UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Evaluating Tool-Augmented ReAct Language Agents

Author:
Jokin Eguzkitza

Supervisor:
Laura Igual
Pablo Álvarez

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

June 29, 2025

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc in Fundamental Principles of Data Science

Master Thesis Title

by Jokin Eguzkitza

This thesis studies how to evaluate ReAct agents that use external tools. Using LangGraph and LangChain three different AI agents are created using locally deployed LLM models served with Ollama. These agents use open-source tools like Wikipedia, Wikidata, Yahoo Finance and PDF readers. To evaluate them, the project combines rule-based checks with RAGAS metrics to measure tool use, answer quality, factual correctness and context use. The results show that prompt design is very important to guide the agent's behaviour, and that typical question-answer metrics are not always enough to measure how well an agent works. This work offers a simple and practical way to test LLM agents in more realistic tasks.

All the corresponding code notebook can be found on the following repository, `https://github.com/Jokinn9/Evaluating-Tool-Augmented-ReAct-Language-Agents`

Acknowledgements

I would like to thank my academic advisor, Laura Igual, for her support, feedback and guidance throughout the development of this thesis. I also want to thank Pablo Álvarez, my company supervisor, for his practical advice and help during the project. I also would like to thank my classmates and friends for their encouragement and useful discussions during this year, as well as my family for their constant support and patience.

Contents

Abstract ii

Acknowledgements iii

1 Introduction 4
- 1.1 Introduction . . . 4
- 1.2 The Rise of the LLM Agent Ecosystem . . . 4
- 1.3 The Challenge of Evaluating LLM Agents . . . 5
- 1.4 Objectives . . . 5

2 Background 7
- 2.1 What Are Agents? . . . 7
- 2.2 From Agents to Agentic Design Patterns . . . 7
  - 2.2.1 Tool Use Pattern . . . 8
  - 2.2.2 Reflection Pattern . . . 8
  - 2.2.3 ReAct Pattern . . . 8
  - 2.2.4 Planning Pattern . . . 9
  - 2.2.5 Multi-Agent Pattern . . . 9
- 2.3 Evaluation Framework and Metrics . . . 10
  - 2.3.1 Custom Evaluation Metrics . . . 10
  - 2.3.2 Evaluation with RAGAS . . . 11

3 Experimental Design and Results 14
- 3.1 Common Agent Architecture . . . 14
  - 3.1.1 Choice of Language Model . . . 14
  - 3.1.2 Agent Construction with LangGraph . . . 14
  - 3.1.3 Shared Agent Loop and Tool Integration . . . 15
  - 3.1.4 Message Format Adaptation . . . 15
  - 3.1.5 Evaluation Setup . . . 16
- 3.2 Agent 1: Baseline ReAct Agent . . . 16
  - 3.2.1 Available Tools . . . 16
  - 3.2.2 Example Behaviour . . . 16
  - 3.2.3 Implementation Notes . . . 17
  - 3.2.4 First Evaluation Results and Observations . . . 17
  - 3.2.5 Limitations and Targeted Improvements . . . 18
- 3.3 Agent 2: React Agent with Wikipedia and Wikidata Tools . . . 20
  - 3.3.1 Available Tools . . . 20
  - 3.3.2 Implementation Notes . . . 20
  - 3.3.3 First Evaluation Results and Observations . . . 20
  - 3.3.4 Limitations and Targeted Improvements . . . 23
- 3.4 Agent 3: Metal-Focused Agent with PDF, Finance and Wikipedia Tools 26
  - 3.4.1 Available Tools . . . 26
  - 3.4.2 Implementation Notes . . . 26

3.4.3 First Evaluation Results and Observations 26
3.4.4 Limitations and Targeted Improvements 28

4 Conclusions **30**

Bibliography **33**

Chapter 1

Introduction

1.1 Introduction

In recent years, the development of large language models (LLMs) has significantly transformed the field of artificial intelligence. Trained on massive text datasets and later fine tuned for different tasks, models like OpenAI's GPT-4 (OpenAI, 2023), Meta's LLaMA (Touvron et al., 2023) and Mistral (Jiang et al., 2023) have shown strong performance in natural language understanding, reasoning and generation. These models have rapidly moved from being research tools to becoming core elements in both industry and academia, supporting applications such as conversational agents, educational tutors, legal reasoning systems and tools for automated software development.

Building on these capabilities, a new generation of intelligent systems has appeared, AI agents, systems that embed LLMs within agentic frameworks. These agents expand the functionality of LLMs beyond isolated prompt completion by enabling structured reasoning, dynamic decision-making and interaction with external tools such as APIs, databases or code execution environments. This leads to a change from passive language models to active agents that can follow goals, complete tasks and adapt to new information.

These agents are powerful because they are flexible and can work on their own. They combine language generation with things like memory, planning, tool use and even working together with other agents. Because of this, they are now used in more complex tasks such as research, customer support or data analysis. What used to be just an idea, the idea of an AI that can think, act and adapt, is now a real solution used in many areas (Amazon Web Services, 2024).

1.2 The Rise of the LLM Agent Ecosystem

This rapid progress has been made possible not only by the models themselves but also by the growing ecosystem of tools and frameworks that make their orchestration easier. Libraries like LangChain, LangGraph, AutoGen and CrewAI have introduced modular structures for creating agents that combine language generation with tool use, memory and control flows . These frameworks provide ways to build agents as sequences of reasoning steps, decision points and external actions, effectively turning LLMs into systems that can be programmed to reason and act.

