



Improve LLM-based Applications with Fallback Mechanisms

Bilge Yücel
Berlin Buzzwords 2024



About me



Bilge Yücel

- 🥑 Developer Relations Engineer at deepset
- 🚒 Open source LLM Framework: Haystack
- 📍 Istanbul, Turkey

X @bilgeycl

in Bilge Yücel

github bilgeyucel

Agenda

- ★ LLMs/RAG
- ★ Introduction to Haystack
- ★ Fallback Mechanisms
- ★ Q&A



Large Language Models (LLMs)

What is Taylor Swift's Tortured Poets Department album about?



There seems to be some confusion, as Taylor Swift does not have an album titled "Tortured Poets Department." This title is associated with the Greek artist Lena Platonos. Taylor Swift, a well-known American singer-songwriter, has released numerous albums, each with its own themes and narratives, but none of them are named "Tortured Poets Department."

If you meant a different album or a specific song by Taylor Swift, please provide more details, and I'd be happy to help!



Retrieval Augmented Generation

Retrieval Augmented Generation

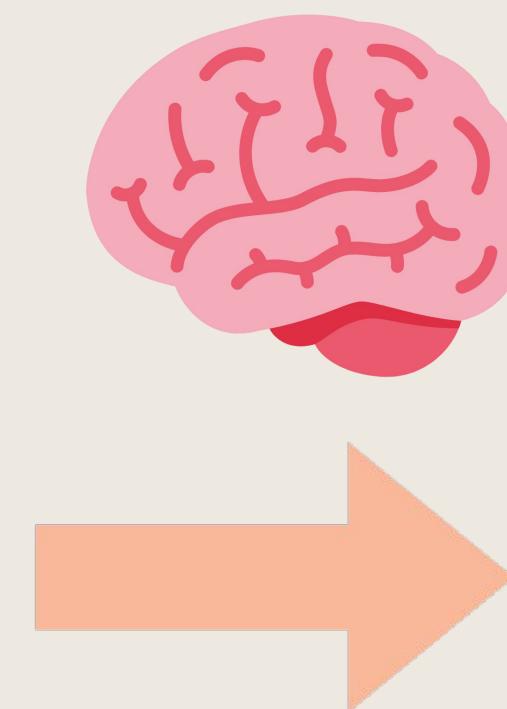


Retrieval Augmented Generation

Answer the following query
given the documents

(Some Relevant
Documents)

What is Tortured Poets
Department album about?

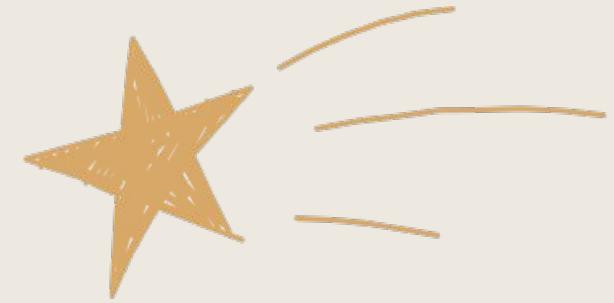


Fans have theorized
that the upcoming
album will detail Swift's
split from actor Joe
Alwyn, whom she dated
for six years before
splitting in April 2023.
Not only do the song
titles...





Haystack
by deepset

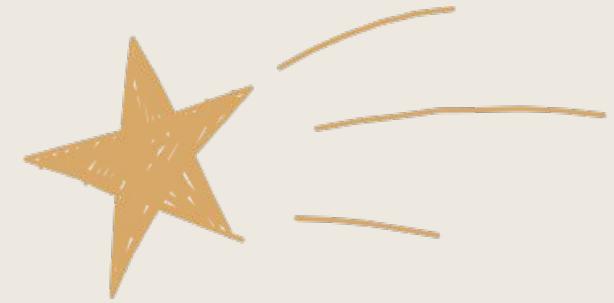


- Haystack is an open source Python framework for building production-ready LLM applications





Haystack
by deepset



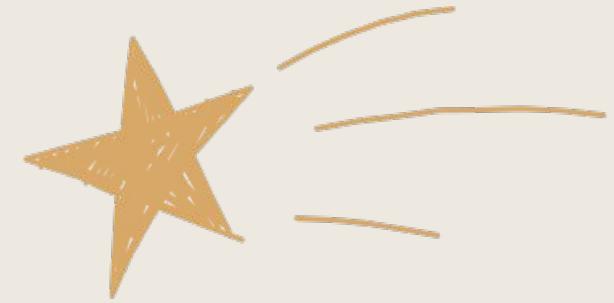
- Haystack is an open source Python framework for building production-ready LLM applications
- Prototyping, evaluation, deployment, monitoring...



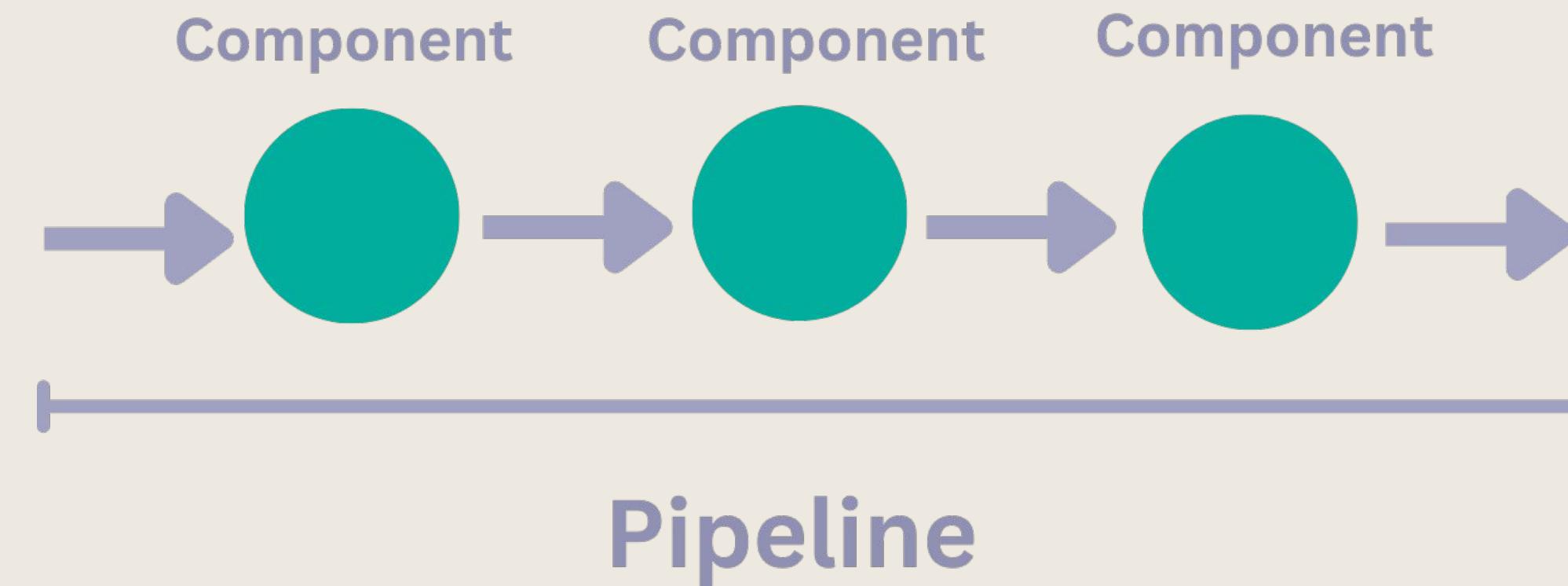


Haystack

by deepset

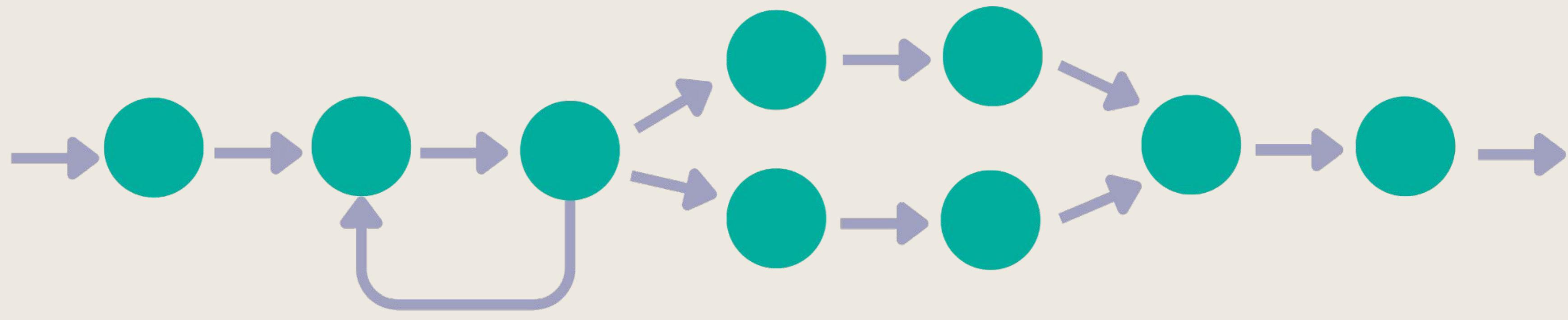


- Haystack is an open source Python framework for building production-ready LLM applications
- Prototyping, evaluation, deployment, monitoring...
- Building blocks: Components & Pipelines



