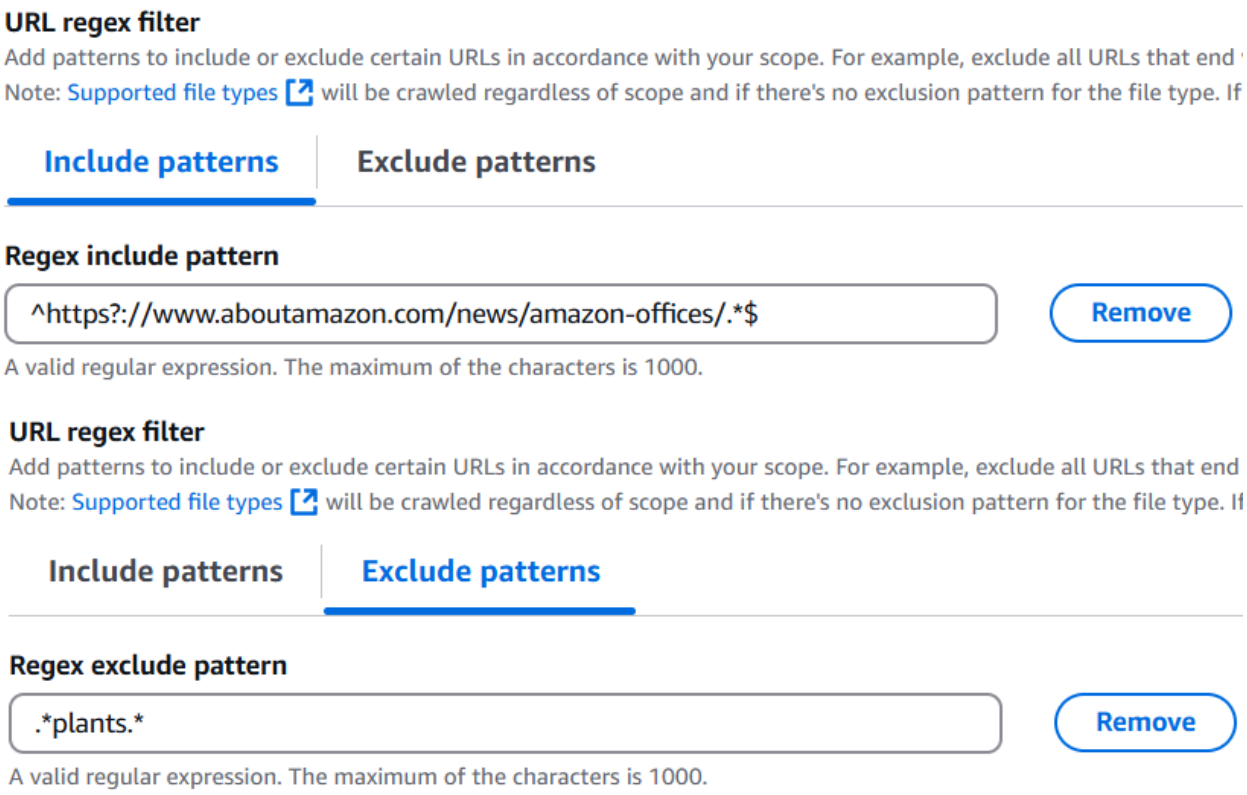
Instead of using S3, I selected **Web Crawler** as a data source and added source URLs. This is because the Web Crawler extracts content from the specified public web pages, enabling the chatbot to provide the latest information as long as those pages are updated. This ensures that responses remain current and reliable without requiring manual data uploads.





Under **Sync Scope**, I initially selected “Default”, causing errors or preventing the data source from successfully synchronizing with the Knowledge Base. To resolve this issue, I switched to “Host only” and added the following URL regex filters. This adjustment finally allowed me to complete the synchronization successfully.





