Exercise Sheet 2

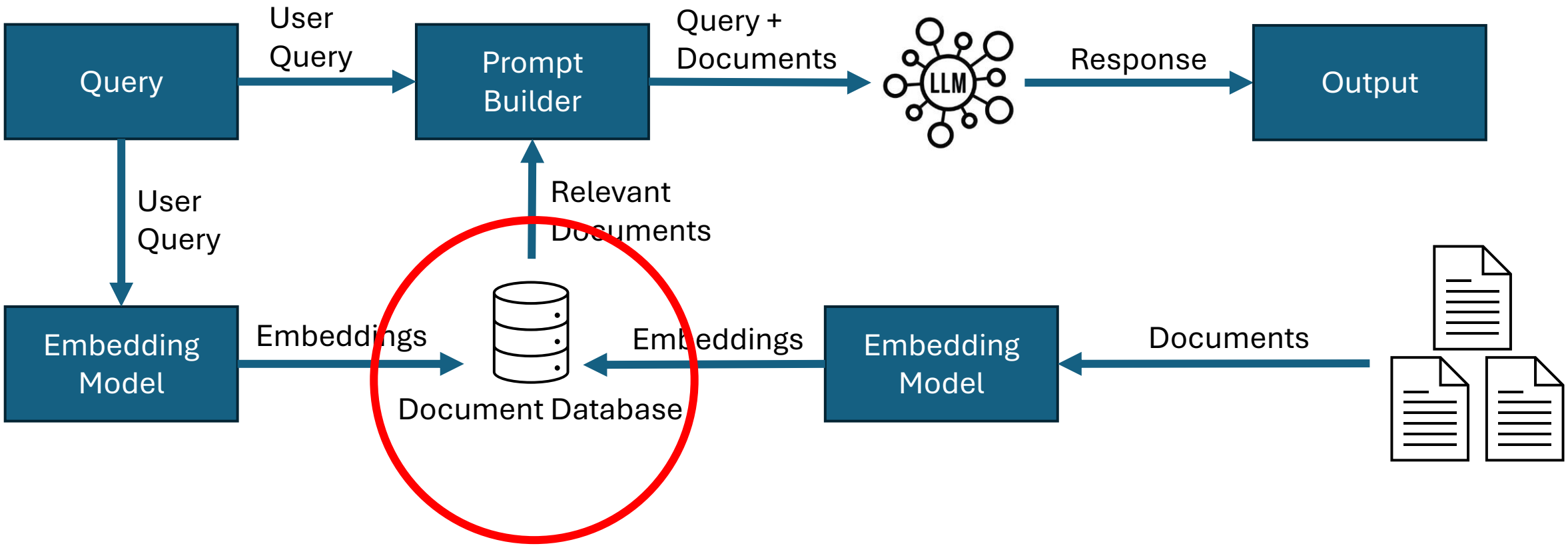
Documents

Document Store

- InMemoryDocumentStore
- Simple document store

```
from haystack.document_stores.in_memory import InMemoryDocumentStore
document_store = InMemoryDocumentStore()
```