elle

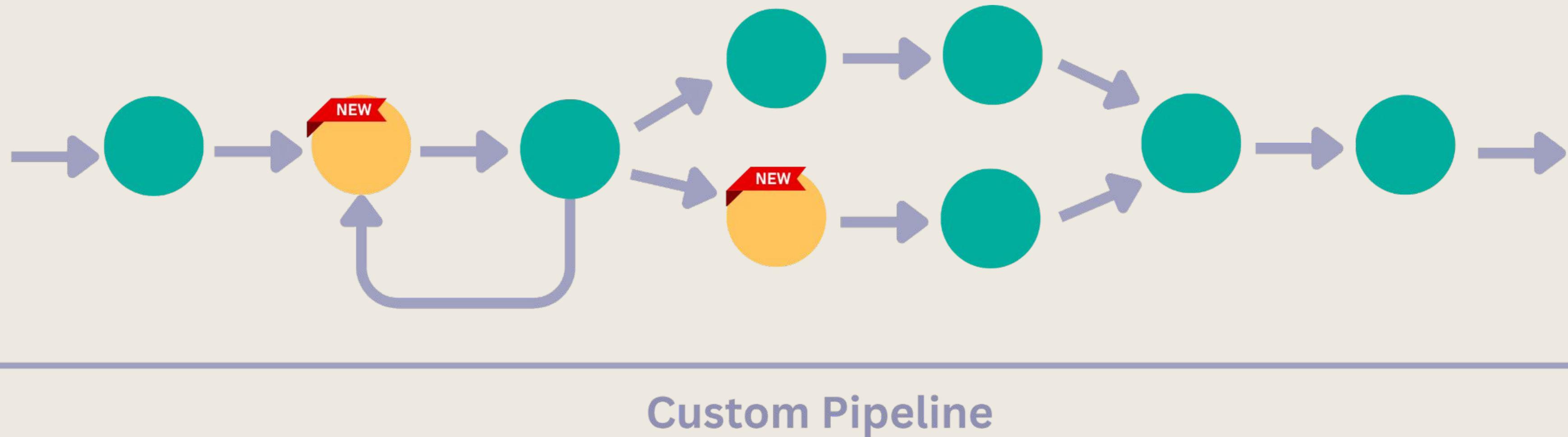
Haystack



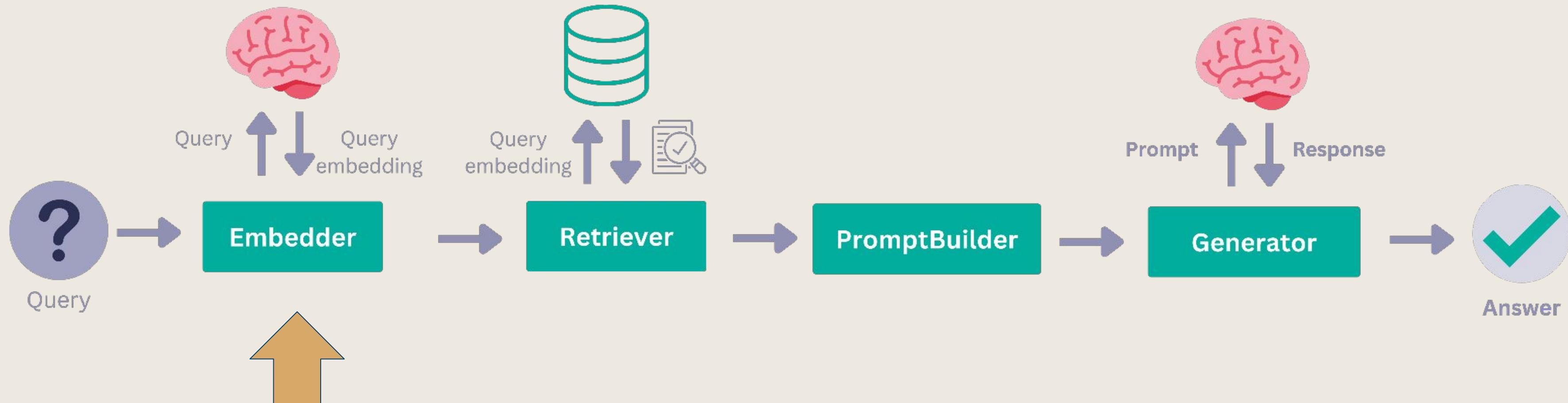
Custom Pipeline

elle

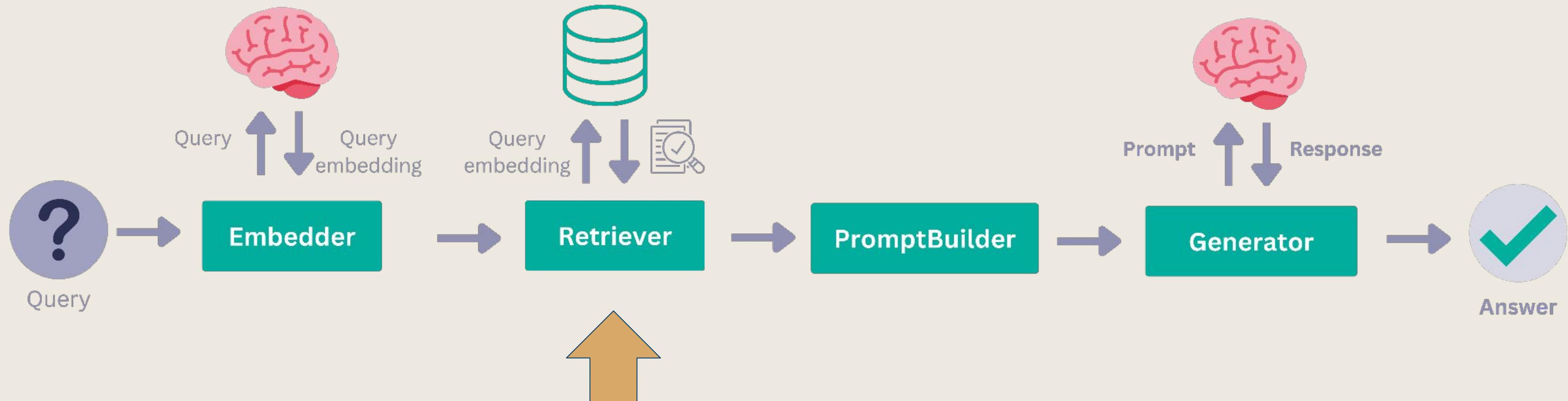
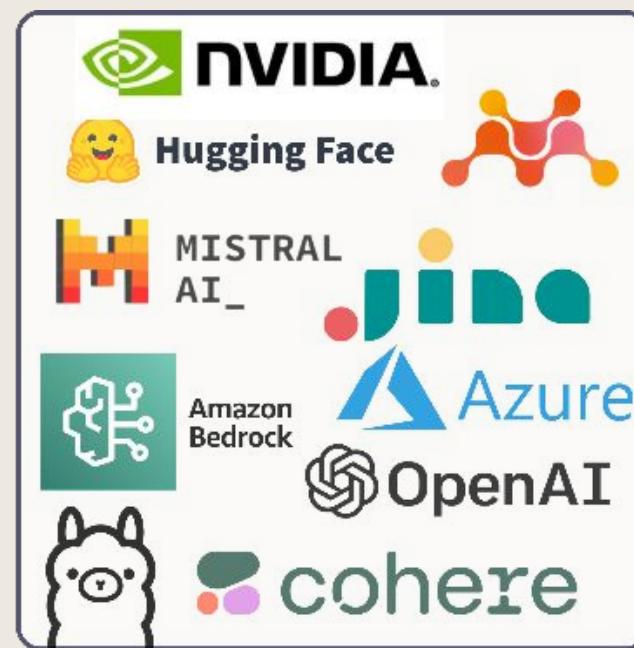
Haystack



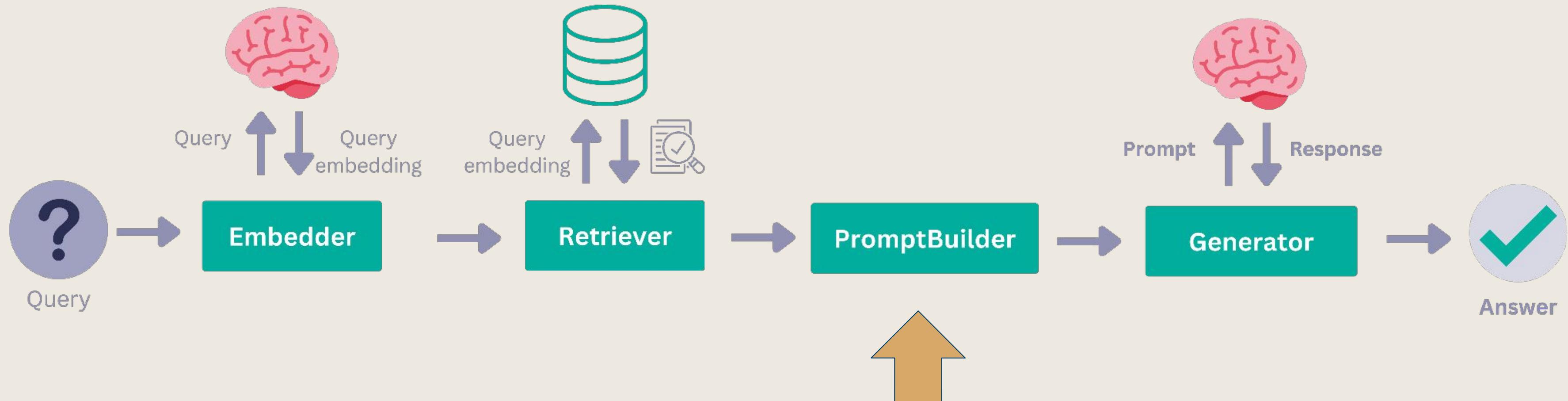
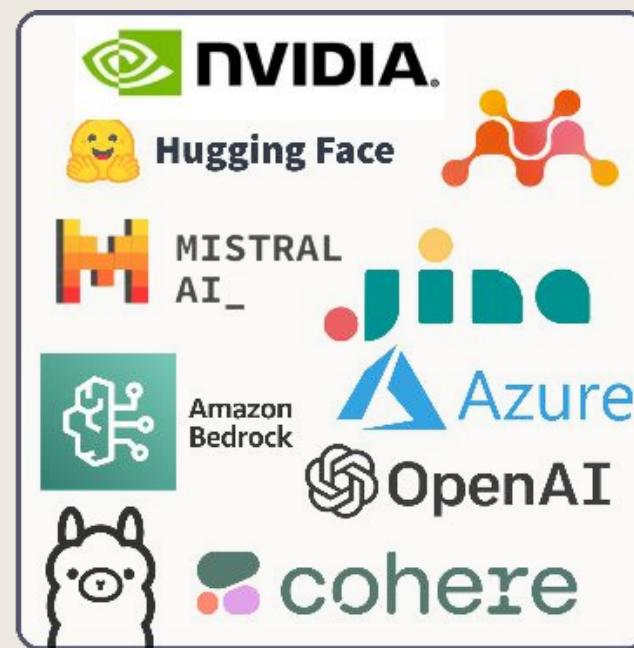
RAG PIPELINE



