# Working with Memory and Agents

- Memory and Agents
- Structured Data and Output Parsing Techniques
- Haystack & Its Usage

Ram N Sangwan

# Working with Memory

# Working with Memory

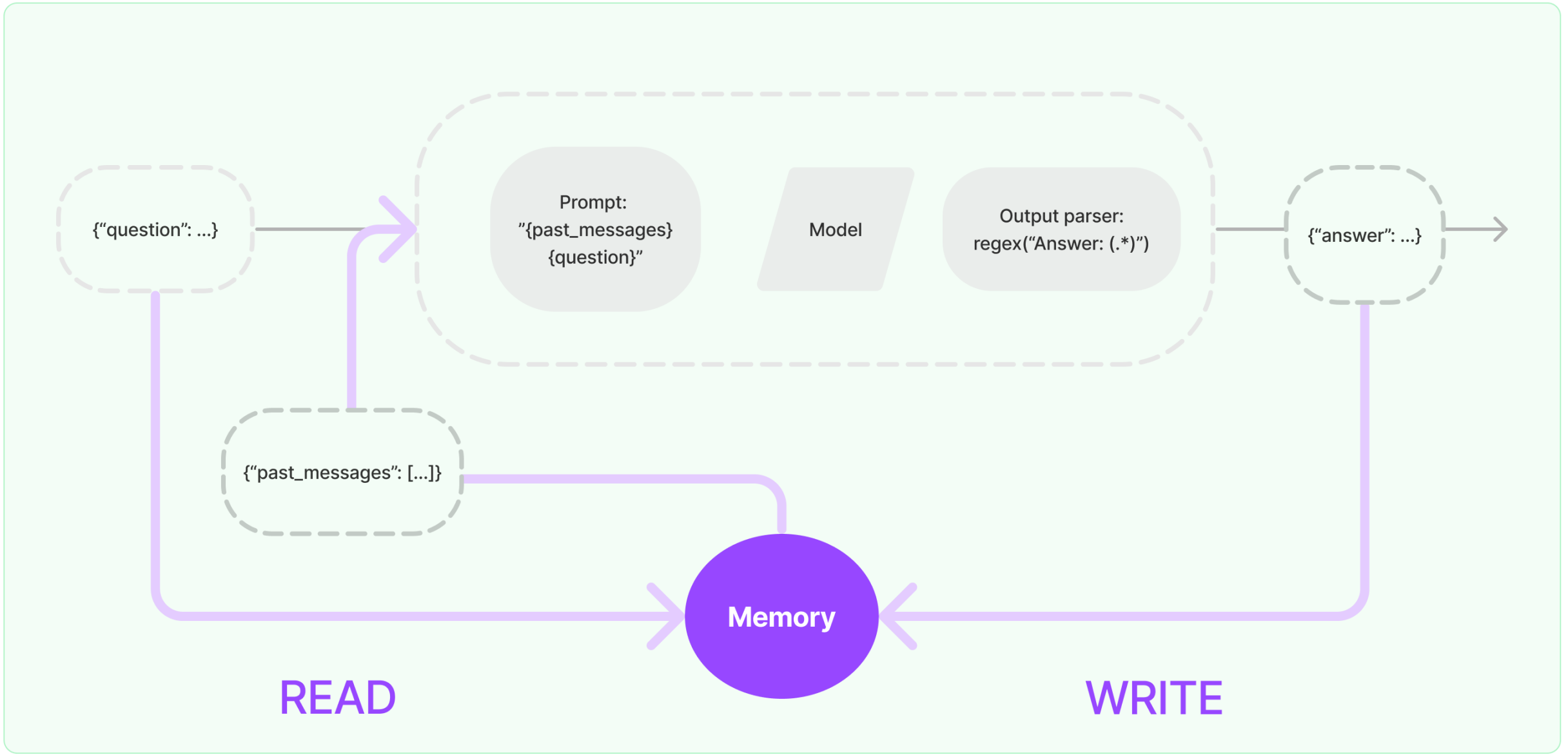
Most LLM applications have a conversational interface.

- An essential component of a conversation is being able to refer to information introduced earlier in the conversation.
- A conversational system should be able to access some window of past messages directly.
- We call this ability to store information about past interactions "memory".
- LangChain provides a lot of utilities for adding memory to a system.
- These utilities can be used by themselves or incorporated seamlessly into a chain.