Retrieval Augmented Generation



Document Embedder

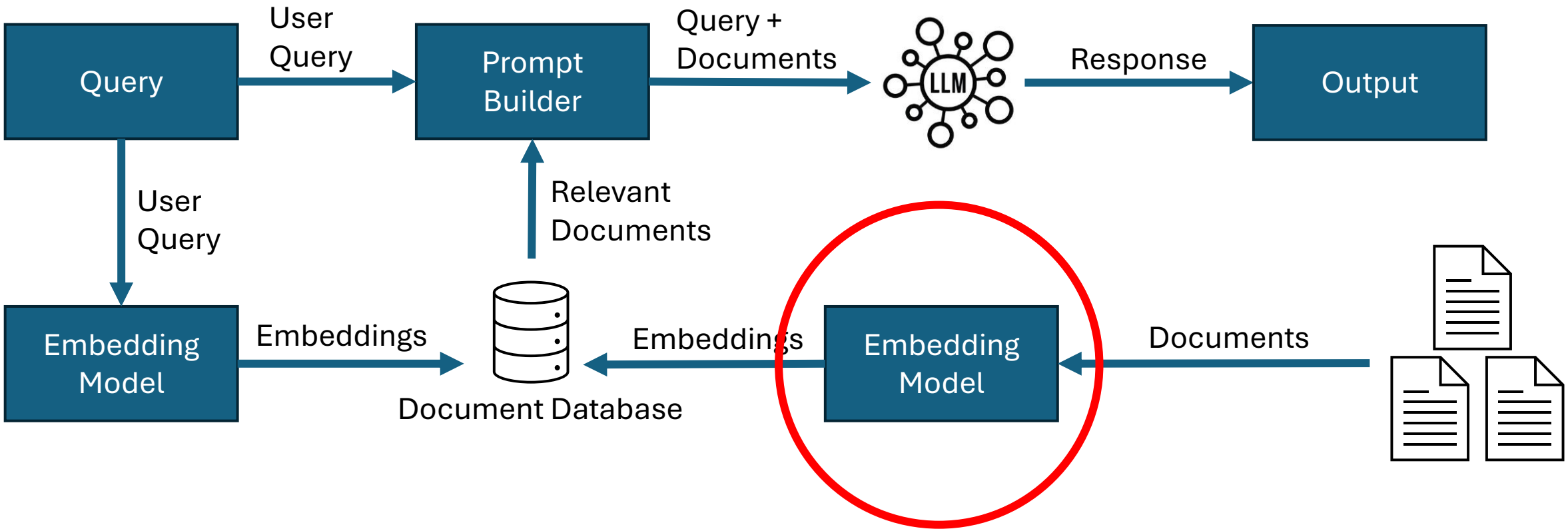
- SentenceTransformers**Document**Embedder
- Computes embeddings for a list of Documents

```
from haystack.components.embedders import SentenceTransformersDocumentEmbedder

doc_embedder = SentenceTransformersDocumentEmbedder(
                        model="sentence-transformers/all-MiniLM-L6-v2")

doc_embedder.warm_up()
```

Retrieval Augmented Generation



Embed Documents and add them to your document store

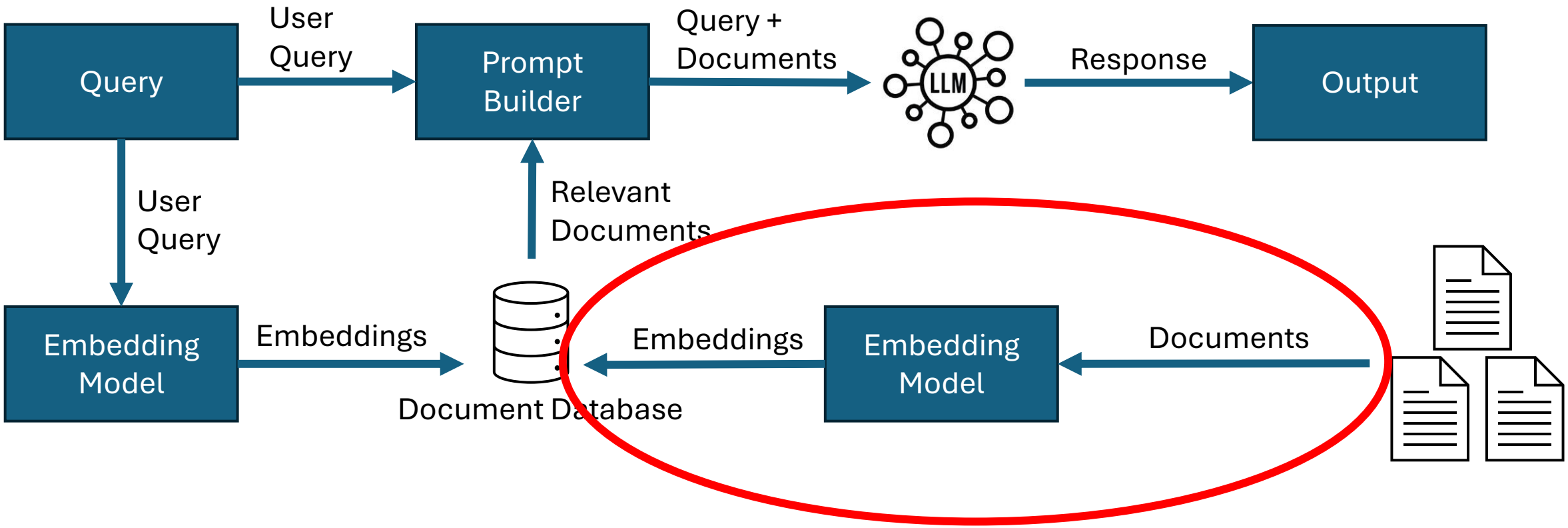
1. Embed documents:

```
docs_with_embeddings = doc_embedder.run(docs)
```

