AI agent evaluation from prompts to metrics

What are AI agents?

An AI agent refers to a software system capable of autonomously carrying out tasks on behalf of a user or another system, by developing its own workflow and using external tooling as required.

Agents go well beyond simple language processing and understanding. They are capable of decision making, problem solving, interacting with the environment and acting in pursuit of goals.

AI agents are now being incorporated into a variety of enterprise solutions, from IT automation and software engineering, to conversational interfaces and code generation implementation. Driven by large language models (LLMs), they can comprehend complex directions, decompose them into steps, interact with resources from outside sources and have the cognitive ability to know when to deploy certain tools or services to help achieve tasks.

Why is evaluating the agents important?

Agent evaluation is an important procedure when creating and deploying autonomous AI systems because it measures how well an agent performs the tasks assigned, makes decisions and interacts with users or environments. This way we can ensure that agents operate reliably, efficiently and ethically in their intended use cases.

Key reasons for agent evaluation include:

- **Functional verification:** This step helps to verify the agent's behaviors and actions in certain conditions, as well as the completion of its objectives in defined constraints.
- **Design optimization:** Identifies the shortcomings and inefficiencies in the agent's reasoning, planning or tool use, allowing us to iteratively improve the agent's architecture and flow.
- **Robustness:** Evaluates the agent's ability to encounter edge cases, adversarial inputs or suboptimal conditions, which can improve fault tolerance and resiliency.
- **Performance and resource metrics:** The observation of latency, throughput, token consumption, memory and other system metrics can be tracked so that we can determine runtime efficiencies and to minimize operational costs.
- **User interaction quality:** Measures the clarity, helpfulness, coherence and relevance of the agent's responses as an indicator of user satisfaction or conversational effectiveness.
- **Goal completion analysis:** By using success criteria, or specific task-based benchmarks, we can assess how reliably and accurately the agent completed its goals.
- **Ethical and safety considerations:** The outputs of the agent can be evaluated for fairness, bias, potential harm and adherence to any safety procedures.

Evaluation metrics for AI agents

Assessing an AI agent's performance uses metrics organized in several formal classes of performance: accuracy, response time (speed) and cost of resources used. Accuracy describes how well the agent gives the correct and relevant responses, as well as the agent's capacity to complete its intended functions. Response time measures the speed that the agent takes to process the input and to produce output. Minimizing latency is especially important in interactive and real-time programs and cost measures the computational resources the agent consumes, such as token use, call to an application programming interface (API) or system time. These metrics provide guidelines to improve the performance of the system and limit operational costs.

While key metrics such as **correctness**, **helpfulness** and **coherence** fall under accuracy, response time (latency) measures metrics including **throughput**, **average latency** and **timeout delay**. Cost metrics include **token usage**, **compute time**, **API call count** and **memory consumption**.

In this tutorial we will explore the key metrics of **correctness, helpfulness and coherence** that fall under **accuracy**.

- **Correctness:** Correctness assesses whether the agent's responses are factually accurate and logically true from the input prompt or task. This metric is often the most basic measure, particularly for fields such as healthcare, legal advice or technical support.
- **Helpfulness:** Helpfulness assesses how useful or actionable the agent's response is for the user's intent. Even if a response is factually correct, it might not be helpful if it does not address a solution or next steps.
- **Coherence:** is related with flow—both logical and narrative flow. It is important in multiturn interactions and in interactions where reasoning is being done over multiple steps. Coherence refers to whether the agent "makes sense" from start to finish.

You will develop a travel agent and evaluate its performance by using an "LLM-as-a-judge."

Prerequisites

1. You need an IBM® Cloud® account to create a watsonx.ai® project.

2. You also need Python version 3.12.7

Steps

Step 1. Set up your environment

While you can choose from several tools, this tutorial walks you through how to set up an IBM account to use a Jupyter Notebook.

1. Log in to watsonx.ai by using your IBM Cloud account.

2. Create a watsonx.ai project. You can get your project ID from within your project. Click the **Manage** tab. Then copy the project ID from the **Details** section of the **General** page. You need this ID for this tutorial.

3. Create a Jupyter Notebook. This step opens a Jupyter Notebook environment where you can copy the code from this tutorial. Alternatively, you can download this notebook to your local system and upload it to your watsonx.ai project as an asset. To view more IBM Granite® tutorials, check out the IBM Granite Community.

Step 2. Set up a watsonx.ai Runtime instance and API key

1. Create a watsonx.ai Runtime service instance (select your appropriate region and choose the Lite plan, which is a free instance).

2. Generate an application programming interface (API) key.

3. Associate the watsonx.ai Runtime service instance to the project that you created in watsonx.ai.

Step 3. Install and import relevant libraries and set up your credentials

We need a few libraries and modules for this tutorial. Make sure to import the following ones and if they're not installed, a quick pip installation resolves the problem.