At the same time, the open-source movement has played a crucial role in making these technologies more accessible. Initiatives such as Hugging Face Transformers, the release of open-source LLM models like LLaMA and Mistral, and tool that allow

them to deploy this LLMs like Ollama or LM Studio have enabled to build and experiment with LLM agents without depending on commercial APIs or cloud-based black boxes.

As a result, there has been a rapid increase in agentic applications, from simple personal assistants to fully autonomous multiagent systems. These agents are no longer just experimental prototypes, they are now being integrated into real production environments, business processes and decision support systems across different sectors. The ability to design and deploy these systems using only open source components makes this trend especially relevant from both a technological and social point of view (Wu et al., 2023).

1.3 The Challenge of Evaluating LLM Agents

The increasing complexity and autonomy of LLM-based agents has also introduced new challenges, particularly in how these systems are evaluated. Traditional NLP evaluation metrics, such as exact match, are not sufficient to capture the behaviour of agents operating in dynamic environments. Unlike static models that produce a single response to a prompt, agents often follow multi-step reasoning chains, call external tools, adapt to user feedback and make decisions based on internal state or memory. Evaluating these systems requires a more detailed approach, one that considers both the final result and the reasoning process that leads to it.

Moreover, agent performance cannot be reduced to a single metric. A proper evaluation should take into account multiple aspects of behaviour and reasoning. These questions are essential not only for benchmarking but also for real world deployment. Poor reasoning or incorrect tool use can lead to issues such as misinformation, system failures or loss of trust. As agents are increasingly adopted in areas as healthcare and finance, the need for robust and transparent evaluation methods becomes necessary (Wu et al., 2025). Some of the key questions to consider include the following:

- Is the answer correct or helpful?
- Were the appropriate tools selected and used?
- Is the reasoning trace logically sound and interpretable?
- How robust is the agent to input variation or ambiguous queries?
- Are there signs of hallucination or inconsistent behaviour?

1.4 Objectives

This thesis emerges from the need to demonstrate how agents can be evaluated. The focus is not only on assessing whether they produce correct outputs, but also on examining how they reach those outputs, how they interact with tools and how their behaviour changes across different tasks. By doing so, the goal is to provide a structured and reproducible framework for evaluating agentic systems that rely on open source components and can be applied in practical scenarios.

The challenge is that while language models can be evaluated using embeddings and well-established metrics, this is not as straightforward with agents. Even though

some evaluation is possible, current tools are more limited. Since this is still a developing area, there is not much work yet on how to properly assess whether an agent is working as expected (Wu et al., 2023). For this reason, we will use the RAGAS library (RAGAS, 2024), as it offers different metrics specifically designed for evaluating this type of system. The idea is to build the agent, send it a set of queries and evaluate the responses. Based on the results, we will analyse where the agent fails and use that information to improve its performance.

The data used to evaluate the different agents is synthetic, meaning it has been generated internally. Since each agent is designed to handle a specific type of query, the evaluation set is created to simulate realistic usage. These queries are not taken from existing datasets but are instead crafted to reflect the tasks the agents are expected to perform. This allows us to test their behaviour in a controlled but relevant way.

Chapter 2

Background

2.1 What Are Agents?

Large Language Models (LLMs) such as GPT, LLaMA or Mistral have significantly advanced the field of natural language processing. These models are impressive at tasks like summarization, translation, question answering and text generation. However, LLMs are fundamentally passive systems: they take an input prompt, produce a probabilistic response and terminate the interaction. They lack memory, persistence or the ability to act autonomously across multiple steps or over time.

These limitations have pushed the development of new approaches in AI. While LLMs are strong at generating text, agents build on top of them by adding memory, tool use and the ability to interact with the environment (Fauscette, 2024).

Agents operate within defined boundaries and are capable of adapting to different inputs or goals. They use tools like APIs or databases to perform tasks that go beyond simple text generation, ranging from automation to more complex problem-solving.

They are especially useful in open-ended and dynamic settings where instructions are given in natural language, such as virtual assistants or embedded helpers. With little to no human supervision, these systems can manage workflows, trigger actions and combine multiple sources of information to complete a task (Google Cloud, 2025).

2.2 From Agents to Agentic Design Patterns

As LLM-based agents have become more common, the way they are built and used has started to vary. Although they all aim to enable autonomous behaviour through language models, the way they reason, act and use tools can differ quite a bit between systems.

This variation comes from underlying design choices, such as how the agent's control flow is organised, how much reasoning is passed to the model, how memory is managed and how actions are selected and carried out. These recurring structures are often described as agentic design patterns. Understanding these patterns is essential not only for building agents, but also for evaluating them. Each pattern brings distinct strengths and weaknesses in terms of interpretability, generalization and performance. Therefore, the choice of agentic design is not merely technical but depends on the application it is intended for. Five of the most popular design patterns used building AI agents are explained below.

2.2.1 Tool Use Pattern

The tool use pattern is based on extending the capabilities of a language model by allowing it to interact with external tools (Figure 2.1). These tools can include things like querying a vector database, running Python code or calling APIs. This gives the agent access to up to date or specialised information, so it is not limited to the knowledge stored in the model itself. Instead of trying to generate the full answer on its own, the LLM decides whether it needs to use a tool and how to structure the tool call. It is especially useful in scenarios where real-time data or external operations are needed (Microsoft, 2024).

