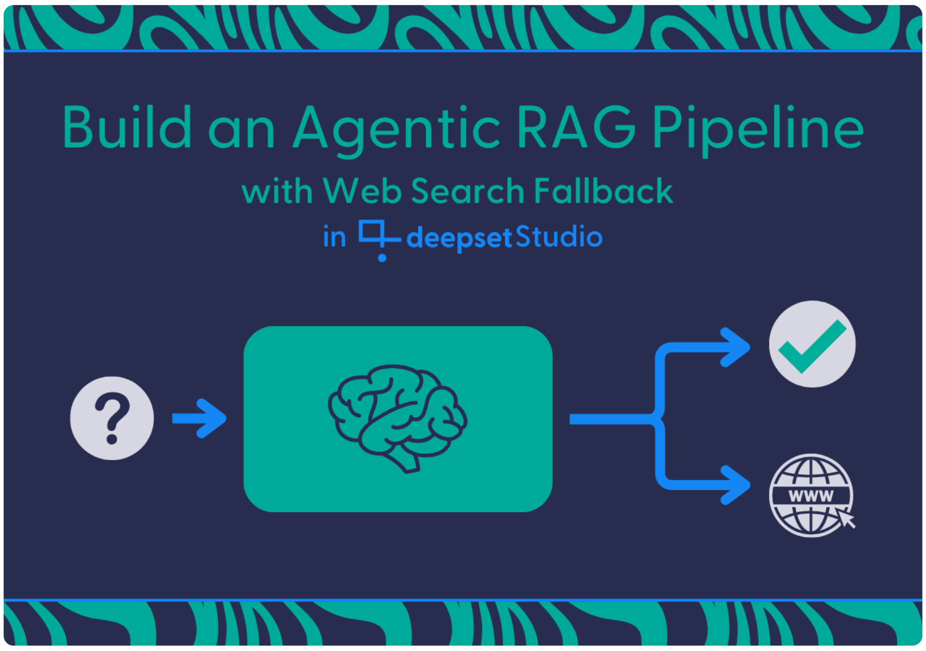
Bilge Yücel
Developer Relations Engineer

Table of contents

Creating a deepset Studio Account
Adding Files
Building the Initial RAG Pipeline
Incorporating a Fallback Mechanism into a RAG pipeline
Adding API Keys
Deploying and Testing the Pipeline
💡 Tips
Conclusion



Agent | Deepset Studio | Advanced Use Cases

Build an Agentic RAG Pipeline in deepset Studio

Use deepset Studio to build an agentic Haystack pipeline with a fallback mechanism for dynamic web search

January 14, 2025

In this article, we'll explore how to build an **agentic Retrieval Augmented Generation (RAG) pipeline** on deepset Studio, the AI application prototyping tool for developers. We'll first build a basic RAG pipeline and then extend the pipeline with a fallback mechanism that can perform a web search if the answer to the user query cannot be found in the database.

> This article also serves as a solution to Day 5 challenge of Advent of Haystack 2024: Elves' Secret for Faster Development 💨

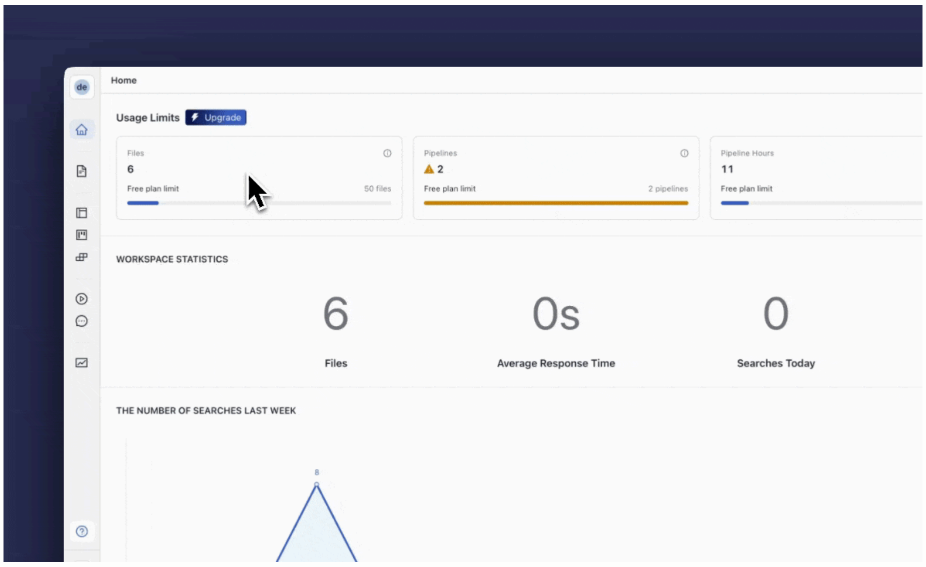
Let's dive in and create a pipeline that doesn't just search but actively decides.