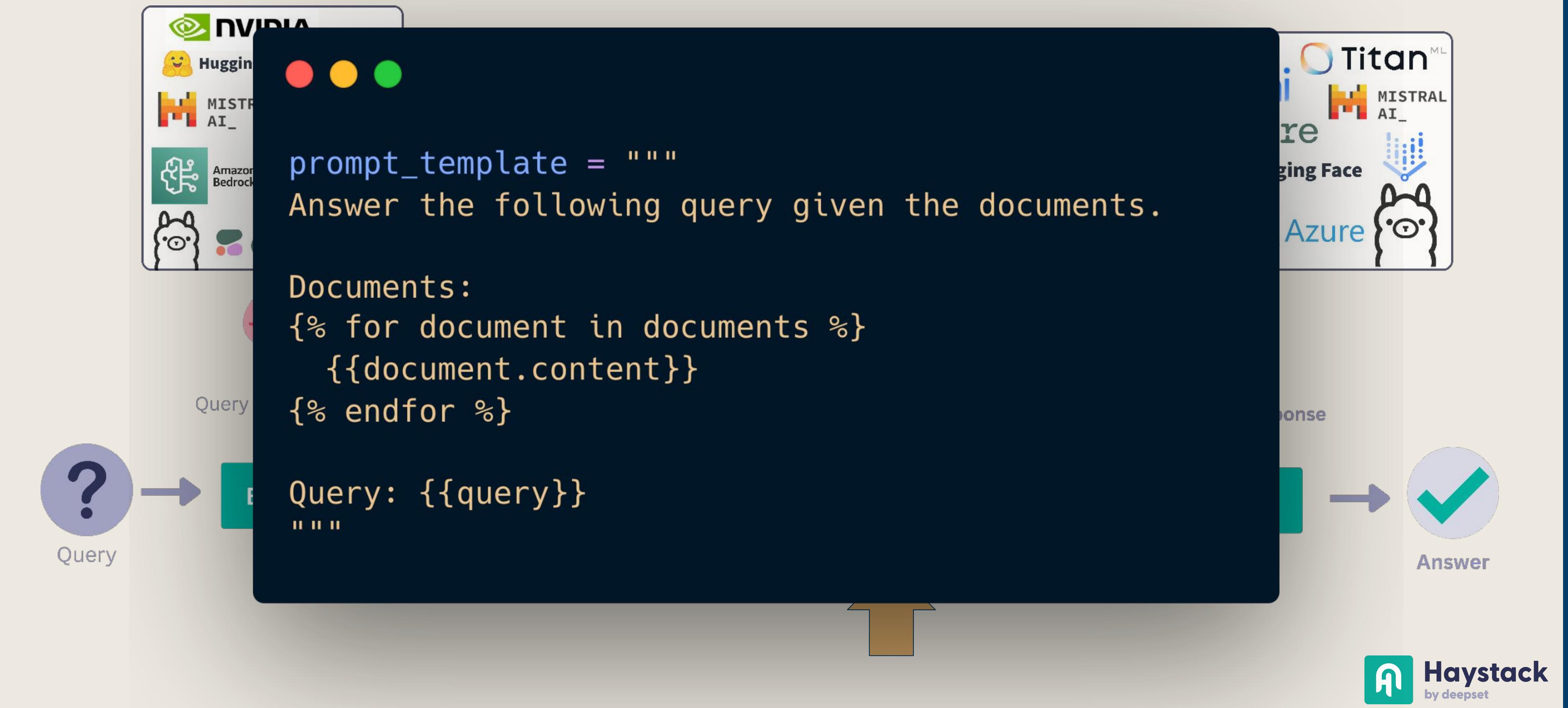
RAG PIPELINE



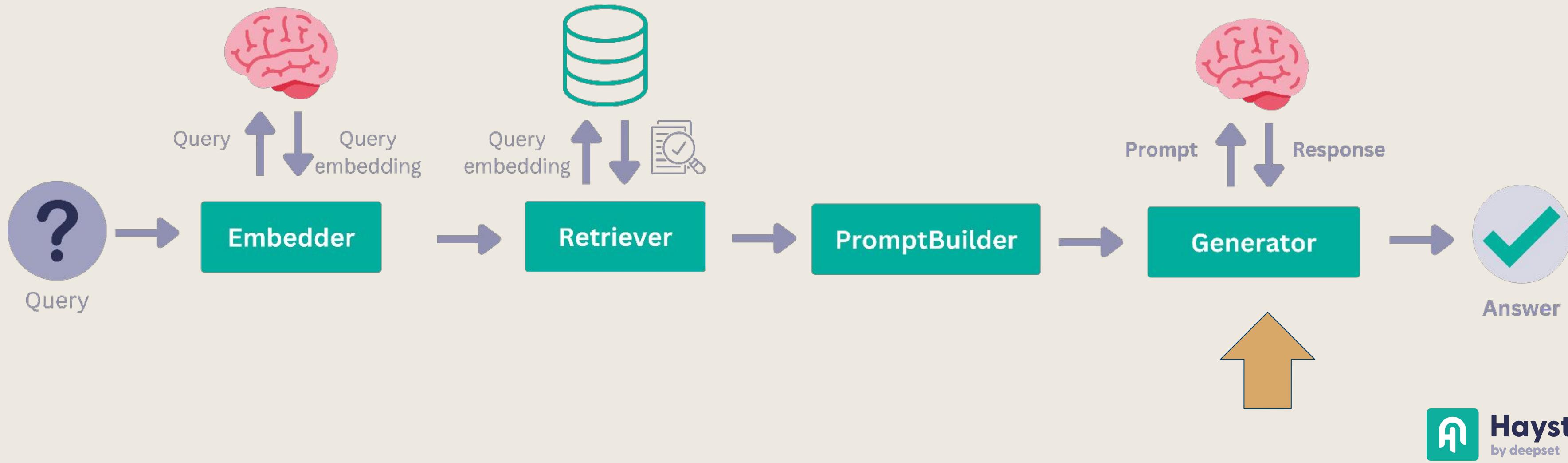
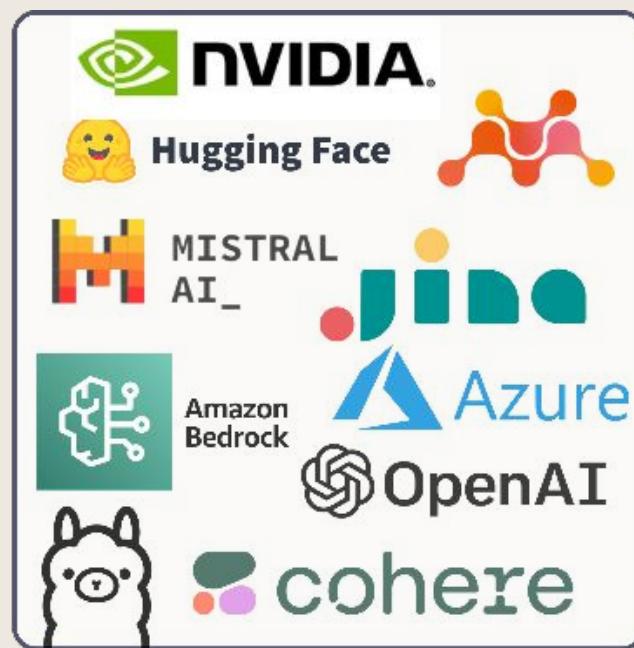
RAG PIPELINE



RAG PIPELINE



RAG PIPELINE



RAG PIPELINE



```
from haystack import Pipeline

rag_pipeline = Pipeline()
rag_pipeline.add_component("embedder", JinaTextEmbedder(model= "jina-embeddings-v2-base-en"))
rag_pipeline.add_component("retriever", QdrantEmbeddingRetriever(document_store=document_store, top_k=5))
rag_pipeline.add_component("prompt_builder", PromptBuilder(template=prompt_template))
rag_pipeline.add_component("generator", HuggingFaceAPIGenerator(api_type="serverless_inference_api",
                                                               api_params={"model": "mistralai/Mistral-7B-Instruct-v0.3"}))

rag_pipeline.connect("text_embedder.embedding", "retriever.query_embedding")
rag_pipeline.connect("retriever", "prompt_builder.documents")
rag_pipeline.connect("prompt_builder", "generator")

query="What's Tortured Poets Department album about?"

rag_pipeline.run({"text_embedder":{"text": query}, "prompt_builder":{"query": query}})

>>> Fans have theorized that the upcoming album will detail Swift's split from actor Joe Alwyn, whom she dated for six years before splitting in April 2023. Not only do the song titles include references to the English actor and their former relationship, but Swift recently teased that the album is about heartbreak as she shared five exclusive playlists with Apple Music.\nAccording to a press release, with each playlist, Swift has chosen songs from her own catalog that fit each stage of a breakup: denial, anger, bargaining, depression and acceptance.
```





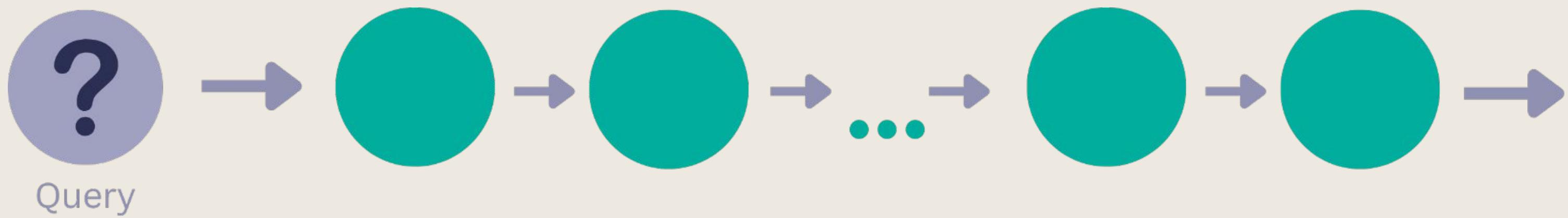
Fallback Mechanisms

Use Cases

- ★ RAG with Web Fallback
- ★ Early Routing with Fallback
- ★ Extracting Structured Data

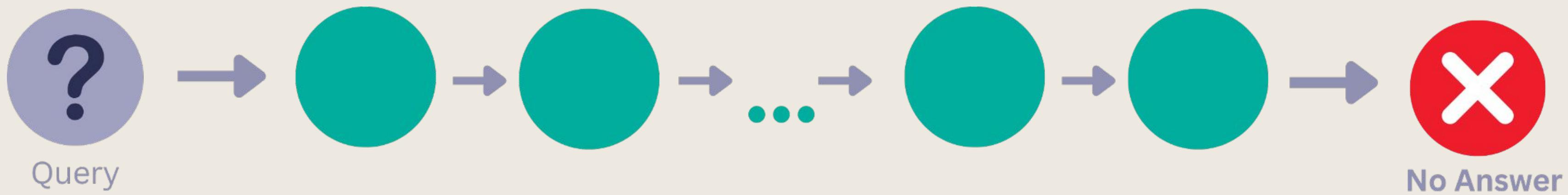


RAG PIPELINE



*When is Taylor Swift's
next concert?*

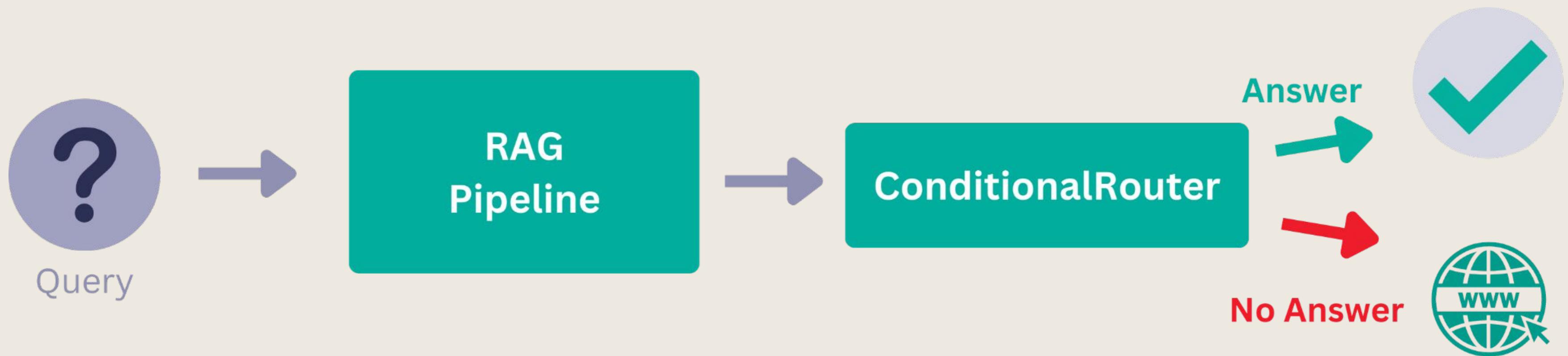
RAG PIPELINE



*When is Taylor Swift's
next concert?*

*The documents do not
provide any
information about
Taylor Swift's
upcoming concert.*

Mechanism: Routing



Routing - Prompt Template



```
prompt_template = """
```

Answer the following query given the documents.

If the answer is not contained within the documents reply with 'no_answer'

Documents:

```
{% for document in documents %}  
    {{document.content}}  
{% endfor %}
```

Query: {{query}}

```
"""
```

Routing - Conditional Router

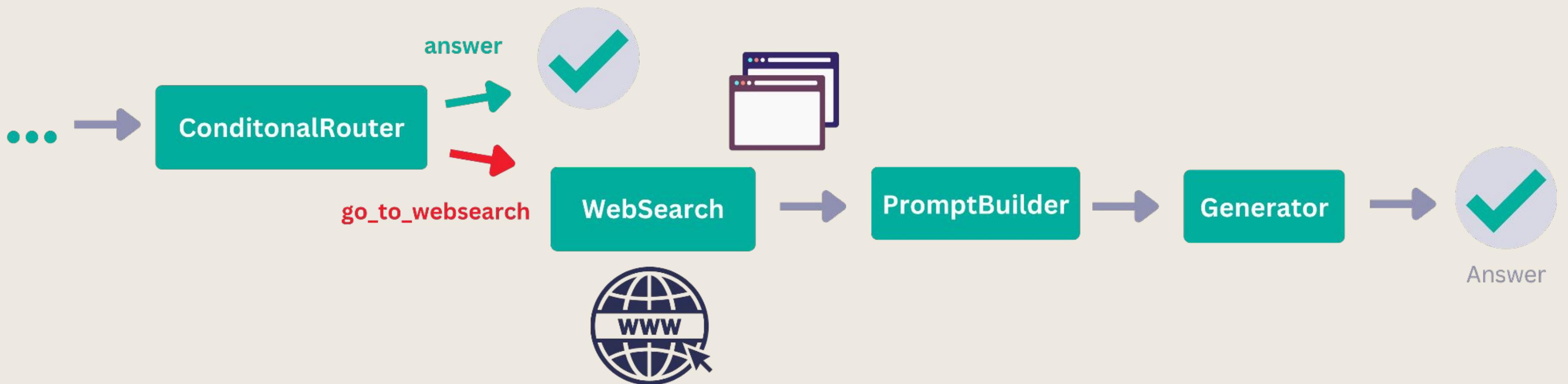
```
● ● ●

from haystack.components.routers import ConditionalRouter

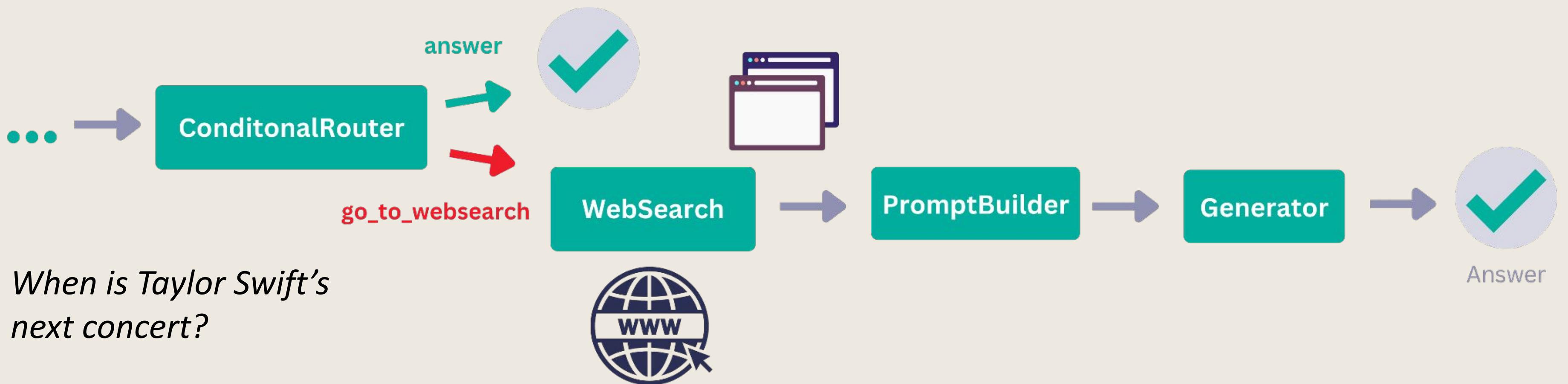
routes = [
    {
        "condition": "{{'no_answer' in replies[0]}}",
        "output": "{{query}}",
        "output_name": "go_to_websearch",
        "output_type": str,
    },
    {
        "condition": "{{'no_answer' not in replies[0]}}",
        "output": "{{replies[0]}}",
        "output_name": "answer",
        "output_type": str,
    },
]

router = ConditionalRouter(routes)
```