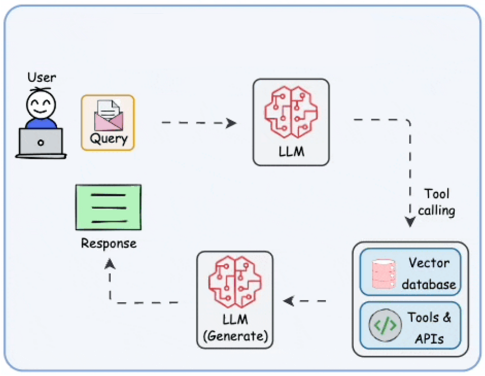

FIGURE 2.1: Architecture of a Tool Use Pattern.

2.2.2 Reflection Pattern

The reflection pattern is based on the idea that the agent can review its own output before giving a final answer (Figure 2.2). After generating an initial response, the model goes back, checks for possible mistakes or inconsistencies and tries to improve the result through one or more iterations. This process allows the agent to improve its reasoning and fix basic mistakes that may come up in the first try. It is used for tasks that involve several steps or need a certain level of precision. Rather than just generating an answer in one go, the model goes back, checks what it did and adjusts the output (Vidhya, 2024).

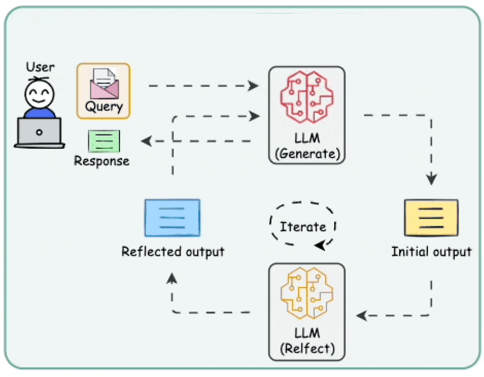

FIGURE 2.2: Architecture of a Reflection Pattern.

2.2.3 ReAct Pattern

The ReAct (Reason and Act) pattern combines the two previous ideas: the agent is able to reflect on its own reasoning and it can also interact with external tools (Figure

2.3). This means it can generate thoughts, decide on an action, execute it and then use the result to continue reasoning.

By mixing reflection and tool use, the agent becomes much more flexible and effective. It can break down a task into steps, access external information when needed and revise its own thinking. This approach was first introduced in the ReAct framework (Yao et al., 2023), which showed how combining reasoning and tool use leads to better performance in complex tasks.

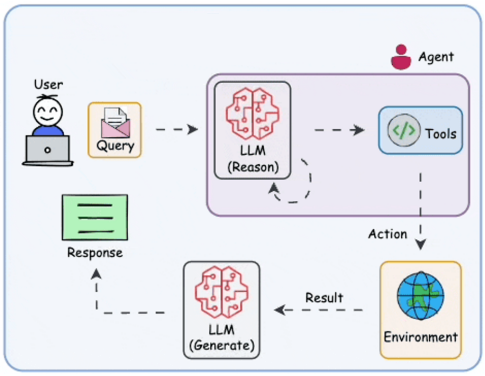

FIGURE 2.3: Architecture of a ReAct Pattern.

2.2.4 Planning Pattern

The planning pattern is about breaking a task into smaller parts before trying to solve it (Figure 2.4). Instead of generating an answer all at once, the agent starts by creating a plan that includes sub-goals or intermediate steps. This can involve subdividing the task, outlining what needs to be done or listing objectives that help guide the reasoning process (DeepLearning.AI, 2024a).

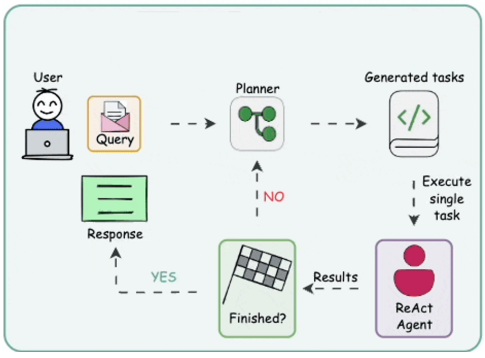

FIGURE 2.4: Architecture of a Planning Pattern.

2.2.5 Multi-Agent Pattern

Multi-agent pattern is based on using several agents that work together to complete a task (Figure 2.5). Each agent has a specific role and is responsible for a particular part of the process. Just like in other patterns, each agent can access tools when needed. What makes this setup different is the collaboration between agents. They can communicate, delegate tasks to one another and combine their outputs to produce the final result (DeepLearning.AI, 2024b).

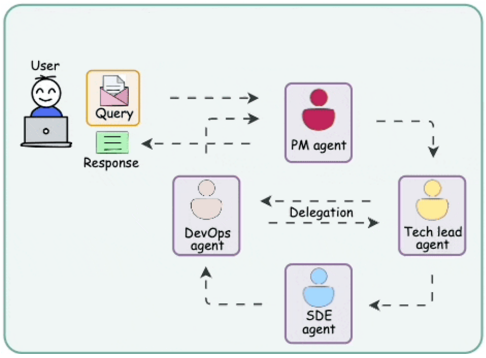

FIGURE 2.5: Architecture of a Multi-Agent Pattern.

2.3 Evaluation Framework and Metrics

Evaluating LLM-based agents requires more than assessing the correctness of their final output. As it has been explained in the previous section, section 2.2, these systems perform multi-step reasoning or make tool-related decisions all of which must be analysed to understand their real capabilities and limitations. To address this goal, different evaluation metrics will be used. Some are more traditional, while others are based on the RAGAS framework.

2.3.1 Custom Evaluation Metrics

To provide an interpretable and reproducible baseline, several manual and rule-based metrics are applied. These metrics are intuitive, offering insight into tool usage correctness, output structure and task efficiency.