Creating a deepset Studio Account

deepset Studio is a development environment for Haystack. It allows you to visually build and test Haystack pipelines. It's free and open to everyone. Learn more about Studio and its features in the announcement blog post.

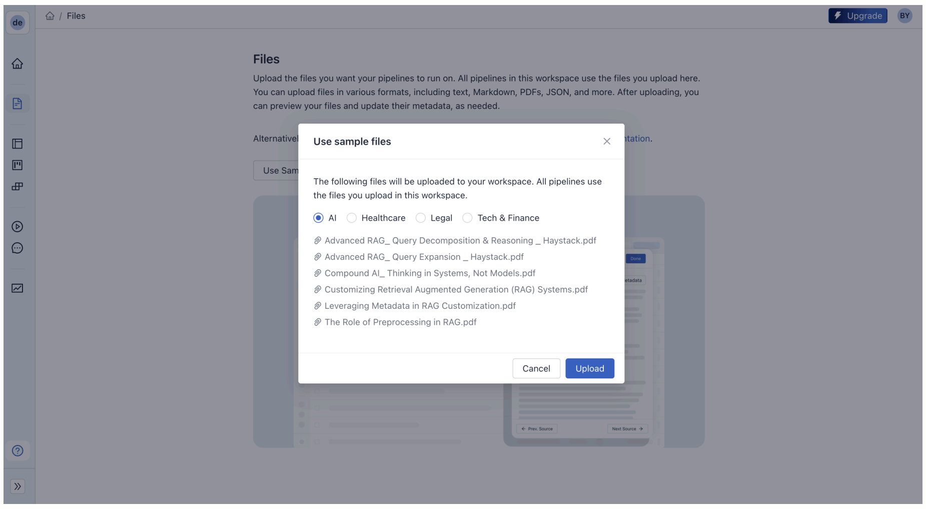
To start building a pipeline on deepset Studio, you need to have an account. If you don't have one yet, sign up here to get access.

Once you're in, you can complete the onboarding tutorial or return to it later in the left sidebar. The left sidebar also has some relevant tabs, such as "Files," "Pipeline Templates," and "Pipelines."



Adding Files

For the sake of simplicity, we'll use the "AI" sample files that come with Studio. Of course, you can use your own files in Studio. If you want to upload your own files, you have two options: you can either use the built-in database that Studio uses, which is an OpenSearch instance, or you can connect to your Weaviate, Pinecone, Qdrant, ElasticSearch instance remotely.



Files

Upload the files you want your pipelines to run on. All pipelines in this workspace use the files you upload here. You can upload files in various formats, including text, Markdown, PDFs, JSON, and more. After uploading, you can preview your files and update their metadata, as needed.

Use sample files

The following files will be uploaded to your workspace. All pipelines use the files you upload in this workspace.

[✓ AI] Healthcare Legal Tech & Finance

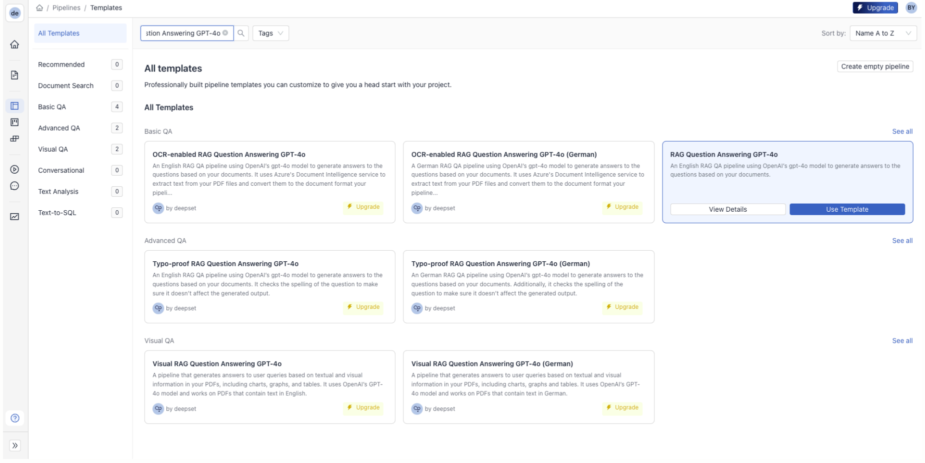
- Advanced RAG_ Query Decomposition & Reasoning _ Haystack.pdf
- Advanced RAG_ Query Expansion _ Haystack.pdf
- Compound AI_ Thinking in Systems, Not Models.pdf
- Customizing Retrieval Augmented Generation (RAG) Systems.pdf
- Leveraging Metadata in RAG Customization.pdf
- The Role of Preprocessing in RAG.pdf

There are example files on various topics

Building the Initial RAG Pipeline

> Check out Create a Pipeline in Pipeline Builder for more detailed explanation.