2. Add embedded documents to your document store

```
document_store.write_documents(docs_with_embeddings["documents"])
```

Retrieval Augmented Generation



Main RAG System

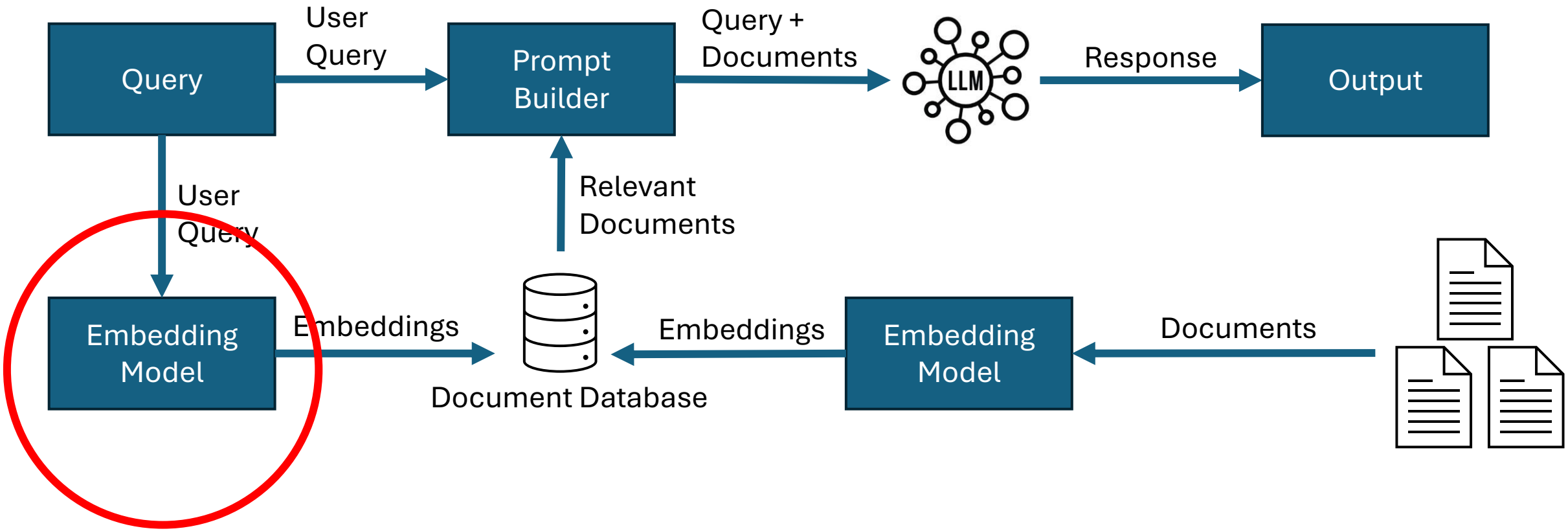
Text Embedder

- SentenceTransformers**Text**Embedder
- Computes an embedding vector for a string

```
from haystack.components.embedders import SentenceTransformersTextEmbedder

text_embedder = SentenceTransformersTextEmbedder(
                    model="sentence-transformers/all-MiniLM-L6-v2")
```