- **Tool Call Accuracy (Exact Match):** this metric compares the actual tool calls made by the agent with a predefined list of expected tool invocations. A match is counted when both the tool name and its arguments match. Accuracy is computed as:

$$\text{Tool Accuracy} = \frac{\text{Number of correct tool calls}}{\text{Number of expected tool calls}}$$

- **Substring Match**: For tasks where specific keywords or formats are required (e.g., "EUR", "converted price"), a simple heuristic checks whether the expected substring appears in the final output. The metric returns a boolean value: `True` if the substring is found, and `False` otherwise.

- **Numeric Value Comparison**: When the output is expected to contain a specific numerical value, the system extracts all numbers from the response and compares them to a target value, allowing for a small floating point tolerance. The result is a boolean value: `True` if the expected number (within tolerance) is found, and `False` otherwise.

- **Number of reasoning steps**: In addition to correctness, the agent trace is analysed by counting the number of reasoning steps involved in producing the final answer.

2.3.2 Evaluation with RAGAS

To complement the rule-based approach, the thesis integrates RAGAS (Retrieval Augmented Generation Assessment), a framework designed for evaluating LLM-based systems using language model scoring and semantic understanding.

- **ToolCallAccuracy (Multi-Turn)**: This metric accounts for tool use correctness over time, even when names or arguments are not literal string matches (*RAGAS Metrics for Agents: Tool Call Accuracy* 2024). It returns `True` if the correct tool was used with appropriate arguments at any point in the interaction, and `False` otherwise.

$$\text{Score} = \frac{\text{Number of correct tool uses}}{\text{Total number of queries evaluated}}$$

RAGAS also provides more classical QA style metrics:

- **Response Relevancy:** measures how relevant a response is to the user input. Higher scores indicate better alignment with the user input, while lower scores are given if the response is incomplete or includes redundant information (*RAGAS: Answer Relevance Metric* 2024). This metric is calculated using the `user_input` and the `response` as follows:

  1. Generate a set of artificial questions (default=3) based on the response. Questions designed to reflect the content of the response.
  2. Compute the cosine similarity between the embedding of the user input (E_0) and the embedding of each generated question (E_{g_i}).
  3. Take the average of these cosine similarity scores.

$$\text{Response Relevancy} = \frac{1}{N}\sum_{i=1}^{N}\text{cosine similarity}(E_{g_i}, E_0)$$

$$\text{Response Relevancy} = \frac{1}{N}\sum_{i=1}^{N}\frac{E_{g_i}\cdot E_0}{\|E_{g_i}\|\|E_0\|}$$

  Where:

  - E_{g_i}: Embedding of the i^{th} generated question.
  - E_0: Embedding of the user input.
  - N: Number of generated questions (default is 3).

  An example can help to understand this:

  - **User question:** *Who is the president of Finland?*
  - **Agent response:** *The current president of Finland is Alexander Stubb.*
  - **Generated questions based on the response:**
    - Who is the current president of Finland?
    - What is the name of Finland's president?
    - Who holds the presidency in Finland?

These generated questions are compared with the original user question using cosine similarity between their embeddings. Since they are semantically very close, the response receives a high *Response Relevancy* score.

- **Faithfulness:** measures how factually consistent a response is with the retrieved context. It ranges from 0 to 1, with higher scores indicating better consistency (*RAGAS: Faithfulness Metric* 2024).

  A response is considered faithful if all its claims can be supported by the retrieved context.To calculate this:

  1. Identify all the claims in the response.
  2. Check each claim to see if it can be inferred from the retrieved context.
  3. Compute the faithfulness score using the formula:

$$\text{Score} = \frac{\text{No. of claims in the response supported by the retrieved context}}{\text{Total number of claims in the response}}$$

  Example.

  - **User question:** *Who is the president of France?*
  - **Retrieved context:** *Emmanuel Macron*
  - **Agent response:** *Emmanuel Macron became president of France in 2010 and was born in Marseille.*

  Claims in the response:

  1. *Emmanuel Macron is president of France* → Supported
  2. *He became president in 2010* → Not supported
  3. *He was born in Marseille* → Not supported

  Faithfulness score:

$$\text{Faithfulness} = \frac{1}{3} \approx 0.33$$

- **Context Precision:** measures the proportion of relevant chunks in the *retrieved contexts*. It is calculated as the mean of the precision@k for each chunk in the context. Precision@k is the ratio of the number of relevant chunks at rank k to the total number of chunks at rank k (*RAGAS: Context Precision Metric* 2024). In our setup, only one chunk is retrieved per query ($K = 1$), so the metric simplifies to a binary decision based on whether that single chunk is relevant or not.

$$\text{Context Precision@}K = \frac{\sum_{k=1}^{K} (\text{Precision@}k \times v_k)}{\text{Total number of relevant items in the top } K \text{ results}}$$

$$\text{Precision@}k = \frac{\text{true positives@}k}{\text{true positives@}k + \text{false positives@}k}$$

  Where K is the total number of chunks in `retrieved_contexts` and $v_k \in \{0, 1\}$ is the relevance indicator at rank k.

Example 1. Relevant context (score = 1.0)

- **User question:** *Who is the president of France?*
- **Retrieved context:** *Emmanuel Macron*
- **LLM evaluation:** The context is relevant ($v_1 = 1$)

$$\text{Context Precision@1} = \frac{1.0 \times 1}{1} = 1.0$$

Example 2 – Irrelevant context (score = 0.0)

- **User question:** *What is the capital of Spain?*
- **Retrieved context:** *Barcelona is famous for its modernist architecture.*
- **LLM evaluation:** the fragment does not answer the question, so it is marked as not relevant ($v_1 = 0$)

$$\text{Context Precision@1} = \frac{1 \times 0}{1} = 0.0$$

These are computed using a locally deployed LLM and open-source embeddings.