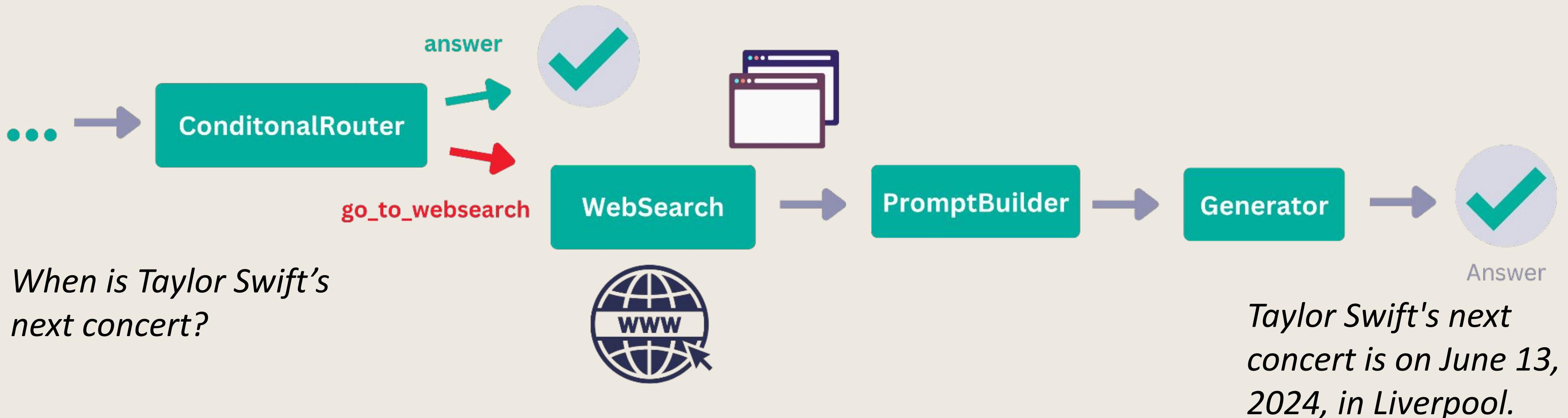
Fallback to Web



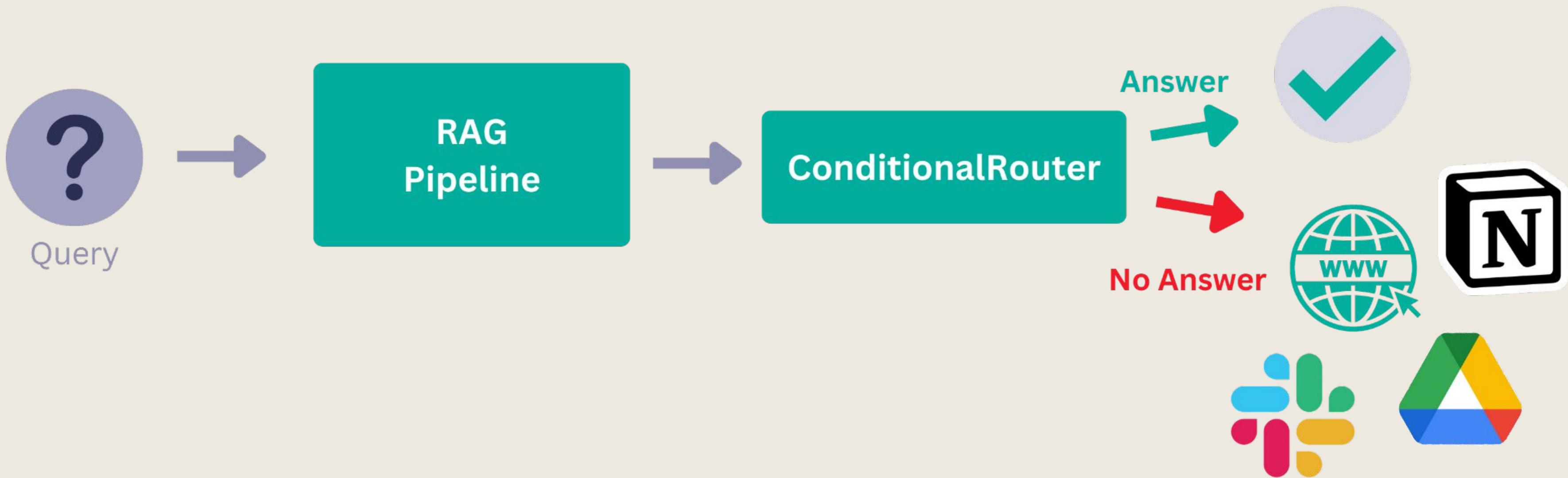
Fallback to Web



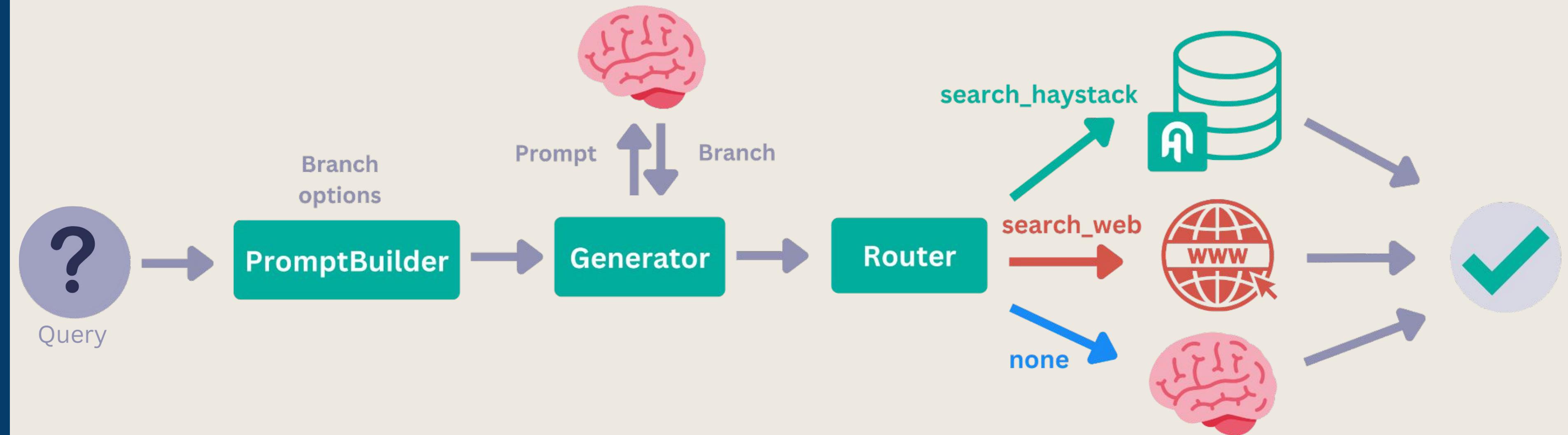
Fallback to Web



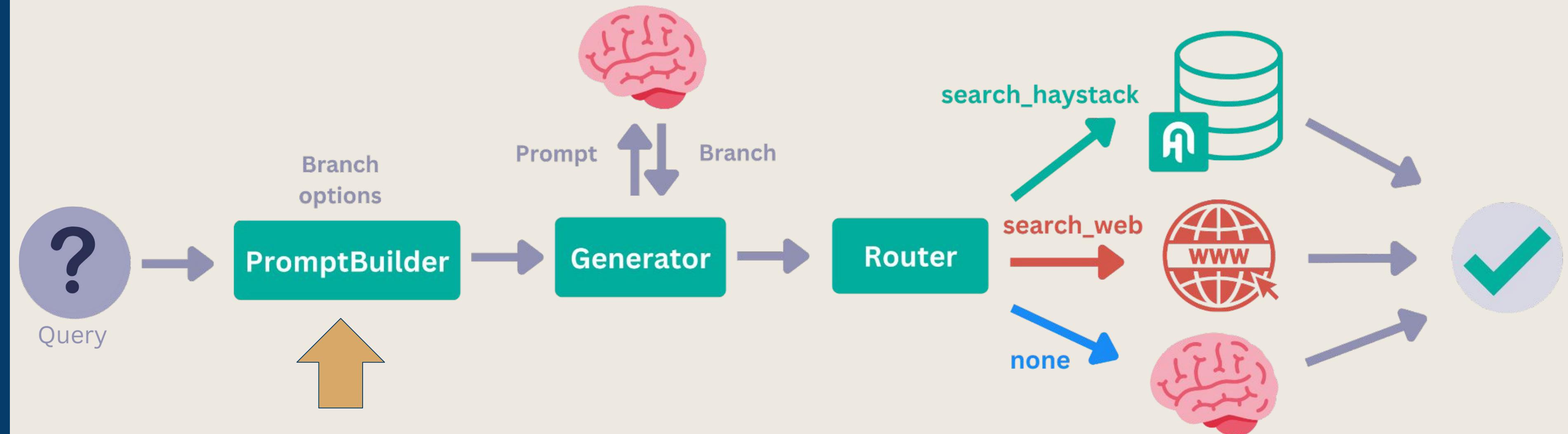
Fallback to Other Sources



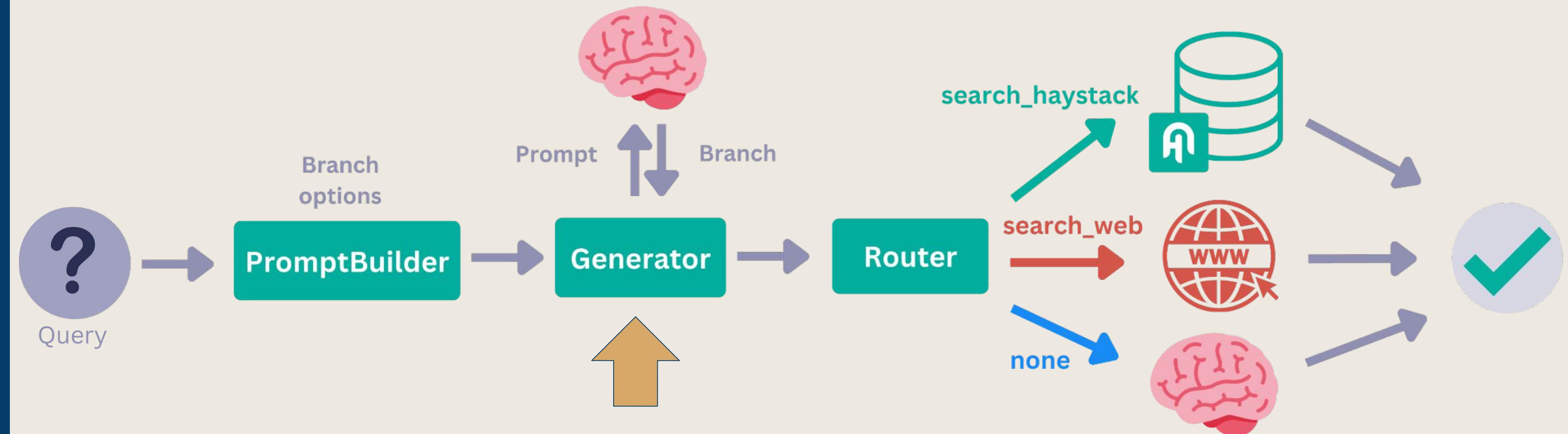
Early Routing



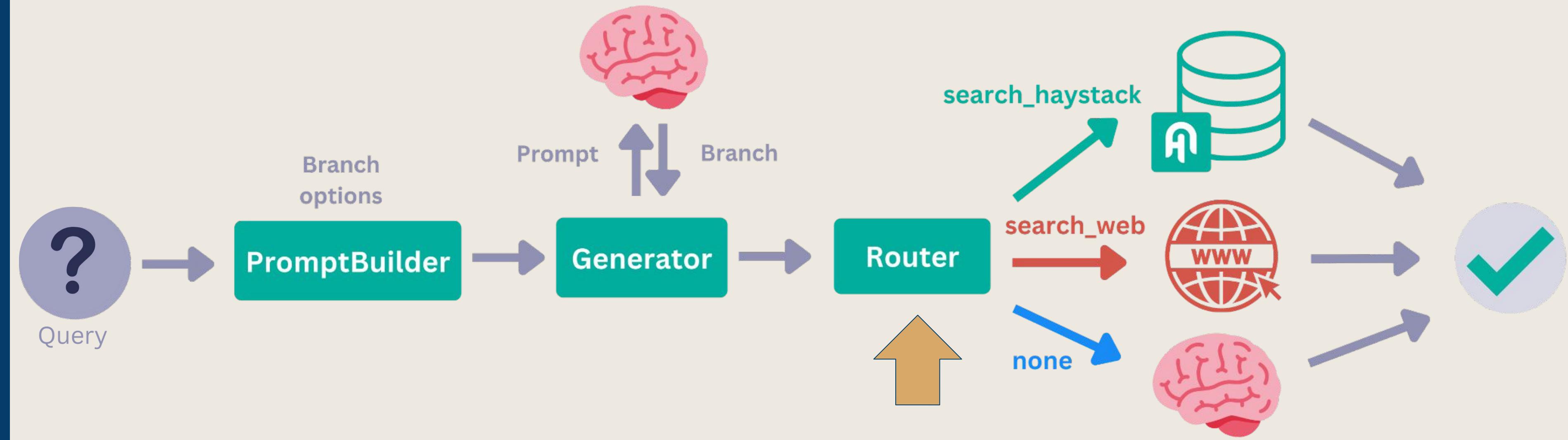
Early Routing



Early Routing



Early Routing (Tools)