# How it Works

A memory system needs to support two basic actions: *reading* and *writing*.

- Every chain defines some core execution logic that expects certain inputs.
- Some of these inputs come directly from the user, but some of these inputs can come from memory.
- A chain will interact with its memory system twice in a given run.
  1. AFTER receiving the initial user inputs but BEFORE executing the core logic, a chain will READ from its memory system and augment the user inputs.
  2. AFTER executing the core logic but BEFORE returning the answer, a chain will WRITE the inputs and outputs of the current run to memory, so that they can be referred to in future runs.


{“question”: ...}
Prompt:
”{past_messages}
{question}”
Model
Output parser:
regex(“Answer: (.*)”)
{“answer”: ...}
{“past_messages”: [...]}
Memory
READ
WRITE

# Working with Memory

Add memory to a chain:

```
from operator import itemgetter

from langchain.memory import ConversationBufferMemory
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_core.runnables import RunnableLambda, RunnablePassthrough
from langchain_community.chat_models import ChatCohere
import os
os.environ["COHERE_API_KEY"] = "wiytrjNSrookxQDxxkpYc1wibwEczpLYIZ1K7r90"

model = ChatCohere()
prompt = ChatPromptTemplate.from_messages(
    [
        ("system", "You are a helpful chatbot"),
        MessagesPlaceholder(variable_name="history"),
        ("human", "{input}"),
    ]
)
```

```
memory = ConversationBufferMemory(return_messages=True)
```

[4]:
```
chain = (
    RunnablePassthrough.assign(
        history=RunnableLambda(memory.load_memory_variables) | itemgetter("history")
    )
    | prompt
    | model
)
```

[5]:
```
inputs = {"input": "hi im Ram"}
response = chain.invoke(inputs)
response
```

[5]:
```
AIMessage(content="Hi Ram, I'm Coral, an AI-assistant chatbot trained to assist human users by providing thorough responses. It's grea
t to meet you. If you have any questions, or need any information on a particular subject, feel free to let me know. \n\nI'm here to h
elp in any way that I can.")
```

[6]:
```
memory.save_context(inputs, {"output": response.content})
```

[7]:
```
memory.load_memory_variables({})
```

[7]:
```
{'history': [HumanMessage(content='hi im Ram'),
  AIMessage(content="Hi Ram, I'm Coral, an AI-assistant chatbot trained to assist human users by providing thorough responses. It's gr
eat to meet you. If you have any questions, or need any information on a particular subject, feel free to let me know. \n\nI'm here to
help in any way that I can.")]}
```

[8]:
```
inputs = {"input": "whats my name"}
response = chain.invoke(inputs)
response
```

[8]:
```
AIMessage(content='Your name is Ram. If you have any further questions, feel free to ask me anytime!')
```

# More on Agents

# More on Agents

Official LangChain Documentation describes agents:

- Some applications will require not just a predetermined chain of calls to LLMs/other tools, but potentially an **unknown chain** that depends on the user's input. In these types of chains, there is a “agent” which has access to a suite of tools.
- Depending on the user input, the agent can then **decide which, if any, of these tools to call**.
- Basically you use the LLM not just for text output, but also for decision making.

# Agents

The language model that drives decision making.

- More specifically, an agent takes in an input and returns a response corresponding to an action to take along with an action input.
- You can see different types of agents for different use cases here.

# Tools

A 'capability' of an agent.

- This is an abstraction on top of a function that makes it easy for LLMs (and agents) to interact with it. Ex: Google search.