Chapter 3

Experimental Design and Results

3.1 Common Agent Architecture

In this project, all evaluated agents share a common architectural structure. While their behaviour is defined by different design patterns and tool configurations, the underlying agent loop, language model and framework is the same.

3.1.1 Choice of Language Model

A core component of any LLM-based agent is the language model itself, which serves as the reasoning engine behind decisions, tool selection, and natural language generation. While commercial models such as ChatGPT or Claude are the best and the most widespread LLMs, they require access to paid APIs.

To address this problem, this project uses a locally hosted open source model, LLaMA 3.2 [1] , developed by Meta AI. In order to run this LLM, the Ollama[2] framework was used, which allows different language models to be downloaded and executed locally on personal hardware. At first, the idea was to work with Mistral because of its smaller size. However, after running several experiments, Mistral was replaced by LLaMA 3.2 due to its instability and inconsistent results.

3.1.2 Agent Construction with LangGraph

To manage the reasoning and decision-making of the agent, this project uses LangGraph[3], a framework that lets you build LLM agents as stateful graphs. In this setup, the agent is represented as a directed graph where each node performs a specific step and the edges define how the process moves from one step to the next. The architecture used in this project has of the following nodes:

- Start Node: Initializes the interaction and passes the user query into the agent flow.
- Agent Node: Invokes the LLM to perform reasoning and decide whether a tool should be used.
- Tools Node: Executes external tool calls if requested, such as querying prices or performing conversions.
- End Node: Terminates the execution either after producing a final answer or reaching a stop condition.

[1] https://ollama.com/library/llama3.2
[2] https://ollama.com/
[3] https://www.langchain.com/langgraph

This structure is wrapped in a named subgraph called `assistant`, which allows LangGraph to control while keeping a clear separation between the reasoning part and the actions taken by the agent.

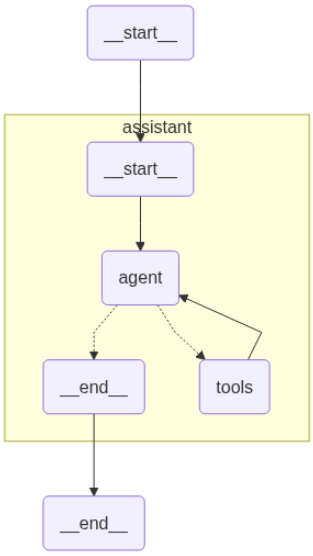

FIGURE 3.1: LangGraph structure of the agent loop.

As shown in Figure 3.1, the flow begins with the *start* node and proceeds to the agent reasoning node. Based on the LLM's output, the system may either move to a tool invocation phase or end the interaction. After each tool call, the agent returns to the reasoning node with the new observation, which enables multi-step reasoning and chaining of actions.

3.1.3 Shared Agent Loop and Tool Integration

All the agents that will be created will follow the structure shown in Figure 3.1. However, there are two main differences between the agents:

1. Each agent will use different tools, such as currency conversion, internet access or metal pricing.

2. The reasoning they follow will be different. In other words, the prompt to build the agent will be different so that they adapt to the situation.

3.1.4 Message Format Adaptation

Although LLaMA 3.2 behaves similarly to OpenAI models in terms of reasoning and tool selection, it differs in how tool calls are structured in the output. LangGraph expects tool invocations to be returned in a dedicated `tool_calls` field, but LLaMA often embeds them directly in the text, which requires additional parsing to extract them properly.

Not only for LangGraph, in the evaluation phase too, RAGAS is designed to work with the `chat_format` message structure used by models like ChatGPT. For this reason, the message must be converted into the expected format before running the evaluation.

To achieve this transformation, a post-processing function was implemented to reformat the model's output. It extracts tool names and arguments and injects them into the expected schema without altering the model's reasoning.

3.1.5 Evaluation Setup

In order to compute the evaluation metrics, the agent's responses need to be compared against the provided context and the original query. This process requires both a language model to interpret the responses and a set of embeddings to measure semantic similarity between texts.

For this purpose, the evaluation pipeline uses the open source model Mistral[4] as the LLM and the `all-MiniLM-L6-v2`[5] model from Sentence Transformers to generate embeddings. The embeddings are loaded via the `HuggingFaceEmbeddings` interface from LangChain[6].

Although the agent itself runs on LLaMA 3.2, a different model is used for evaluation. This separation is justified by the fact that the evaluation task has different requirements: rather than generating fluent responses, it focuses on comparing meaning and factual consistency. In this context, using a lighter and faster model like Mistral allows efficient scoring without significantly affecting the results.

3.2 Agent 1: Baseline ReAct Agent

This first agent serves as the baseline configuration for evaluating reasoning and tool use. It implements the ReAct design pattern, interleaving natural language reasoning with tool invocation in a loop guided by LangGraph.

3.2.1 Available Tools

The agent is equipped with two tools that simulate access to external data:

- `get_metal_price (metal_name)`: returns the current price (in USD per gram) of a specified metal.
- `get_currency_exchange (base, target)`: returns a fake exchange rate between two currencies.

3.2.2 Example Behaviour

When asked, for example, "What is the price of gold in EUR?", the agent should follow this reasoning process:

1. Calls `get_metal_price("gold")` to retrieve the price in USD.
2. Calls `get_currency_exchange("USD", "EUR")` to retrieve the exchange rate.
3. Combines both results to compute and return the final price in EUR.