To start building the pipeline, we need to switch to the "Pipeline Templates" tab. Here, you can find several pre-built pipelines (with components, models, and optimal parameters) for different use cases (RAG, chat, summarization, document similarity, etc.) to speed up the building process and avoid starting from scratch.



All templates

Professionally built pipeline templates you can customize to give you a head start with your project.

All Templates

Basic QA

OCR-enabled RAG Question Answering GPT-4o
An English RAG QA pipeline using OpenAI's gpt-4o model to generate answers to the questions based on your documents. It uses Azure's Document Intelligence service to extract text from your PDF files and convert them to the document format your pipeli...

OCR-enabled RAG Question Answering GPT-4o (German)
A German RAG QA pipeline using OpenAI's gpt-4o model to generate answers to the questions based on your documents. It uses Azure's Document Intelligence service to extract text from your PDF files and convert them to the document format your pipeline...

RAG Question Answering GPT-4o
An English RAG QA pipeline using OpenAI's gpt-4o model to generate answers to the questions based on your documents.

Advanced QA

Typo-proof RAG Question Answering GPT-4o
An English RAG QA pipeline using OpenAI's gpt-4o model to generate answers to the questions based on your documents. It checks the spelling of the question to make sure it doesn't affect the generated output.

Typo-proof RAG Question Answering GPT-4o (German)
An German RAG QA pipeline using OpenAI's gpt-4o model to generate answers to the questions based on your documents. Additionally, it checks the spelling of the question to make sure it doesn't affect the generated output.

Visual QA

Visual RAG Question Answering GPT-4o
A pipeline that generates answers to user queries based on textual and visual information in your PDFs, including charts, graphs, and tables. It uses OpenAI's GPT-4o model and works on PDFs that contain text in English.

Visual RAG Question Answering GPT-4o (German)
A pipeline that generates answers to user queries based on textual and visual information in your PDFs, including charts, graphs and tables. It uses OpenAI's GPT-4o model and works on PDFs that contain text in German.

Locate "RAG Question Answering GPT-4o" and click "Use Template." On the modal screen, you can change the default name or leave it as is. Clicking "Create Pipeline" takes you to the Pipeline Builder, the drag-and-drop interface for creating and editing the pipeline.

This pipeline template comes with a comprehensive indexing pipeline that processes all file types, splits them into chunks and creates embeddings using the `intfloat/e5-base-v2` model. The query pipeline has hybrid retrieval with a Ranker and uses `gpt-4o` from OpenAI for generation.

> ⚠️ If you're using your own database instance, you'll need to update your indexing pipeline accordingly. For more information, see Connect to an External Document Store.

Default query pipeline coming with the "RAG Question Answering GPT-4o" template

We'll leave the indexing pipeline as it is, but update the query pipeline to incorporate the web search fallback mechanism.

Incorporating a Fallback Mechanism into a RAG pipeline

We want our pipeline to exhibit agentic behavior by dynamically deciding its course of action. Specifically, it will first perform RAG on our database, and if the query cannot be resolved, it will intelligently shift to a web search fallback. This decision-making capability mirrors an agentic design, enhancing the pipeline's robustness and flexibility.

We already have a working RAG pipeline that we'll extend with additional components to include the web fallback mechanism.

> Read the Tutorial: Building Fallbacks to Websearch with Conditional Routing to understand how to design a Haystack pipeline with a fallback mechanism.