● ○ ●

```
prompt_template = ""
```

You are a virtual assistant, equipped with the following tools:

- `{"tool_name": "search_web", "tool_description": "Access to Google search, use this tool whenever information on recent events is needed"}`
- `{"tool_name": "search_haystack", "tool_description": "Access to Haystack documentation, use this tool whenever information on building with LLMs, custom AI applications, Haystack, the open source LLM framework, is needed"}`

Select the most appropriate tool to resolve the user's query. Respond in JSON format, specifying the user request and the chosen tool for the response.

If you can't match user query to an above listed tools, respond with `none`.

#####

Here are some examples:

```
```json
{
 "query": "Why did Elon Musk recently sue OpenAI?",
 "response": "search_web"
}
{
 "query": "What are the init parameters of OpenAIGenerator component?",
 "response": "search_haystack"
}
{
 "query": "I like elephants, which kind of elephants exist?",
 "response": "none"
}
{
 "query": "What's the capital of Germany?",
 "response": "none"
}
```

```

Choose the best tool (or none) for each user request, considering the current context of the conversation specified above.

```
{"query": {{query}}, "response": }
"""
```

```
● ○ ●
```

```
prompt_template = ""
```

```
You are a virtual assistant, equipped with the following tools:
```

- `{"tool_name": "search_web", "tool_description": "Access to Google search, use this tool whenever information on recent events is needed"}`
- `{"tool_name": "search_haystack", "tool_description": "Access to Haystack documentation, use this tool whenever information on building with LLMs, custom AI applications, Haystack, the open source LLM framework, is needed"}`

```
Select the most appropriate tool to resolve the user's query. Respond in JSON format, specifying the user request and the chosen tool for the response.
```

```
If you can't match user query to an above listed tools, respond with `none`.
```

```
#####
```

```
Here are some examples:
```

```
```json
{
 "query": "Why did Elon Musk recently sue OpenAI?",
 "response": "search_web"
}
{
 "query": "What are the init parameters of OpenAIGenerator component?",
 "response": "search_haystack"
}
{
 "query": "I like elephants, which kind of elephants exist?",
 "response": "none"
}
{
 "query": "What's the capital of Germany?",
 "response": "none"
}
```

```

```
Choose the best tool (or none) for each user request, considering the current context of the conversation specified above.
```

```
{"query": {{query}}, "response": }
"""
```

● ○ ●

```
prompt_template = ""
```

You are a virtual assistant, equipped with the following tools:

- `{"tool_name": "search_web", "tool_description": "Access to Google search, use this tool whenever information on recent events is needed"}`
- `{"tool_name": "search_haystack", "tool_description": "Access to Haystack documentation, use this tool whenever information on building with LLMs, custom AI applications, Haystack, the open source LLM framework, is needed"}`

Select the most appropriate tool to resolve the user's query. Respond in JSON format, specifying the user request and the chosen tool for the response.

If you can't match user query to an above listed tools, respond with `none`.

```
#####
```

Here are some examples:

```
```json
{
 "query": "Why did Elon Musk recently sue OpenAI?",
 "response": "search_web"
}
{
 "query": "What are the init parameters of OpenAIGenerator component?",
 "response": "search_haystack"
}
{
 "query": "I like elephants, which kind of elephants exist?",
 "response": "none"
}
{
 "query": "What's the capital of Germany?",
 "response": "none"
}
```

```

Choose the best tool (or none) for each user request, considering the current context of the conversation specified above.

```
{"query": {{query}}, "response": }
"""
```

● ○ ●

```
prompt_template = ""
```

You are a virtual assistant, equipped with the following tools:

- `{"tool_name": "search_web", "tool_description": "Access to Google search, use this tool whenever information on recent events is needed"}`
- `{"tool_name": "search_haystack", "tool_description": "Access to Haystack documentation, use this tool whenever information on building with LLMs, custom AI applications, Haystack, the open source LLM framework, is needed"}`

Select the most appropriate tool to resolve the user's query. Respond in JSON format, specifying the user request and the chosen tool for the response.

If you can't match user query to an above listed tools, respond with `none`.

```
#####
```

Here are some examples:

```
```json
{
 "query": "Why did Elon Musk recently sue OpenAI?",
 "response": "search_web"
}
{
 "query": "What are the init parameters of OpenAIGenerator component?",
 "response": "search_haystack"
}
{
 "query": "I like elephants, which kind of elephants exist?",
 "response": "none"
}
{
 "query": "What's the capital of Germany?",
 "response": "none"
}
```

```

Choose the best tool (or none) for each user request, considering the current context of the conversation specified above.

```
{"query": {{query}}, "response": }
"""
```

● ○ ●

```
prompt_template = ""
```

You are a virtual assistant, equipped with the following tools:

- `{"tool_name": "search_web", "tool_description": "Access to Google search, use this tool whenever information on recent events is needed"}`
- `{"tool_name": "search_haystack", "tool_description": "Access to Haystack documentation, use this tool whenever information on building with LLMs, custom AI applications, Haystack, the open source LLM framework, is needed"}`

Select the most appropriate tool to resolve the user's query. Respond in JSON format, specifying the user request and the chosen tool for the response.

If you can't match user query to an above listed tools, respond with `none`.

#####

Here are some examples:

```
```json
{
 "query": "Why did Elon Musk recently sue OpenAI?",
 "response": "search_web"
}
{
 "query": "What are the init parameters of OpenAIGenerator component?",
 "response": "search_haystack"
}
{
 "query": "I like elephants, which kind of elephants exist?",
 "response": "none"
}
{
 "query": "What's the capital of Germany?",
 "response": "none"
}
```

```

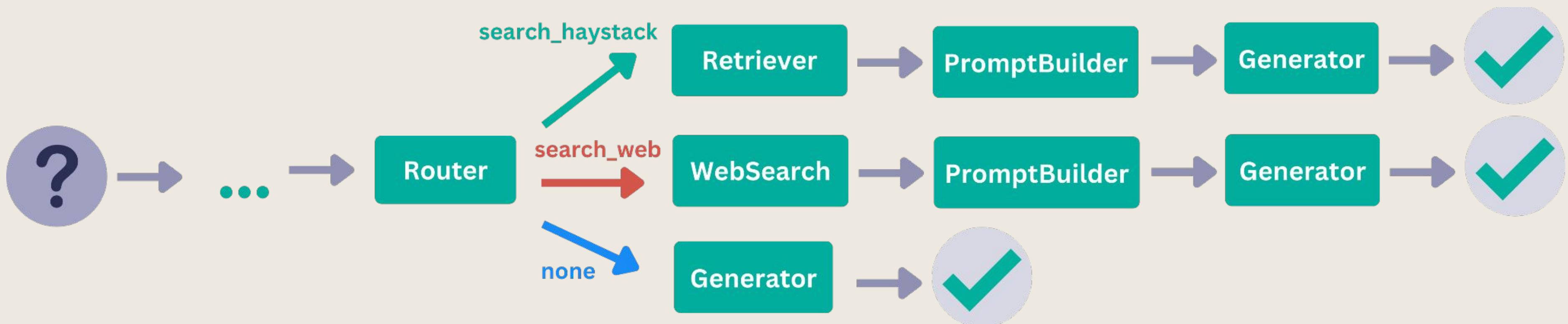
Choose the best tool (or none) for each user request, considering the current context of the conversation specified above.

```
{"query": {{query}}, "response": }
"""
```

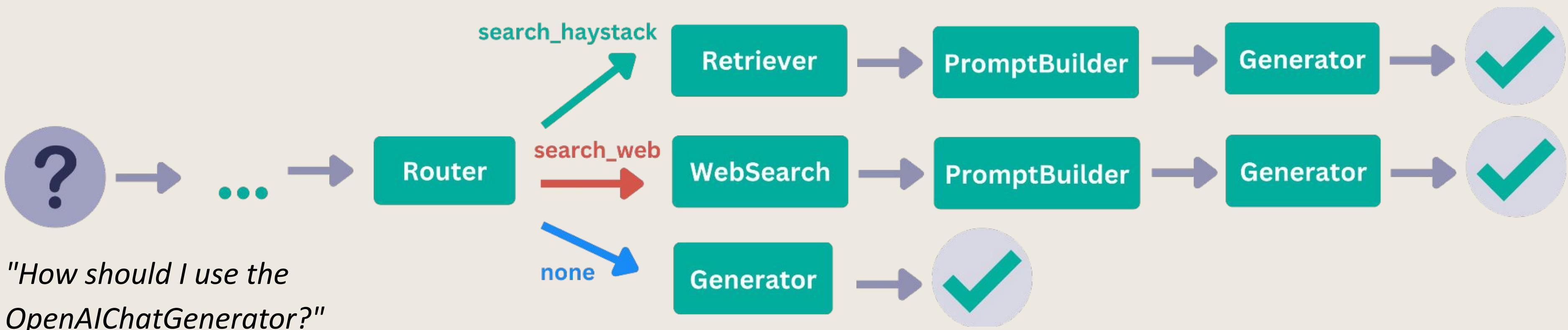
Early Routing - Conditional Router

```
● ● ●  
main_routes = [  
    {  
        "condition": "{{'search_haystack' in replies[0]}}",  
        "output": "{{query}}",  
        "output_name": "search_haystack",  
        "output_type": str,  
    },  
    {  
        "condition": "{{'search_web' in replies[0]}}",  
        "output": "{{query}}",  
        "output_name": "search_web",  
        "output_type": str,  
    },  
    {  
        "condition": "{{'none' in replies[0]}}",  
        "output": "{{query}}",  
        "output_name": "none",  
        "output_type": str,  
    }  
]
```

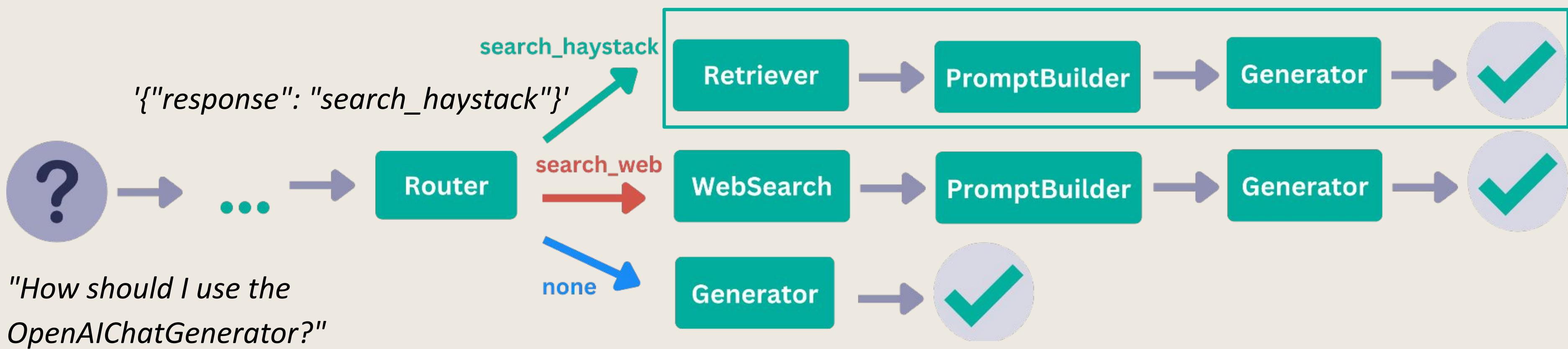
Early Routing (Tools)