[4] https://mistral.ai
[5] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[6] https://python.langchain.com/docs/integrations/text_embedding/huggingface

3.2.3 Implementation Notes

Initially, no explicit prompt was used, as the integration with LangChain[7] and LangGraph[8] exposes tool descriptions to the LLM, allowing it to decide when to use them. However, this approach proved unreliable: the agent often failed to invoke the tools correctly or hallucinated responses without accessing them. To address this issue, a custom prompt was introduced to guide the agent on how and when to use the available tools. After this change, the agent consistently used the tools as intended, and its performance improved significantly. The prompt used was:

```
You are a ReAct agent. Read carefully the dataset. For example:
"What is the price of METAL in CURRENCY?"

1. Always call get_metal_price with the name of the metal.
2. If the currency is not USD, call get_currency_exchange with
base="USD" and target=CURRENCY.
3. Use both results to compute and explain the final price.
4. If the currency is already USD, you don't need to call
get_currency_exchange.
```

3.2.4 First Evaluation Results and Observations

To validate the behaviour of the baseline ReAct agent, we conducted a series of evaluations focused on its ability to correctly use tools, return accurate values, and maintain efficiency across a diverse set of queries. The metrics considered include tool usage accuracy, output validation and number of reasoning steps.

The following figure compares the symbolic *Tool Accuracy* (exact match of calls) with the semantic *RAGAS ToolCallAccuracy*, which evaluates whether tools were used correctly.

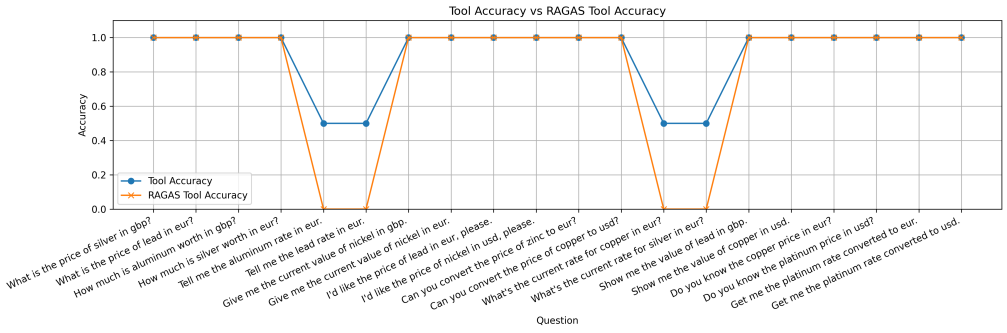

FIGURE 3.2: Tool Accuracy vs. RAGAS Tool Accuracy

As seen in Figure 3.2, while the agent performs well in most cases, a small subset of queries led to mismatches between expected and detected tool calls, which negatively affected the RAGAS score. When *Tool Accuracy* is defined as 0.5, that means the agent only invoked one of the two expected tools. In these cases, the agent either skipped the second tool call or replaced it with an incorrect one. This leads RAGAS to assign a score of 0, as the tool usage no longer aligns with the intended task structure.

[7] https://www.langchain.com/

[8] https://www.langchain.com/langgraph

The next plot reports whether the final output contained the expected keywords, and whether the extracted numerical values matched the expected value within a tolerance of 0.01.

As introduced in Section 2.3.1, the metric *Output Contains Expected* measures whether the final response includes the name of the queried metal and the target currency, ensuring that the agent has correctly interpreted and answered the question. On the other hand, *Value Correctness* checks whether the numerical result falls within an acceptable margin of the expected value, allowing for slight deviations due to rounding. Together, these two criteria provide a practical way to evaluate the accuracy and informativeness of the agent's final output.

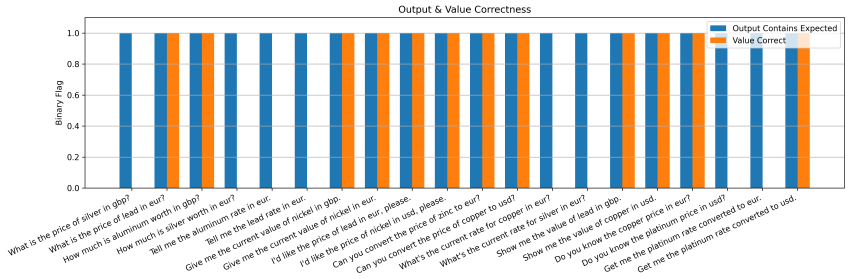

FIGURE 3.3: Output contains expected phrases and value correctness

Figure 3.3 reveals that most responses were complete and correct, though a few returned inaccurate values despite proper tool usage. In particular, there were cases where the agent selected and called the correct tools, but failed to compute or express the final result accurately. We further explored reasoning complexity by tracking the number of LLM steps per query and the total execution time required.

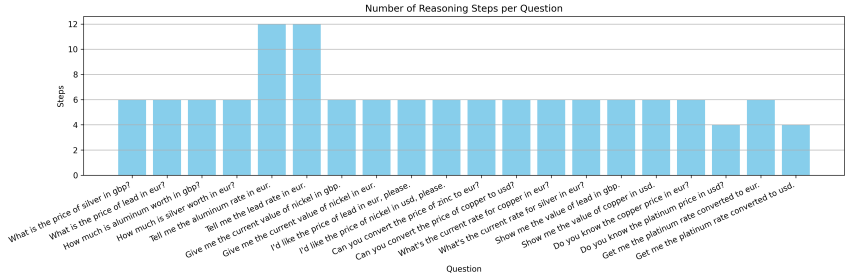

FIGURE 3.4: Number of Reasoning Steps per Query

As shown in Figure 3.4, the vast majority of queries required six reasoning steps, which aligns with the expected structure: initial input, internal thought, tool call, second thought, and final answer. However, a small number of queries triggered significantly more steps likely due to repeated tool calls or loops caused by the hallucination of the agent. On the other end, some queries ended early, either because the agent failed to complete the full reasoning loop or terminated prematurely due to hallucination.