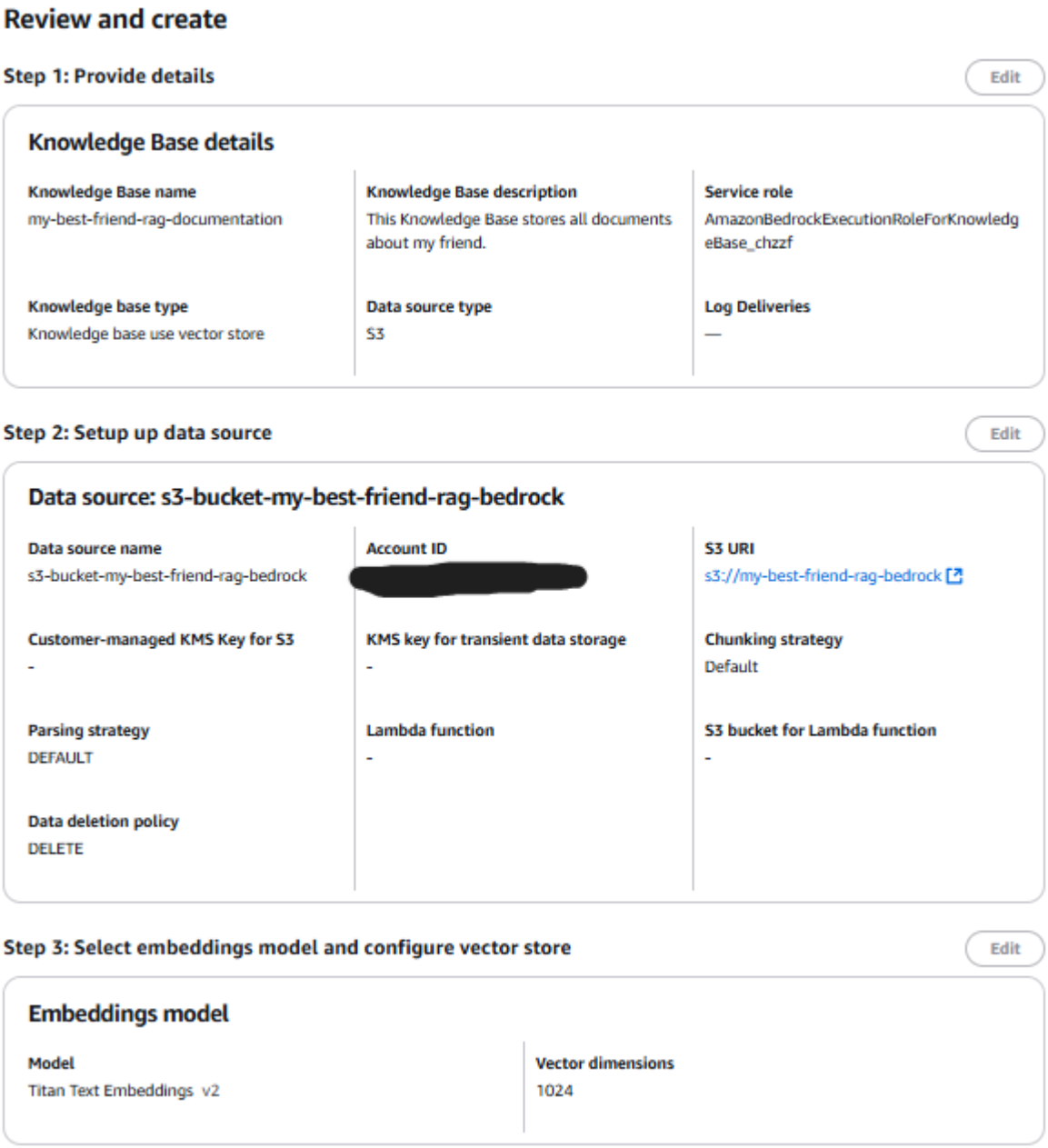
Knowledge Base Configuration:

Note: *From this step, the workflows for Use Case 1 and Use Case 2 are the same.*

My Knowledge Base uses a vector store, which is a search engine/database that stores data based on their semantic meaning. When users query the Knowledge Base, OpenSearch will find the relevant chunks of data and pass them to Bedrock for response generation.

Embeddings are vector representations of the semantic meaning of a text chunk. I selected **Titan Text Embeddings v2** as the embedding model because it is fast, accurate, and affordable.

Chunking is the process of splitting up text into smaller pieces or chunks. This helps improve data search efficiency within the vector store. In the Knowledge Base, chunks are configured to be approximately 300 tokens in size each.



AI Models:

AI models play a crucial role in the chatbot, as they translate the search results from the Knowledge Base into human-like text. Without AI models, the chatbot would only respond with raw chunks of text from the documents, which would not provide the best user experience.