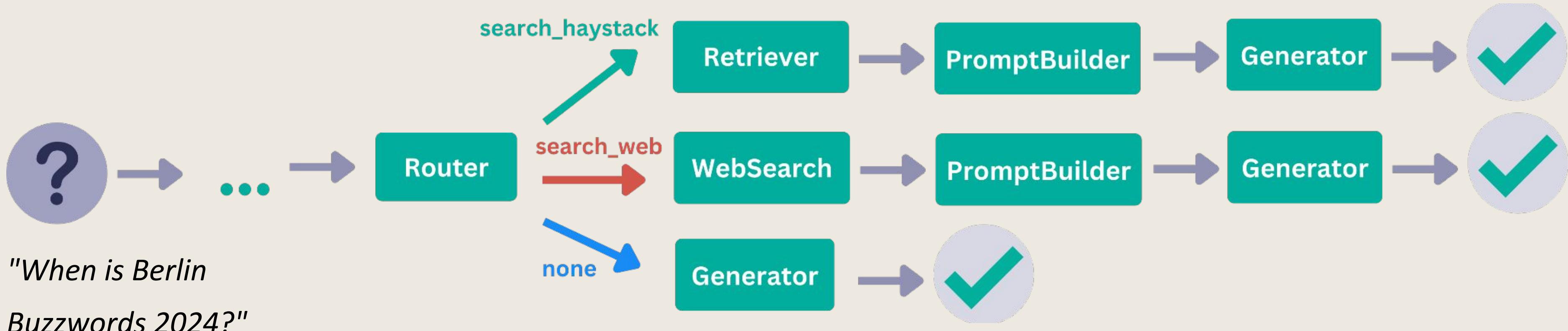
Early Routing (Tools)



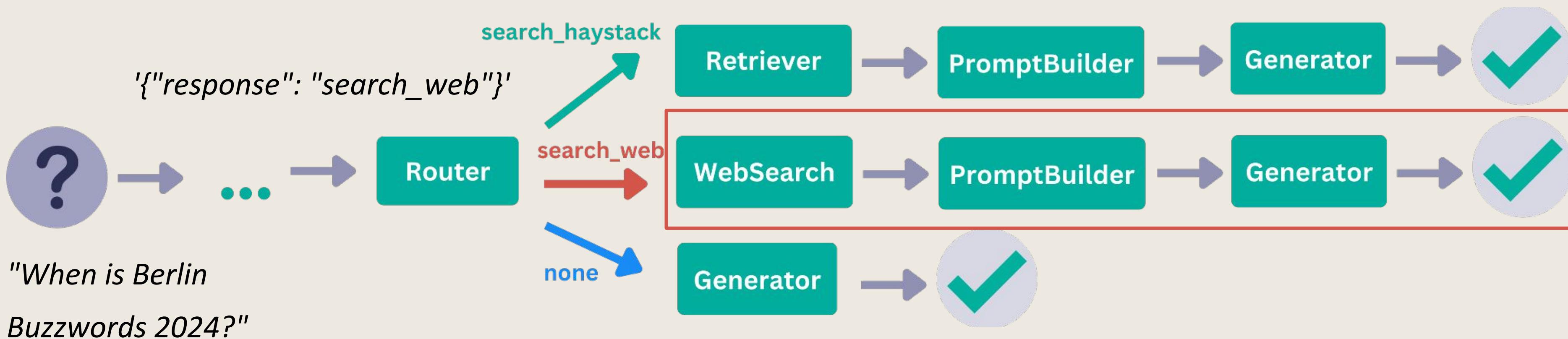
Early Routing (Tools)



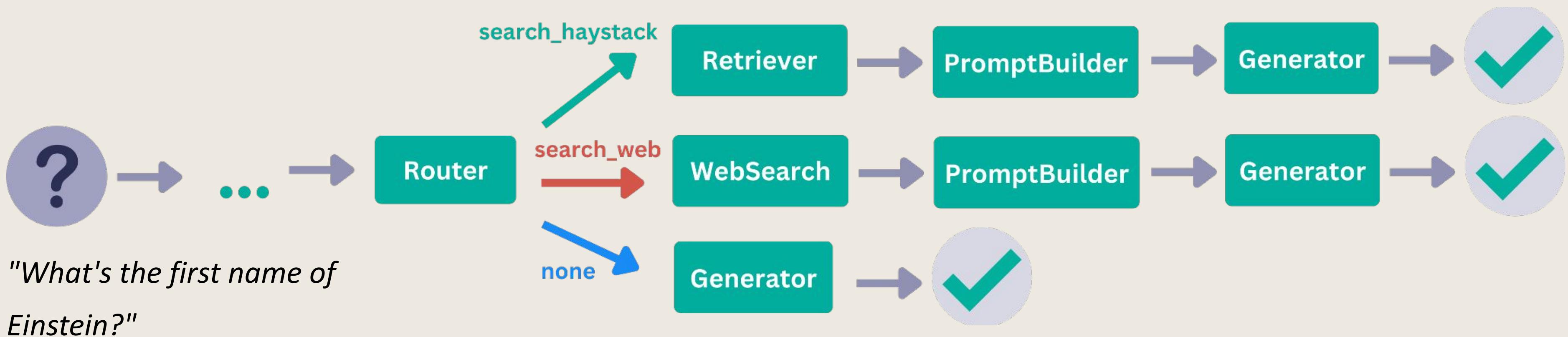
Early Routing (Tools)



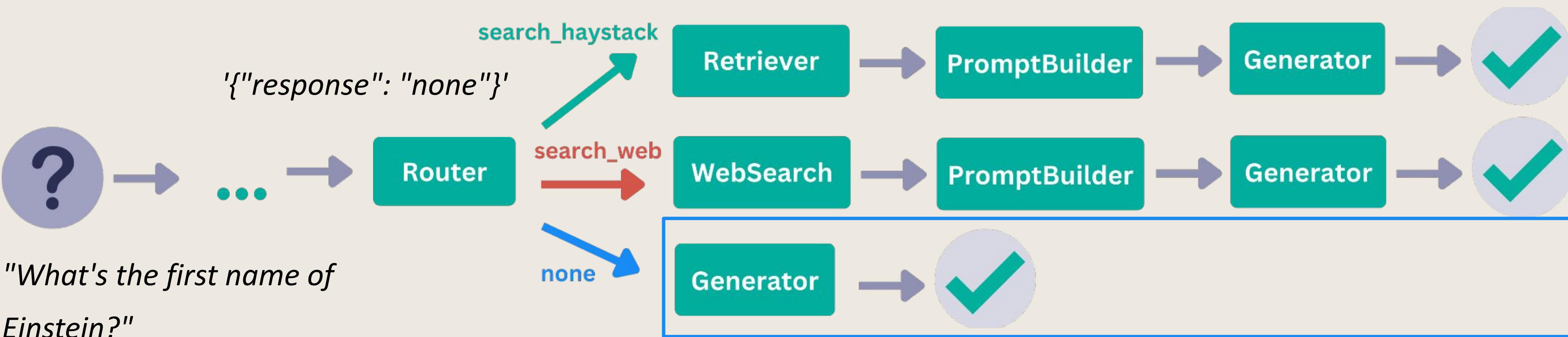
Early Routing (Tools)



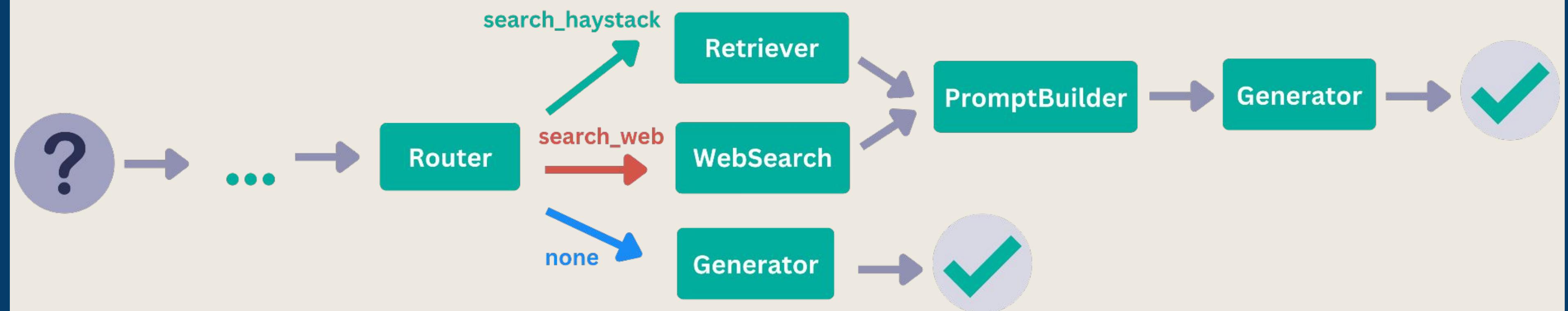
Early Routing (Tools)



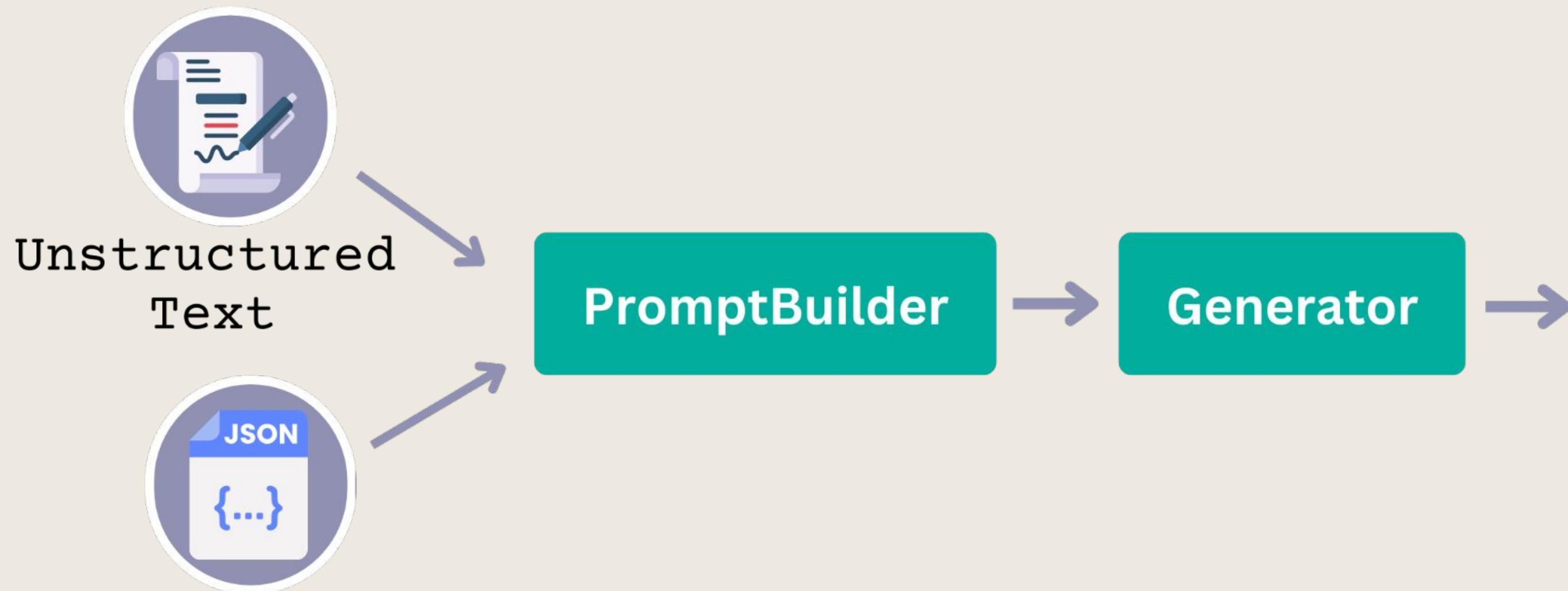
Early Routing (Tools)