Retrieval Augmented Generation



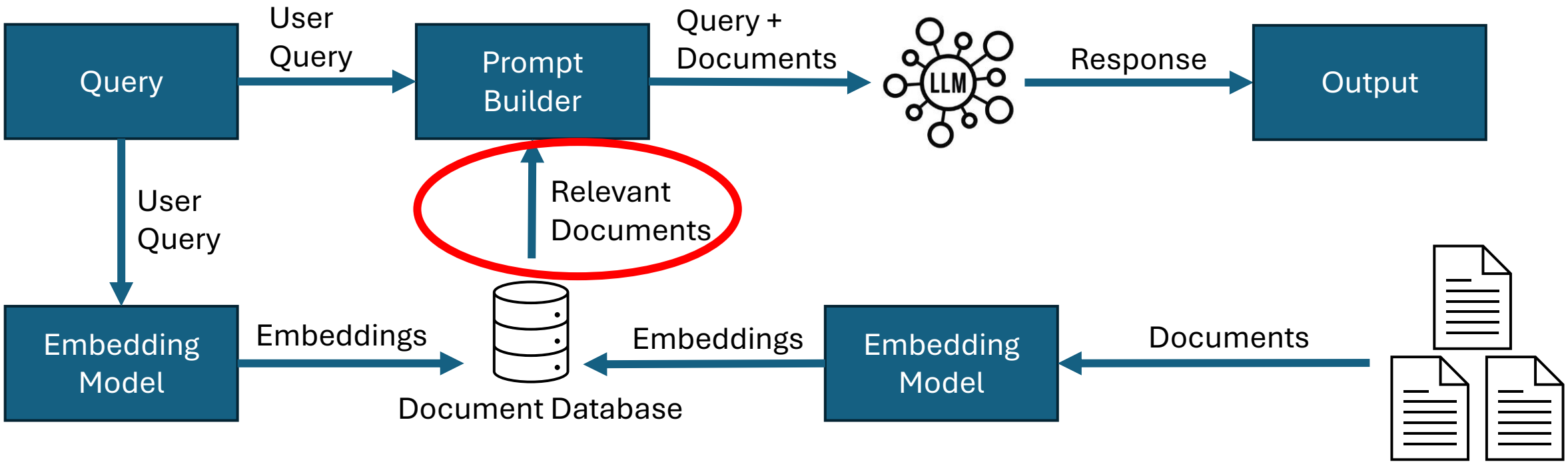
Retriever

- InMemoryEmbeddingRetriever
- For embedding-based retrieval
- Works with the InMemoryDocumentStore

```
from haystack.components.retrievers import InMemoryEmbeddingRetriever

retriever = InMemoryEmbeddingRetriever(
              document_store=document_store,
              top_k = 2)
```

Retrieval Augmented Generation



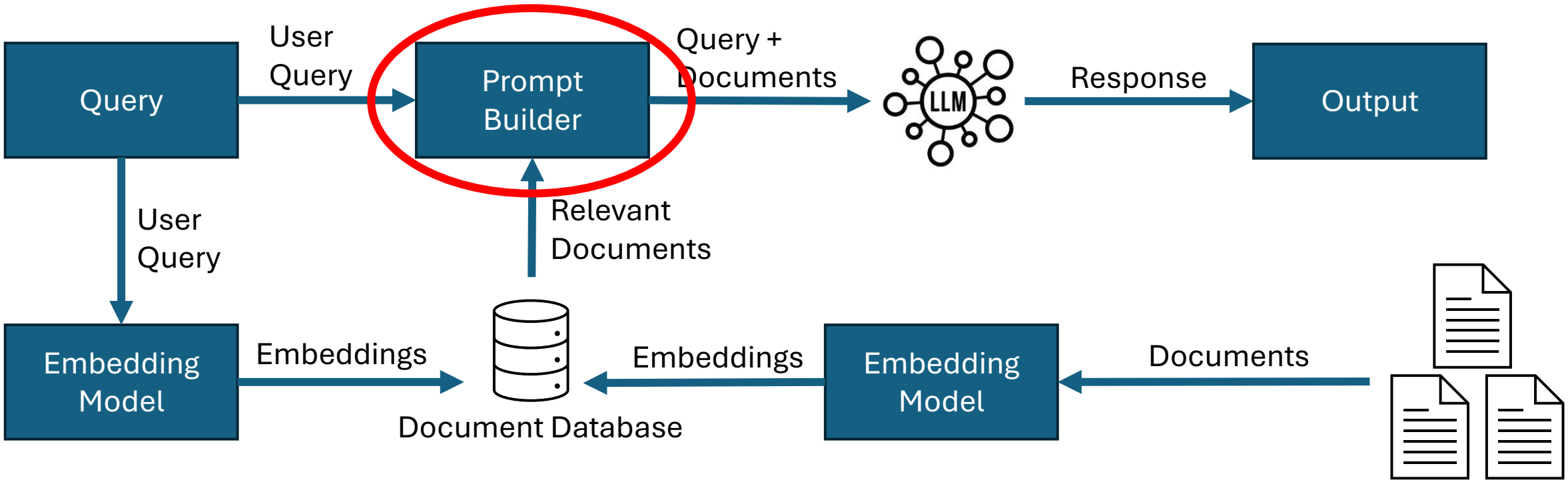
Prompt Builder

- Used before generator to fill variables in a prompt template
- Receives the retrieved documents

```
from haystack.components.builders import PromptBuilder

prompt_builder = PromptBuilder(template=template,
       required_variables=["documents", "question"])
```

Retrieval Augmented Generation



Template

```
template ="""
Given example recipes, design a recipe for the
user's ingredients and give a step-by-step tutorial
on how to cook it as a plain text.
    Example Recipes:
    {% for document in documents %}
     {{ document.content }}
    {% endfor %}

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

Generator