To get access to AI models in Bedrock, I had to visit the “Model Access” page and request access explicitly. AWS requires this explicit access since some AI model providers have extra forms or rules for usage, and AWS needs to verify the availability of these models.

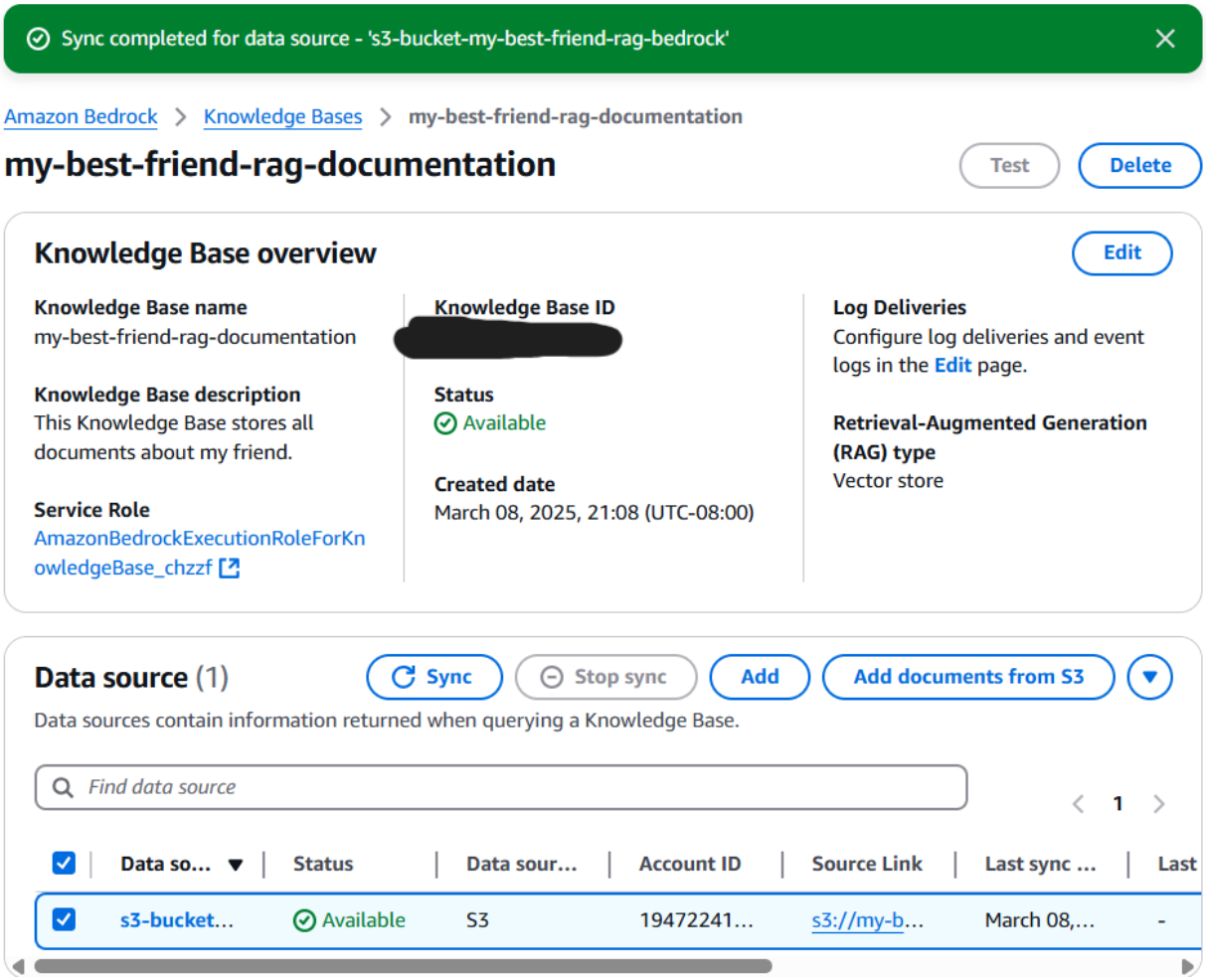
Models	Access status	Modality
Amazon (4)	1/4 access granted	
Titan Text Embeddings V2	Access granted	Embedding
Nova Pro Cross-region inference	Available to request	Text & Vision
Nova Lite Cross-region inference	Available to request	Text & Vision
Nova Micro Cross-region inference	Available to request	Text
Anthropic (5)	0/5 access granted	
Claude 3.7 Sonnet Cross-region inference	Available to request	Text & Vision
Claude 3.5 Haiku Cross-region inference	Available to request	Text
Claude 3.5 Sonnet v2 Cross-region inference	Available to request	Text & Vision
Claude 3.5 Sonnet Cross-region inference	Available to request	Text & Vision
Claude 3 Haiku Cross-region inference	Available to request	Text & Vision
Meta (8)	2/8 access granted	
Llama 3.3 70B Instruct	Access granted	Text
Llama 3.2 1B Instruct Cross-region inference	Available to request	Text
Llama 3.2 3B Instruct Cross-region inference	Available to request	Text
Llama 3.2 11B Vision Instruct Cross-region inference	Available to request	Text & Vision
Llama 3.2 90B Vision Instruct Cross-region inference	Available to request	Text & Vision
Llama 3.1 405B Instruct Cross-region inference	Available to request	Text
Llama 3.1 70B Instruct Cross-region inference	Available to request	Text
Llama 3.1 8B Instruct Cross-region inference	Access granted	Text