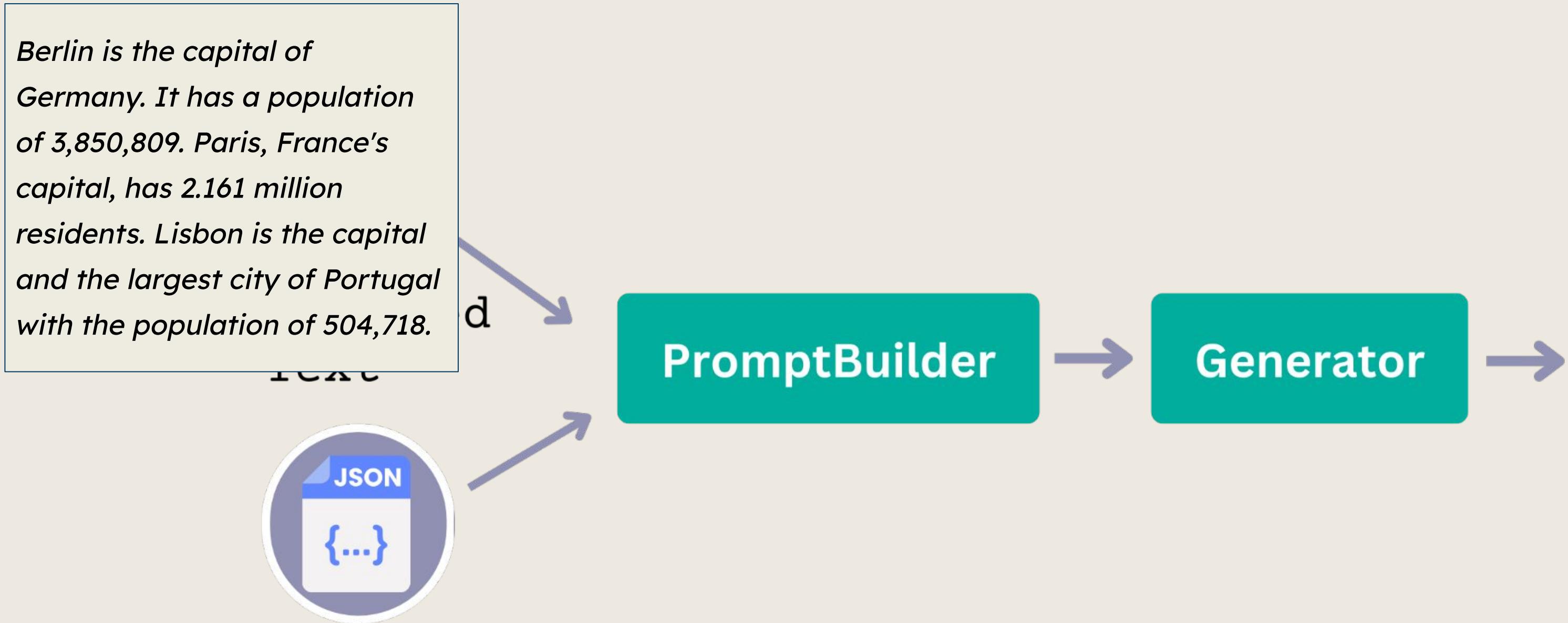
Early Routing (Tools)



Extracting Structured Data



Extracting Structured Data



Extracting Structured Data

Berlin is the capital of Germany. It has a population of 3,850,809. Paris, France's capital, has 2.161 million residents. Lisbon is the capital and the largest city of Portugal with the population of 504,718.

text

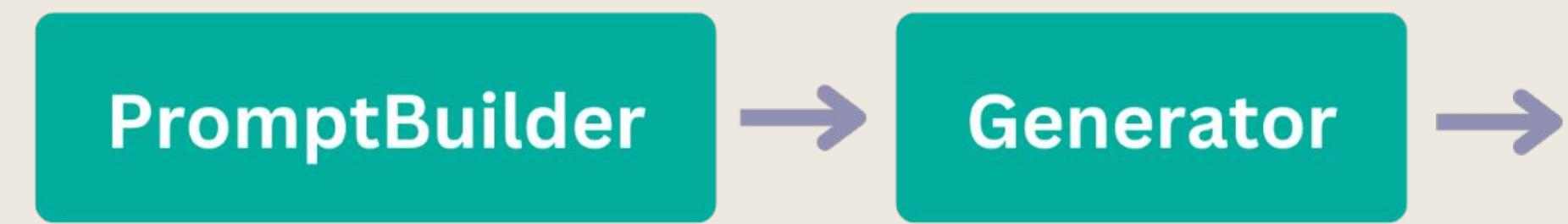
d



```
class City(BaseModel):  
    name: str  
    country: str  
    population: int
```



```
class CitiesData(BaseModel):  
    cities: List[City]
```



Extracting Structured Data

Berlin is the capital of Germany. It has a population of 3,850,809. Paris, France's capital, has 2.161 million residents. Lisbon is the capital and the largest city of Portugal with the population of 504,718.

text

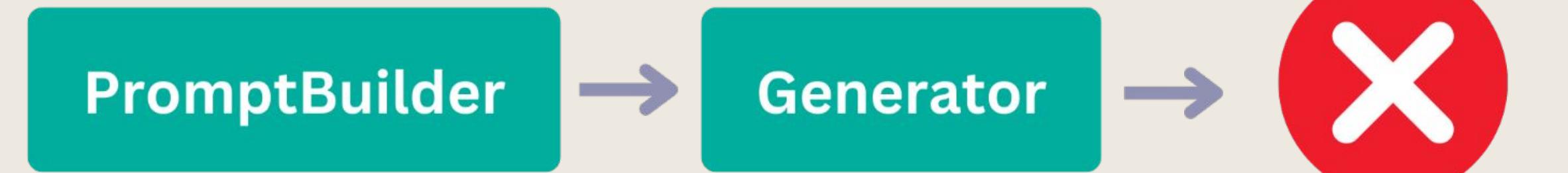
d



```
class City(BaseModel):  
    name: str  
    country: str  
    population: int
```



```
class CitiesData(BaseModel):  
    cities: List[City]
```



Invalid JSON

Extracting Structured Data

Berlin is the capital of Germany. It has a population of 3,850,809. Paris, France's capital, has 2.161 million residents. Lisbon is the capital and the largest city of Portugal with the population of 504,718.

text

d



```
class City(BaseModel):  
    name: str  
    country: str  
    population: int
```



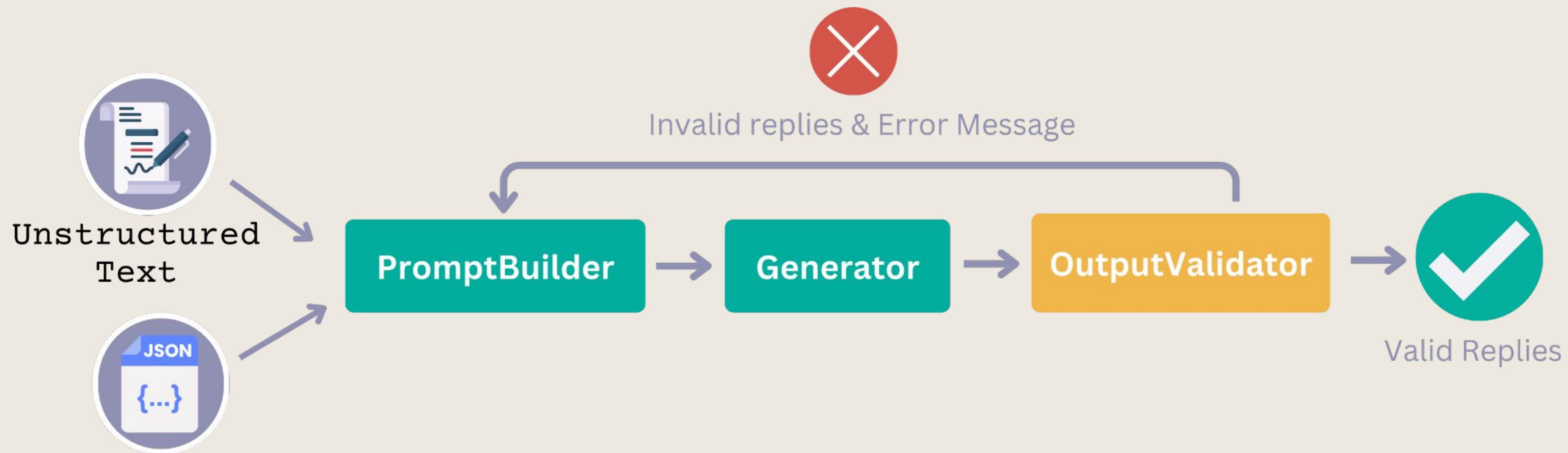
```
class CitiesData(BaseModel):  
    cities: List[City]
```

PromptBuilder

Generator

```
{  
    "cities": [  
        {  
            "name": "Berlin",  
            "population": 3850809,  
            "country": "Germany"  
        },  
        {  
            "name": "Paris",  
            "population": 2161000,  
            "country": "France"  
        },  
        {  
            "name": "Lisbon",  
            "population": 504718  
        }  
    ]  
}
```

Mechanism: Looping



OutputValidator



```
import pydantic
from haystack import component

@Component
class OutputValidator:
    def __init__(self, pydantic_model: pydantic.BaseModel):
        self.pydantic_model = pydantic_model

    @component.output_types(valid_replies=List[str], invalid_replies=Optional[List[str]], error_message=Optional[str])
    def run(self, replies: List[str]):

        try:
            output_dict = json.loads(replies[0])
            self.pydantic_model.parse_obj(output_dict)

        return {"valid_replies": replies}

    except (ValueError, ValidationError) as e:
        return {"invalid_replies": replies, "error_message": str(e)}
```

OutputValidator



```
import pydantic
from haystack import component

@Component
class OutputValidator:
    def __init__(self, pydantic_model: pydantic.BaseModel):
        self.pydantic_model = pydantic_model

    @component.output_types(valid_replies=List[str], invalid_replies=Optional[List[str]], error_message=Optional[str])
    def run(self, replies: List[str]):

        try:
            output_dict = json.loads(replies[0])
            self.pydantic_model.parse_obj(output_dict)

        return {"valid_replies": replies}

    except (ValueError, ValidationError) as e:
        return {"invalid_replies": replies, "error_message": str(e)}
```



Prompt Template



```
prompt_template = """
Create a JSON object from the information present in this passage: {{passage}}.
Only use information that is present in the passage. Follow this JSON schema, but only return
the actual instances without any additional schema definition:
{{schema}}
Make sure your response is a dict and not a list.
{% if invalid_replies and error_message %}
    You already created the following output in a previous attempt: {{invalid_replies}}
    However, this doesn't comply with the format requirements from above and triggered this Python
exception: {{error_message}}
    Correct the output and try again. Just return the corrected output without any extra
explanations.
{% endif %}
"""

```

Pipeline code



```
from haystack import Pipeline

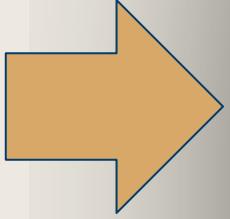
pipeline = Pipeline(max_loops_allowed=5)

# Add components to your pipeline
pipeline.add_component(instance=prompt_builder, name="prompt_builder")
pipeline.add_component(instance=generator, name="llm")
pipeline.add_component(instance=output_validator, name="output_validator")

# Now, connect the components to each other
pipeline.connect("prompt_builder", "llm")
pipeline.connect("llm", "output_validator")
# If a component has more than one output or input, explicitly specify the connections:
pipeline.connect("output_validator.invalid_replies", "prompt_builder.invalid_replies")
pipeline.connect("output_validator.error_message", "prompt_builder.error_message")

passage = "Berlin is the capital of Germany. It has a population of 3,850,809. Paris,  
France's capital, has 2.161 million residents. Lisbon is the capital and the largest city  
of Portugal with the population of 504,718."

pipeline.run({"prompt_builder": {"passage": passage, "schema": json_schema}})
```