**Toolkit**

- Groups of tools that your agent can select from

# Working with Tools and Toolkit

```
from langchain.agents import load_tools
from langchain.agents import initialize_agent
from langchain.llms import Cohere
import json
import os
os.environ["COHERE_API_KEY"] = "wiytrjNSrookxQDxxkpYc1wibwEczpLYIZ1K7r90"

llm = Cohere(model="command", max_tokens=256, temperature=0.75)
```

```
!export SERP_API_KEY="58baec75dbad0620f59fe693592a0677d42f05d9ad287c564aed091c3c9ac389"
```

```
serpapi_api_key=os.getenv("SERP_API_KEY", "58baec75dbad0620f59fe693592a0677d42f05d9ad287c564aed091c3c9ac389")
```

```
toolkit = load_tools(["serpapi"], llm=llm, serpapi_api_key=serpapi_api_key)
```

```
agent = initialize_agent(toolkit, llm, agent="zero-shot-react-description", verbose=True, return_intermediate_steps=True)
```

```
response = agent({"input":"what was the first album of the"
                  "band that Sonu Nigam is a part of?"})
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/langchain_core/_api/deprecation.py:117: LangChainDepre
cationWarning: The function `__call__` was deprecated in LangChain 0.1.0 and will be removed in 0.2.0. Use invoke instead.
  warn_deprecated(
```

```
> Entering new AgentExecutor chain...
 this band is called 'Silk Band' and was founded by Sonu Nigam himself. Let's see if we can find the name of their first album online.
Action: Search
Action Input: first silk band album sonu nigam
```

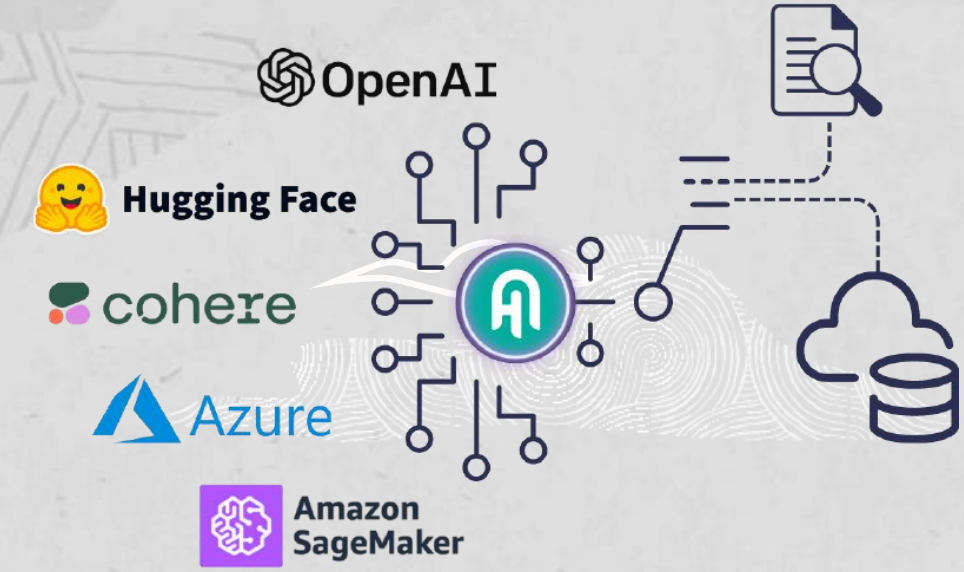
# Haystack & Its Usage

# What is Haystack?

- Open source Python framework by deepset for building custom apps with large language models (LLMs).
- It lets you quickly try out the latest models in natural language processing (NLP) while being flexible and easy to use.
- Community has helped shape Haystack into what it is today:
  - a complete framework for building production-ready NLP apps.

https://haystack.deepset.ai/overview/intro

# Building with Haystack




- Haystack offers comprehensive tooling for developing state-of-the-art NLP systems that use LLMs (such as Models from Cohere, Falcon).
- With Haystack, you can effortlessly experiment with various models hosted on platforms like Hugging Face, Cohere, or your local models to find the perfect fit for your use case.

# What you can build

- **Semantic search** on a large collection of documents in any language
- **Generative question answering** on a knowledge base containing mixed types of information:
  - images, text, and tables.
- **Natural language chatbots** powered by cutting-edge generative models like Cohere
  - An LLM-based Haystack **Agent** capable of resolving complex queries
- **Information extraction** from documents to populate your database or build a knowledge graph