Note, this tutorial was built by using Python 3.12.7.

```
!pip install -q langchain langchain-ibm langchain_experimental langchain-text-splitters
langchain_chroma transformers bs4 langchain_huggingface sentence-transformers
import os import getpass import requests import random import json from typing import Type from
typing import Dict, List from langchain_ibm import WatsonxLLM from langchain_ibm import
ChatWatsonx from ibm_watsonx_ai.metanames import GenTextParamsMetaNames as GenParams
from langchain_ibm import WatsonxEmbeddings from langchain.vectorstores import Chroma from
langchain.prompts import PromptTemplate from langchain.chains import LLMChain from
langchain.agents.agent_types import AgentType from langchain.prompts import ChatPromptTemplate
from langchain.evaluation import load_evaluator from langchain.agents import initialize_agent, Tool
```

To set our credentials, we need the **WATSONX_APIKEY and WATSONX_PROJECT_ID** you generated in step 1. We will also set the URL serving as the API endpoint. Your API endpoint can differ depending on your geographical location.

```
WATSONX_APIKEY = getpass.getpass("Please enter your watsonx.ai Runtime API key (hit enter): ")
WATSONX_PROJECT_ID = getpass.getpass("Please enter your project ID (hit enter): ") URL =
"https://us-south.ml.cloud.ibm.com"
```

Step 4. Initialize your LLM

We will use the Granite 3 -8B Instruct model for this tutorial. To initialize the LLM, we need to set the model parameters. To learn more about these model parameters, such as the minimum and maximum token limits, refer to the documentation.

```
llm = ChatWatsonx(model_id="ibm/granite-3-8b-instruct", url = URL, apikey = WATSONX_APIKEY,
project_id = WATSONX_PROJECT_ID, params = {"decoding_method": "greedy","temperature": 0,
"min_new_tokens": 5, "max_new_tokens": 2000})
```

Step 5. Build a travel explorer agent (buddy)

Let's build a travel explorer buddy that helps users with trip planning and travel research.

We will create a simple travel assistant application that can retrieve airline and hotel information in response to user inquiries by connecting to an external travel API. In order to integrate with AI agents for dynamic travel planning, we will have a straightforward function that makes API queries and wrap it in a tool.

```
def travel_api(query: str) -> str: # Example of connecting to a real travel API response =
requests.get("https://www.partners.skyscanner.net", params={"query": query}) if response.status_code
== 200: return response.json().get("result", "No results found.") return "Error contacting travel API."
travel_tool = Tool( name="TravelPlannerTool", func=travel_api, description="Connects to a travel API
to find flights and hotels for a given city and date" )
agent = initialize_agent( tools=[travel_tool], llm=llm,
agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=True, handle_parsing_errors=
"Check your output and make sure it conforms! Do not output an action and a final answer at the same
time." )
query = "What are the best places to visit in India during winters?" response = agent.invoke(query)
print("\n--- Travel Agent Response ---") print(response)
```

Step 6. Run an evaluation and get the score

Finally, we run an evaluation and print the final evaluation score. In order to evaluate the trip planner by using three distinct criteria (correctness, helpfulness and coherence), a structured evaluation prompt is developed for an evaluator LLM.

```
from langchain.prompts import PromptTemplate from langchain.chains import LLMChain
evaluation_prompt = PromptTemplate( input_variables=["input", "prediction"], template=""" You are
an evaluation agent. ### User Input: {input} ### Agent's Response: {prediction} Evaluate this
response based on: - Correctness (Is the information factually accurate?) - Helpfulness (Is it useful and
actionable?) - Coherence (Is it well-written and logically structured?) Reply in this format:
Correctness: <score>/5 - <reason> Helpfulness: <score>/5 - <reason> Coherence: <score>/5 -
<reason> """ )
eval_input = evaluation_prompt.format(input=query, prediction=response) evaluation_result =
agent.invoke(eval_input)
```

The output shows both qualitative and quantitative assessment of the travel planner generated by using three criteria—correctness, helpfulness and coherence.

Let's break down what each score and metric means in the context of the agent's output:

- **Correctness** tells us how factually accurate and logical the response sounds. In the previous example, the factual content is correct, hence the correctness score is (5 out of 5).
- **Helpfulness** shows how helpful and pertinent the response is to the user's needs measured by its helpfulness. A score of (5 out 5) in this scenario means that the AI travel plan is useful and thoughtfully designed. The score indicates that It is helpful for someone searching for the best places to visit in India during winter for the first time.
- **Coherence** shows if the planner is logically organized and easy to read. Our example received a high coherence score of 5.

Conclusion

When evaluating an agent's ability to truly meet user needs, criteria such as coherence, helpfulness and accuracy play a central role. Regardless of whether you're working with OpenAI, IBM Granite or other LLM-as-a-service models, it's crucial to rely on structured evaluation methods—such as evaluation datasets, benchmarks, annotations and ground truth—to thoroughly test final outputs. In practical use cases like chatbots or RAG-based customer support, open source frameworks like LangGraph are invaluable. They support scalable automation, dependable routing and enable rapid iteration cycles.

These technologies also make it easier to power generative AI systems, debug behaviors and optimize and configure complex workflows. By carefully defining test cases and keeping an eye on observability metrics like computation cost, price and latency, teams can consistently improve system performance. Ultimately, applying a reliable and repeatable evaluation approach brings rigor to machine learning systems and strengthens their trustworthiness over time.