Pipeline code



```
from haystack import Pipeline

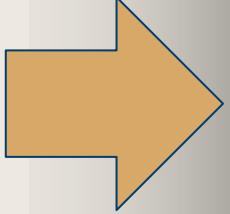
pipeline = Pipeline(max_loops_allowed=5)

# Add components to your pipeline
pipeline.add_component(instance=prompt_builder, name="prompt_builder")
pipeline.add_component(instance=generator, name="llm")
pipeline.add_component(instance=output_validator, name="output_validator")

# Now, connect the components to each other
pipeline.connect("prompt_builder", "llm")
pipeline.connect("llm", "output_validator")
# If a component has more than one output or input, explicitly specify the connections:
pipeline.connect("output_validator.invalid_replies", "prompt_builder.invalid_replies")
pipeline.connect("output_validator.error_message", "prompt_builder.error_message")

passage = "Berlin is the capital of Germany. It has a population of 3,850,809. Paris,  
France's capital, has 2.161 million residents. Lisbon is the capital and the largest city  
of Portugal with the population of 504,718."

pipeline.run({"prompt_builder": {"passage": passage, "schema": json_schema}})
```



Pipeline code



```
from haystack import Pipeline

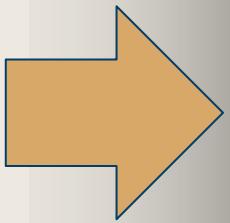
pipeline = Pipeline(max_loops_allowed=5)

# Add components to your pipeline
pipeline.add_component(instance=prompt_builder, name="prompt_builder")
pipeline.add_component(instance=generator, name="llm")
pipeline.add_component(instance=output_validator, name="output_validator")

# Now, connect the components to each other
pipeline.connect("prompt_builder", "llm")
pipeline.connect("llm", "output_validator")
# If a component has more than one output or input, explicitly specify the connections:
pipeline.connect("output_validator.invalid_replies", "prompt_builder.invalid_replies")
pipeline.connect("output_validator.error_message", "prompt_builder.error_message")

passage = "Berlin is the capital of Germany. It has a population of 3,850,809. Paris,  
France's capital, has 2.161 million residents. Lisbon is the capital and the largest city  
of Portugal with the population of 504,718."

pipeline.run({"prompt_builder": {"passage": passage, "schema": json_schema}})
```



Pipeline code



```
from haystack import Pipeline

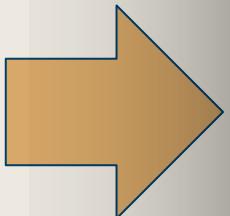
pipeline = Pipeline(max_loops_allowed=5)

# Add components to your pipeline
pipeline.add_component(instance=prompt_builder, name="prompt_builder")
pipeline.add_component(instance=generator, name="llm")
pipeline.add_component(instance=output_validator, name="output_validator")

# Now, connect the components to each other
pipeline.connect("prompt_builder", "llm")
pipeline.connect("llm", "output_validator")
# If a component has more than one output or input, explicitly specify the connections:
pipeline.connect("output_validator.invalid_replies", "prompt_builder.invalid_replies")
pipeline.connect("output_validator.error_message", "prompt_builder.error_message")

passage = "Berlin is the capital of Germany. It has a population of 3,850,809. Paris,  
France's capital, has 2.161 million residents. Lisbon is the capital and the largest city  
of Portugal with the population of 504,718."

pipeline.run({"prompt_builder": {"passage": passage, "schema": json_schema}})
```



Pipeline code



```
from haystack import Pipeline

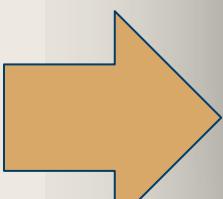
pipeline = Pipeline(max_loops_allowed=5)

# Add components to your pipeline
pipeline.add_component(instance=prompt_builder, name="prompt_builder")
pipeline.add_component(instance=generator, name="llm")
pipeline.add_component(instance=output_validator, name="output_validator")

# Now, connect the components to each other
pipeline.connect("prompt_builder", "llm")
pipeline.connect("llm", "output_validator")
# If a component has more than one output or input, explicitly specify the connections:
pipeline.connect("output_validator.invalid_replies", "prompt_builder.invalid_replies")
pipeline.connect("output_validator.error_message", "prompt_builder.error_message")

passage = "Berlin is the capital of Germany. It has a population of 3,850,809. Paris,  
France's capital, has 2.161 million residents. Lisbon is the capital and the largest city  
of Portugal with the population of 504,718."

pipeline.run({"prompt_builder": {"passage": passage, "schema": json_schema}})
```



Looping

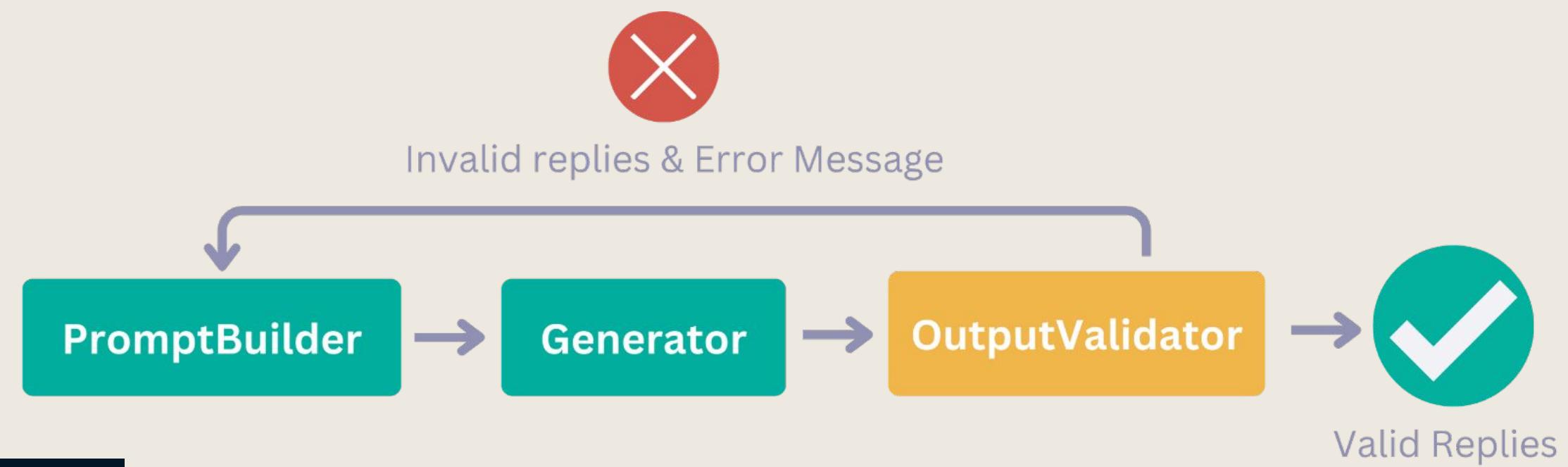
Berlin is the capital of Germany. It has a population of 3,850,809. Paris, France's capital, has 2.161 million residents. Lisbon is the capital and the largest city of Portugal with the population of 504,718.

Text



```
class City(BaseModel):  
    name: str  
    country: str  
    population: int
```

```
class CitiesData(BaseModel):  
    cities: List[City]
```



Looping

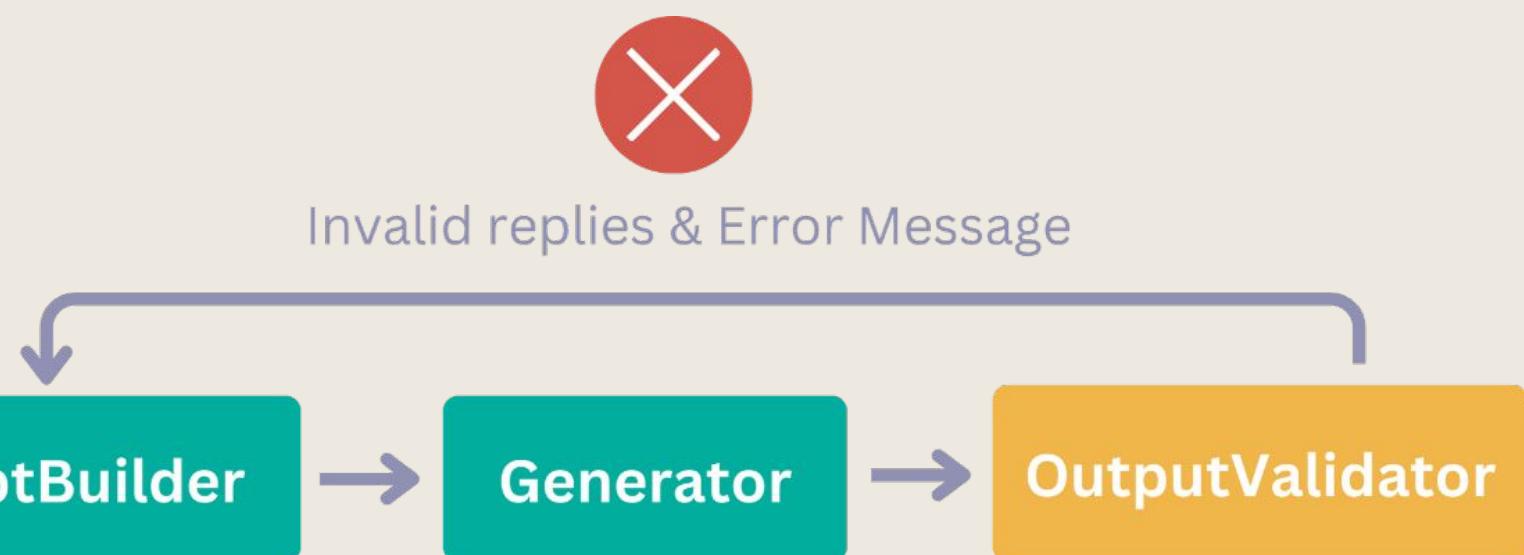
Berlin is the capital of Germany. It has a population of 3,850,809. Paris, France's capital, has 2.161 million residents. Lisbon is the capital and the largest city of Portugal with the population of 504,718.

Text



```
class City(BaseModel):  
    name: str  
    country: str  
    population: int
```

```
class CitiesData(BaseModel):  
    cities: List[City]
```



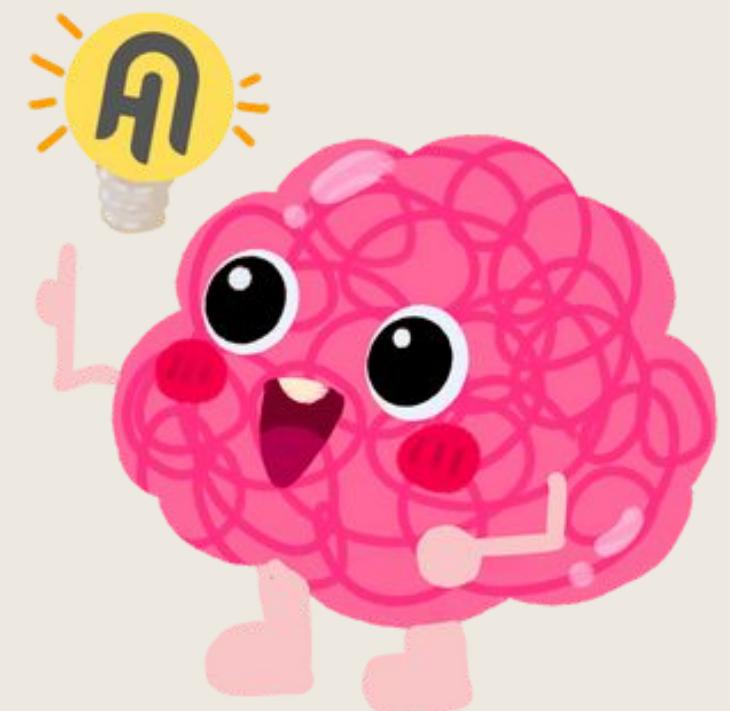
```
{  
    "cities": [  
        {  
            "name": "Berlin",  
            "country": "Germany",  
            "population": 3850809  
        },  
        {  
            "name": "Paris",  
            "country": "France",  
            "population": 2161000  
        },  
        {  
            "name": "Lisbon",  
            "country": "Portugal",  
            "population": 504718  
        }  
    ]  
}
```

elle

Recap



- Fallbacks are handy to make a search in alternative sources
- Fallback branches can be action triggers
- Early routing is also possible if you have defined options
- Loops are great for validation and auto-correction



Sources

 Fallback to Web



 Early Routing



 Generating Structured Output



Thank you



Haystack is open source! ⭐



haystack.deepset.ai

-  Documentation
-  Tutorials
-  Cookbooks
-  Articles
-  Join the Community

-  [@bilgeycl](#)
-  [Bilge Yücel](#)
-  [bilgeyucel](#)