```
from haystack_integrations.components.generators.ollama import
OllamaGenerator

generator = OllamaGenerator(model="llama3.1:8b",
                            url = "http://localhost:11434",
                            timeout = 30*60,
                            generation_kwargs={
                              "num_ctx": 4096,
                              "temperature": 0.9,
                              })
```

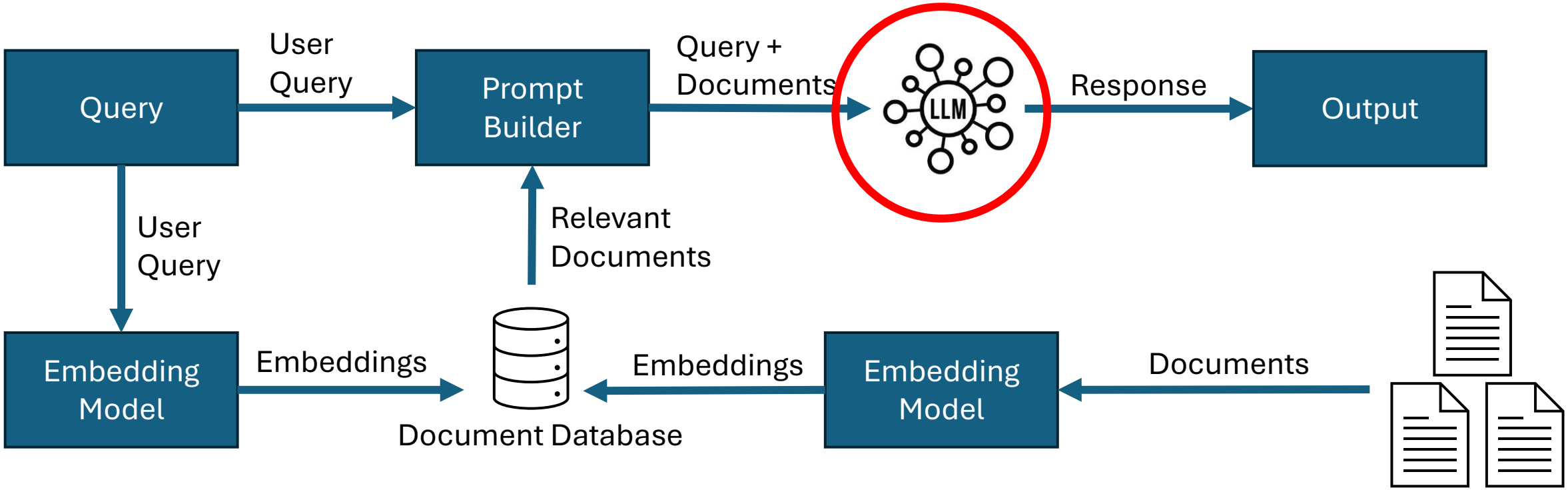
Server IP:

How many seconds until error:

How many tokens:

Determinism (0 is deterministic):

Retrieval Augmented Generation



Pipeline

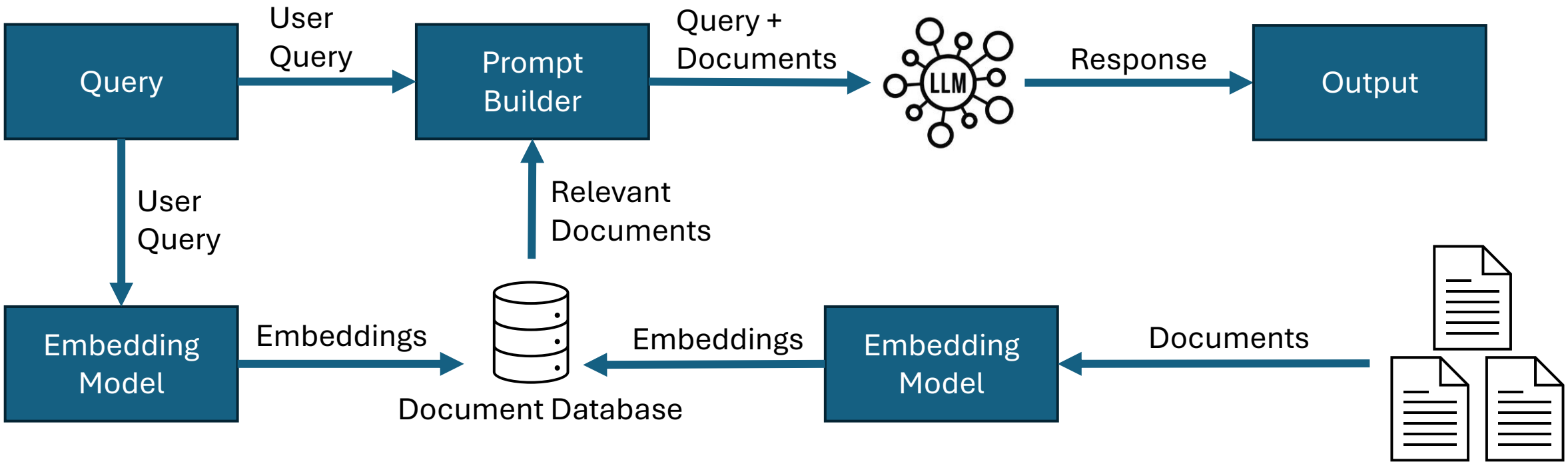
- Used to connect the different components