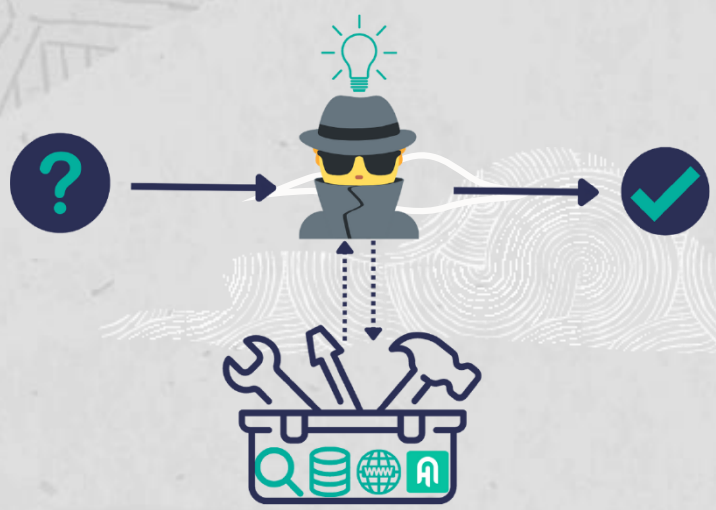
# Components

- At the core of Haystack are its components - fundamental building blocks that can perform tasks like document retrieval, text generation, or summarization.
- A single component is already quite powerful.
- It can manage local language models or communicate with a hosted model through an API.

# Pipelines

- Pipelines are structures made up of components, such as a Retriever and Reader, connected to infrastructure building blocks, such as a DocumentStore (for example, Elasticsearch) to form complex systems.
- Haystack offers ready-made pipelines for most common tasks, such as question answering, document retrieval, or summarization.

# Haystack Agents



- Makes use of a large language model to resolve complex tasks.
- When initializing the Agent, you give it a set of tools, which can be pipeline components or whole pipelines.
- The Agent can use to those tools iteratively to arrive at an answer.
- When given a query, the Agent determines which tools are useful to answer this query and calls them in a loop until it gets the answer.
- This way, it can achieve much more than extractive or generative question answering pipelines.

# LangChain Vs Haystack

Ram N Sangwan

# LangChain - Use Cases

1. **Complex NLP Workflows**:

   For multiple NLP tasks (e.g., text generation, summarization, translation) in a single workflow, LangChain provides flexible ways to chain these tasks together.

2. **Custom Conversational Agents:**

   Agents tailored to specific tasks or domains, LangChain allows you to build and configure these agents effectively.

3. **Task Automation:**

   For automating repetitive text-based tasks such as drafting emails or generating reports.

4. **Integrating Various APIs:**

   LangChain can interact with multiple language model APIs (OpenAI GPT, Cohere, etc.).

5. **Experimentation and Prototyping.**

# LangChain - Not to Use Cases

1. **Simple, Single-Task Applications:**

   E.g., sentiment analysis, LangChain might be overkill.

2. **Heavy-Duty Information Retrieval:**

   LangChain is not specifically optimized for large-scale information retrieval tasks, where specialized frameworks like Haystack might be more appropriate.

3. **Resource-Constrained Environments:**

   LangChain's flexibility and feature set might introduce overhead that is not ideal for very resource-constrained environments.

# Haystack - Use Cases

1. **Document Search and Retrieval:**

   Haystack excels at building search systems that can retrieve relevant documents from a large corpus based on a query. This makes it ideal for enterprise search, legal document search, and academic research.

2. **Question Answering Systems:**

   A system that can answer questions based on a specific set of documents, Haystack provides robust tools for extracting answers from text.

3. **Knowledge Bases and FAQs:**

   Haystack is well-suited for creating dynamic FAQs and knowledge bases.

4. **Pipeline Customization:**

   Enabling the integration of various components such as retrievers, readers, and rankers.

5. **Scalability:**

   Haystack is designed to handle large datasets and can be scaled to meet the needs of enterprise-level applications.

# Haystack - Not to Use Cases

1. **General NLP Tasks:**

   Haystack is specialized for search and retrieval tasks. If you need to perform a variety of general NLP tasks (e.g., text generation, summarization), other frameworks like LangChain might be more suitable.
2. **Simple Applications.**
3. **Real-Time Conversational Agents.**
4. **Limited Data Scenarios:**

   If your application does not involve large datasets or complex retrieval tasks, the benefits of Haystack's powerful retrieval capabilities might not be fully realized.

# LangChain Vs Haystack - Summary

- **LangChain** is ideal for complex NLP workflows, custom conversational agents, and task automation.
- **Haystack** excels in document search and retrieval, question answering systems, and scalable knowledge bases.

# Thank You