Knowledge Base Synchronization:

Although the S3 bucket has been connected to the Knowledge Base since its creation, synchronizing the data is still necessary. This is because synchronization is the process that actually transfers the data from S3 into the Knowledge Base and OpenSearch Serverless.

The synchronization process involves three key steps as below:

1. **Ingesting** – Bedrock takes the data from S3.
2. **Processing** – Bedrock chunks and embeds the data.
3. **Storing** – Bedrock stores the processed data in the vector store, OpenSearch Serverless.



Testing My Chatbot:

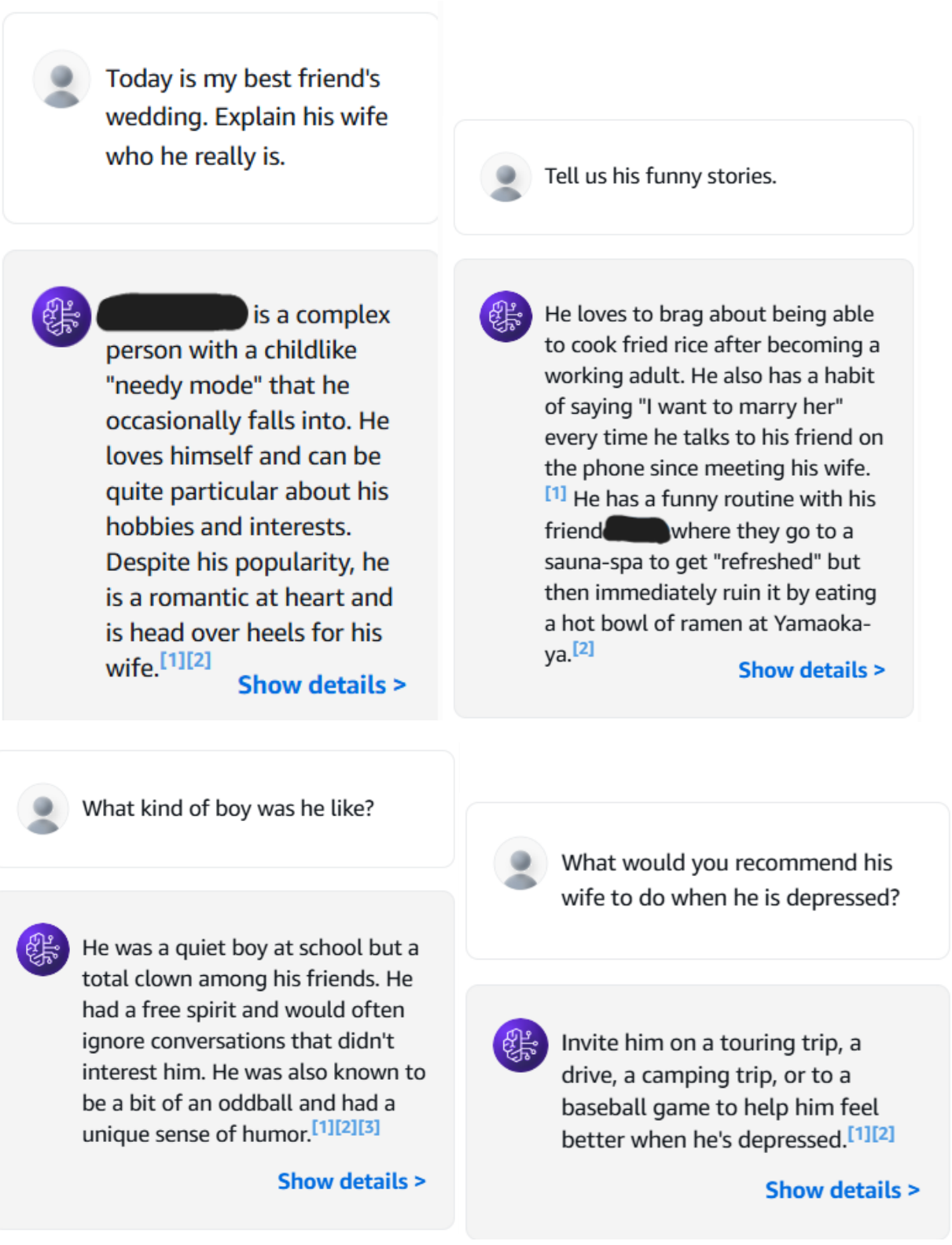
My initial attempt to test my chatbot with **Llama 3.1 8B** as the AI model caused an error because it was unavailable on-demand. To fix this, I switched to **Llama 3.3 70B**, AWS offered on-demand as a newer and more efficient model.