```
from haystack import Pipeline
basic_rag_pipeline = Pipeline()

basic_rag_pipeline.add_component("text_embedder", text_embedder)
basic_rag_pipeline.add_component("retriever", retriever)
basic_rag_pipeline.add_component("prompt_builder", prompt_builder)
basic_rag_pipeline.add_component("llm", generator)


basic_rag_pipeline.connect("text_embedder.embedding", "retriever.query_embedding")
basic_rag_pipeline.connect("retriever", "prompt_builder.documents")
basic_rag_pipeline.connect("prompt_builder.prompt", "llm.prompt")
```

Retrieval Augmented Generation



Run your Pipeline

- Given a query, run your pipeline by giving your input to:
  - Prompt Builder
  - Text Embedder

```
query = "Describe a vegetarian recipe that contains oranges, carrots and
cookies."

response = basic_rag_pipeline.run(
              {
               "text_embedder": {"text": query},
               "prompt_builder": {"question": query}
              })
```

Additional Infos

Chat Prompt Builder

- Used before generator to fill variables in a prompt template
- Receives the retrieved documents
- More variability with different roles for different types of content

```
system_message = ChatMessage.from_system("You are an assistant helping tourists in {{ language }}.")

user_message = ChatMessage.from_user("What are the best places to visit in {{ city }}?")

assistant_message = ChatMessage.from_assistant("The best places to visit in {{ city }} include the Eiffel Tower, Louvre Museum, and Montmartre.")
```

Chat Generator

- A Chat Prompt Builder is not compatible with a regular generator like the OllamaGenerator
- Change the generator to OllamaChatGenerator and adjust inputs and outputs in your pipeline

Build your own component

- Why? - Sometimes the existing components are not suitable for a task
- Custom Retrieval Function
- Possibility of having two different paths in a pipeline

https://docs.haystack.deepset.ai/docs/custom-components

Example

```
from haystack import component


@component
class WelcomeTextGenerator:
  """
  A component generating personal welcome message and making it upper case
  """
  @component.output_types(welcome_text=str, note=str)
  def run(self, name:str):
    return {"welcome_text": f'Hello {name}, welcome to Haystack!'.upper(), "note": "welcome message is ready"}
```

Example

```
from haystack import component

@component
class WelcomeTextGenerator:
  """
  A component generating personal welcome message and making
it upper case
  """
  @component.output_types(welcome_text=str, note=str)
  def run(self, name:str):
    return {"welcome_text": f'Hello {name}, welcome to
Haystack!'.upper(),
            "note": "welcome message is ready"}
```

Example

```
from haystack import component


@component
class WelcomeTextGenerator:
  """
  A component generating personal welcome message and making
it upper case
  """
  @component.output_types(welcome_text=str, note=str)
  def run(self, name:str):
    return {"welcome_text": f'Hello {name}, welcome to
Haystack!'.upper(),
            "note": "welcome message is ready"}
```

Example

```
from haystack import component


@component
class WelcomeTextGenerator:
  """
  A component generating personal welcome message and making
it upper case
  """
  @component.output_types(welcome_text=str, note=str)
  def run(self, name:str):
    return {"welcome_text": f'Hello {name}, welcome to
Haystack!'.upper(),
            "note": "welcome message is ready"}
```

Example

```
from haystack import component


@component
class WelcomeTextGenerator:
  """
  A component generating personal welcome message and making
it upper case
  """
  @component.output_types(welcome_text=str, note=str)
  def run(self, name:str):
    return {"welcome_text": f'Hello {name}, welcome to
Haystack!'.upper(),
            "note": "welcome message is ready"}
```

Thank you for listening!