3.2.5 Limitations and Targeted Improvements

Although the baseline ReAct agent demonstrated solid performance overall, the evaluation surfaced specific limitations that required refinement. In particular, some

responses exhibited minor numerical inaccuracies despite correct tool usage, and others showed inconsistent behavior when interpreting instructions from the initial prompt. To address these issues, two focused improvements were introduced:

- **Prompt refinement**: The original prompt was updated and simplified to include a more instructive guidance.

  ```
  You are a ReAct agent. Read carefully the dataset. For example:
  "What is the price of METAL in CURRENCY?":

  1) Call get_metal_price with METAL.
  2) Then call get_currency_exchange with base='USD',
  target=CURRENCY.
  3) Finally, respond combining both results.
  ```

- **Relaxed value comparison threshold**: The evaluation logic was adjusted to a higher tolerance when comparing numeric outputs to expected values. Specifically, the strict comparison `abs(num - expected_value) < 0.01` was relaxed to avoid false negatives caused by rounding. "'

These two modifications improved the agent's output correctness without altering its architectural structure. These are the results obtained, for the same inputs used before.

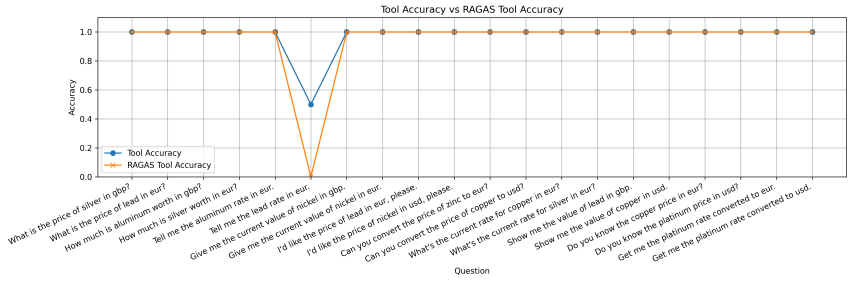

FIGURE 3.5: Tool Accuracy vs. RAGAS Tool Accuracy

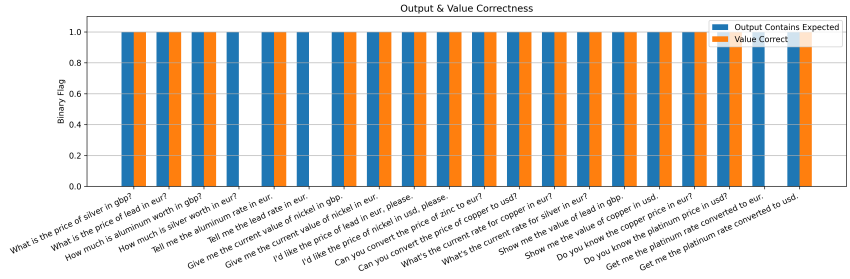

FIGURE 3.6: Output contains expected phrases and value correctness

Comparing with the previous model it can be seen in Figure 3.5 and 3.6 that the results have improved and that the agent fails less than before.

3.3 Agent 2: React Agent with Wikipedia and Wikidata Tools

This agent is designed to answer factual questions using external sources instead of relying only on the model's internal knowledge. It has access to two tools: one that gets short summaries of people from Wikipedia[9] , and another that finds out who the current president of a given country is using Wikidata[10] .

3.3.1 Available Tools

The agent is set up with two specific tools to get information from public sources:

- `get_summary_of(person)`: returns a short summary of a person by querying Wikipedia's API.
- `get_current_president_of(country)`: finds the name of the current president of a country by querying Wikidata.

3.3.2 Implementation Notes

At the beginning, we tried accessing the internet using a tool based on the DuckDuckGo API, but it didn't work well, so we switched to Wikipedia. There were also issues with the prompt, since the model was hallucinating. After testing different versions, we ended up keeping only the Wikipedia tool and the one for checking who the current president is. The final prompt used was:

```
You are a ReAct agent.  You must use tools to answer questions - do not assume you
know any answer beforehand.

If the question is like "Who is the president of COUNTRY?":
1) Call the tool ‘get_current_president_of‘ with the country’s name.
2) Use the tool’s output as your answer.

If the question is like "Give me a summary of PERSON":
1) Call the tool ‘get_summary_of‘ with the person’s name.
2) Use the tool output to generate a natural, fluent summary.
3) Your answer must be factually faithful to the tool output - do not invent or
include information from outside sources.

Example:
- Question:  "Give me a summary of Barack Obama?"
- Tool output:  "Barack Hussein Obama II is an American politician who (...)"
- Your response:  "Barack Obama served as the 44th U.S. president and was (...)"

Don’t call two tools at the same time, first call one and after the next.
Make sure your response is accurate, relevant, and based only on the tool result.
```

3.3.3 First Evaluation Results and Observations

Once created the agent, a smaller set of inputs was used for testing compared to the previous agent. The main reason is that the evaluation metrics being used this time require more processing time, then, it was necessary a reduced number of queries.