When I asked about topics unrelated to the Knowledge Base's data, the chatbot was unable to respond. This means that the chatbot only knows the information I stored in S3. It does not have access to any information outside of the Knowledge Base.

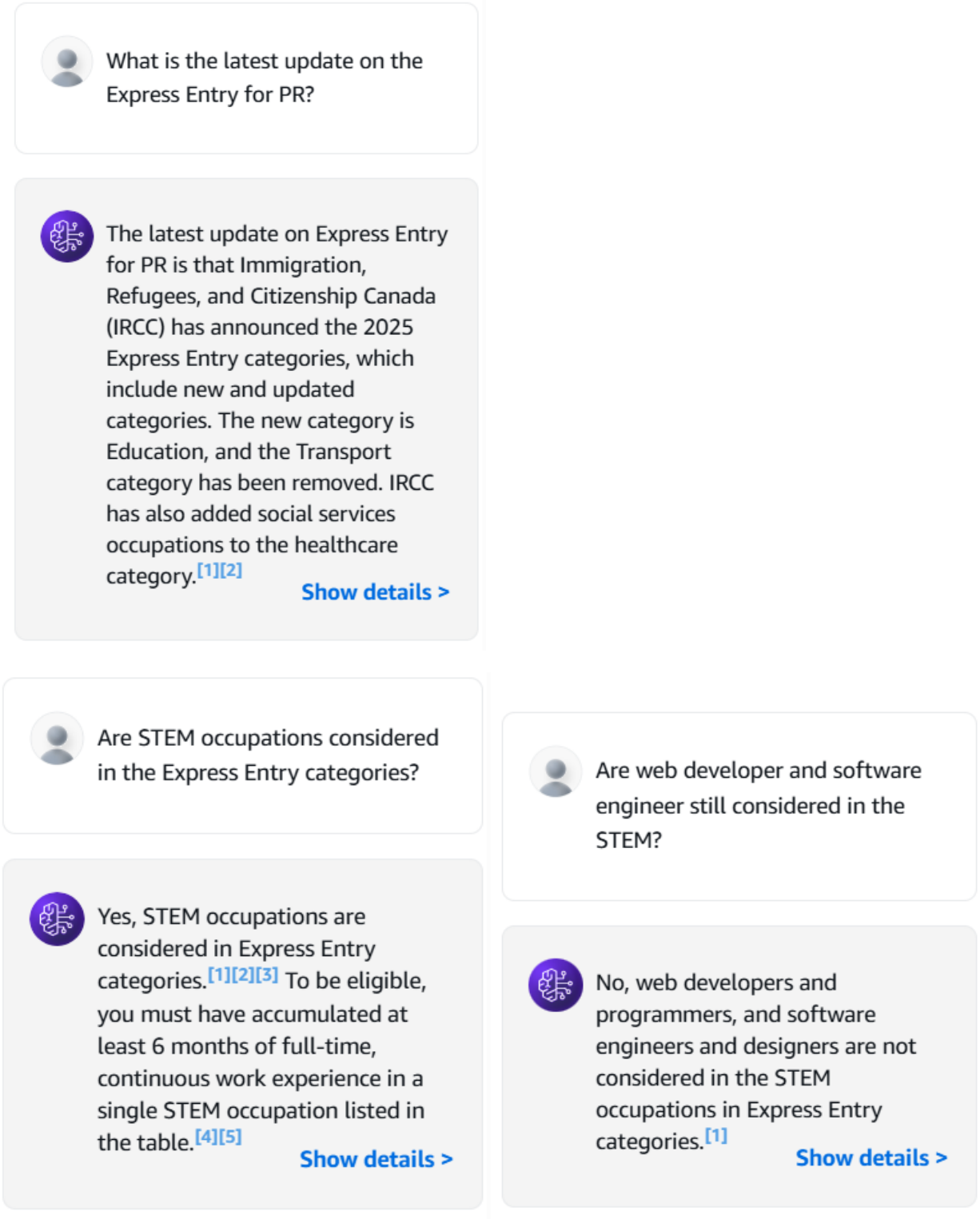
Additionally, you can disable the **Generate Response** setting to see the raw chunks of data directly from the Knowledge Base. During testing, the chatbot just threw a list of paragraphs to answer a question. In contrast, the AI model transforms these chunks into a coherent sentence when generating responses.