Update the default prompt

In the extended RAG pipeline, the LLM used in the Generator component will have a dual function. In addition to generating the answer based on the documents in the database, it should also indicate when an answer cannot be generated based on the given documents. The template pipeline already comes with an extensive prompt, so all you need to do is tweak the prompt slightly with instructions to return `NO_ANSWER` if the documents cannot answer the question. Here's the new prompt we'll use:

```
You are a technical expert.
You answer questions truthfully based on provi
Ignore typing errors in the question.
For each document check whether it is related 
Only use documents that are related to the que
Ignore documents that are not related to the q
If the answer exists in several documents, sum
Only answer based on the documents provided. D
```

```
Just output the structured, informative and pr
If the documents can't answer the question, sa
Always use references in the form [NUMBER OF D
Never name the documents, only enter a number 
The reference must only refer to the number th
Otherwise, do not use brackets in your answer 
These are the documents:
{% for document in documents %}
Document[{{ loop.index }}]:
Name of Source File: {{ document.meta.file_nam
{{ document.content }}
{% endfor %}

Question: {{ question }}
Answer:
```

Add ConditionalRouter

In Haystack, routing is the most convenient way to build a fallback mechanism into a pipeline and enable agentic behavior. Router components can help direct input to different branches based on some condition or specification of the input, such as metadata or file type.

For this example, we will add the `ConditionalRouter` component to help the pipeline decide whether to proceed with the database query results or invoke the web search branch based on the LLM's response. This component will be connected to the `OpenAIGenerator` and check the LLM response. If the response has the keyword `NO_ANSWER`, it will direct the query to the web search branch. If not, the search will be terminated. Here's how you need to define the conditions in `ConditionalRouter` to achieve this:

```
- condition: '{{''NO_ANSWER'' in replies[0]}}'
  output: '{{query}}'
  output_name: go_to_web
```

```
    output_type: str
  - condition: '{{''NO_ANSWER'' not in replies[0
    output: '{{replies}}'
    output_name: replies
    output_type: typing.List[str]
```

These conditions will create two outputs/edges for the `ConditionalRouter`: "replies" and "go_to_web". The "replies" edge will be connected to the **`AnswerBuilder`** as it completes the search, and the "go_to_web" edge will be connected to the web search branch.

Create the Web Search Branch

The fallback branch will be a simple RAG pipeline but this time, we'll use **`SerperDevWebSearch`** instead of the Retriever, which is one of the WebSearch components. This component will get the query from the `ConditionalRouter` and retrieve relevant information from the web. The web search pipeline will continue with a new **`PromptBuilder`**, **`Generator`** and `AnswerBuilder`. As a prompt, we can use a shorter one this time:

```
Answer the following query given the documents
Your answer should indicate that your answer wa

Documents:
{% for document in documents %}
  {{document.content}}
{% endfor %}

Query: {{query}}
```

For the generator, we can again use the `gpt-4o` model through `OpenAIGenerator`

Tie Up the Loose Ends

In deepset Studio, a query pipeline must end with an “Output” node that can return a list of Answers, a list of Documents, or both. `AnswerBuilder` is a handy component that builds the Answer object using the query, LLM answers, and other optional information such as documents or meta. Since we are getting Answer objects from two different branches here, we also need an **`AnswerJoiner`** to concatenate the list of Answers and pass it to the “Output” node.

The pipeline should look like this when it is complete.

⚠️ Don’t forget to press “Save” to keep all these changes before moving forward 🙂

Adding API Keys

Before we deploy and begin testing our pipeline, we need to add `OPENAI_API_KEY` and `SERPERDEV_API_KEY` to deepset Studio. You need to use “ Connections” for OpenAI and “ Secrets” for Serper under the menu in the top right corner.

Deploying and Testing the Pipeline

Once you’re done with all the steps above, deploy the pipeline. This process might take some time as after deployment, deepset Studio sets up a document store instance, processes all files, creates embeddings, and indexes them. When everything is complete, you’ll see `Indexed` tag on the pipeline.

When the deployment is complete, you can test your pipeline in the Playground. You can find the "Playground" tab on the sidebar. This UI gives you the ability to run some queries on your pipeline and investigate the response. You can give it a go with the query, "What's Compound AI?". Then, ask Santa's birthday to confirm that the pipeline uses the fallback branch when required.

For this query, pipeline uses the fallback branch

💡 Tips

- Try some queries and rate each answer with the buttons. This will help you collect feedback systematically and understand how your pipeline performs.
- If you're working with other people, you can share your pipeline with them.
- When you're done with the development and testing, you can easily export your pipeline in Python or in YAML format to deploy in your own infrastructure or upgrade to deepset Cloud, the enterprise version of deepset Studio.

Conclusion

In this article, we demonstrated how to build, deploy, and test an agentic Haystack pipeline in deepset Studio that intelligently switches from database search to web search, all without worrying about the underlying infrastructure.

Haystack's modular design, combined with deepset Studio's user-friendly environment, makes developing sophisticated AI applications accessible for everyone, from beginners to seasoned developers.

Get started with Haystack and get your free deepset Studio account now!