Demonstration:

Use Case 1 – Personalized Q&A Chatbot



Use Case 2 – Automated Immigration Updates



Performance Optimization & Results with Data:

Use Case 1 – Personalized Q&A Chatbot

I compared response speed by testing identical prompts while storing **15 vs. 5** documents in S3. As a result, I discovered that the response speed might hypothetically depend on the volume of stored documents, according to the following data:

Test Results:

- 1st Prompt: **4.61 sec.** (15 docs) → **3.26 sec.** (5 docs)
- 2nd Prompt: **5.70 sec.** (15 docs) → **5.50 sec.** (5 docs)
- 3rd Prompt: **6.18 sec.** (15 docs) → **5.78 sec.** (5 docs)
- 4th Prompt: **6.81 sec.** (15 docs) → **6.80 sec.** (5 docs)

Key Findings:

On average, the response speed was **0.49 seconds faster** when storing fewer documents. This is probably because the AI chatbot took longer to process and retrieve information from the larger volume of documents.

Trade-Off:

While reducing the number of documents could improve the performance of response speed, it may also negatively affect the quality of responses. Finding the right balance between speed and knowledge depth is crucial for optimizing chatbot performance.

Note: *Since their wedding is still upcoming and the application would be a surprise, I haven't shown it to them yet.*

Use Case 2 – Automated Immigration Updates

I used to spend approximately 15 to 20 minutes searching for reliable sources and reading through numerous documents every time the IRCC announced new updates regarding immigration, especially Express Entry rules for PR. However, the RAG chatbot can now retrieve updates directly from added source URLs and provide the relevant information with reliable sources, reducing my search time by **66.67 %** to **75%** – from **15-20 minutes** to only **5 minutes** (including the time to verify accuracy).

References:

How to set up a RAG chatbot in AWS Bedrock: A Step-by-Step Guide. (n.d.). Community.aws. https://community.aws/content/2tnxBahN1n18B1Aqfsn1yoNQ7cq/how-to-set-up-a-rag-chatbot-in-aws-bedrock-a-step-by-step-guide

Implement web crawling in Amazon Bedrock Knowledge Bases | Amazon Web Services. (2024, August 29). Amazon Web Services. https://aws.amazon.com/blogs/machine-learning/implement-web-crawling-in-knowledge-bases-for-amazon-bedrock/

Sonawane, P. (2025, February 22). Building an AI-Powered RAG chatbot on Bedrock via the AWS console. *Medium*. https://medium.com/@piyushsonawane10/building-an-ai-powered-rag-chatbot-on-bedrock-via-the-aws-console-8ac62cf3e6b6