[9] https://en.wikipedia.org/api/rest_v1/
[10] https://www.wikidata.org/w/api.php

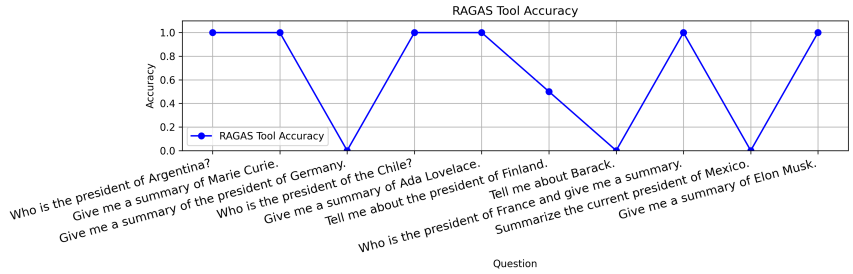

FIGURE 3.7: Tool Accuracy scores for each query using RAGAS.

As shown in Figure 3.7, some queries score the maximum value for tool accuracy, which means the agent selected and executed the correct tools in those cases. However, there are a clear failures where the score drops to zero. This happens with prompts where the agent doesn't identify the type of action that must be taken, mainly for those where more than a tool was need to be used.

Secondly, focusing on faithfulness (Figure 3.8), most responses are well aligned with the retrieved content. The majority of scores are 1.0, meaning the model did not introduce any unsupported information. However, there are a few notable exceptions that illustrate the behaviour of the metric.

In the case of *Who is the president of Argentina?*, even though the tool was correctly invoked and returned the right answer, the final response was not marked as faithful (score 0.0). This happened because the model rephrased the output.

A similar score of 0.0 appears in *Who is the president of France and give me a summary*, where the tool retrieved two correct fragments about Emmanuel Macron, but the model failed to use them and instead claimed it could not find any relevant information. Since the final answer ignored the retrieved content completely, faithfulness drops to the lowest possible value.

For *Tell me about Barack.*, the tool returned the summary of Obama's background, but the model generated a longer response. Even though all of this information is correct, it was not present in the retrieved content, so the metric scored 0.8125 .

Summarise the current president of Mexico receives a perfect faithfulness score of 1.0 despite the tool returning an error. This is because the model did not invent an answer but acknowledged the lack of retrieved content and responded accordingly, which the metric rewards as a fully faithful behaviour.

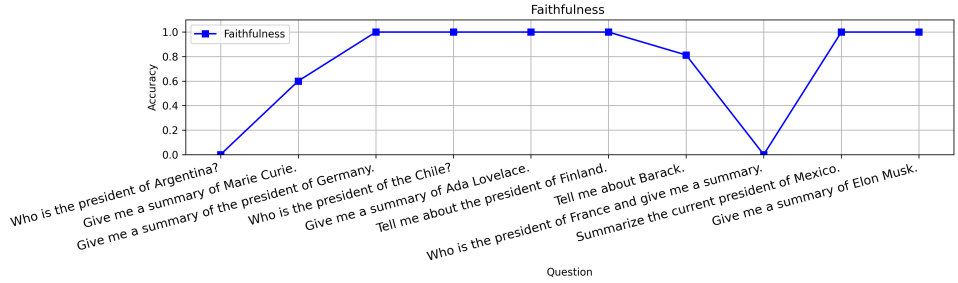

FIGURE 3.8: Faithfulness results for the evaluated queries.

Regarding answer relevancy (Figure 3.9), results are generally strong (above 0.75), suggesting that the majority of responses address the user's question effectively. For instance, answers about the presidents of Argentina, Germany or Chile receive high scores, indicating that the information provided is relevant and directly answers the input prompt.

However, there are clear drops in some cases. There are lower scores that appear in *Give me a summary of Marie Curie* and *Tell me about Barack* for instance. Although the model provides accurate historical facts it introduces additional details well beyond the retrieved biography. While informative, this extra content partially dilutes the focus of the response in relation to the original question.

The two lowest scores appear in *Who is the president of France and give me a summary* and *Summarise the current president of Mexico*. In the first, the model ignores the tool output and claims no information was found, which fails to address the question. In the second, the tool fails and the model gives a fallback response without answering the user's request. Although these replies may be faithful, they are not considered relevant as they do not fulfil the user's intent.

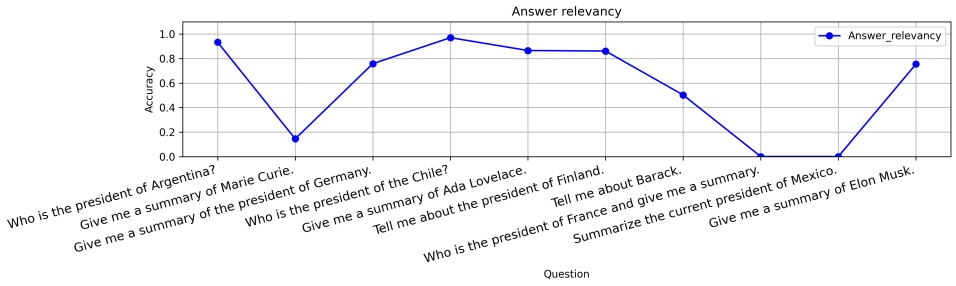

FIGURE 3.9: Answer Relevancy per query.

Finally, context precision (Figure 3.10) remains consistently high across almost all queries, indicating that the retrieved passages are generally well targeted and relevant to the user question. In most cases, the content used by the model aligns closely with the expected answer.

The only exception is *Summarise the current president of Mexico*, where the score drops due to a tool failure that returned no usable content. Notably, *Who is the president of France and give me a summary* also scores high, as the tool correctly retrieved relevant information. However, the model failed to use it, which affects other metrics but not context precision itself.

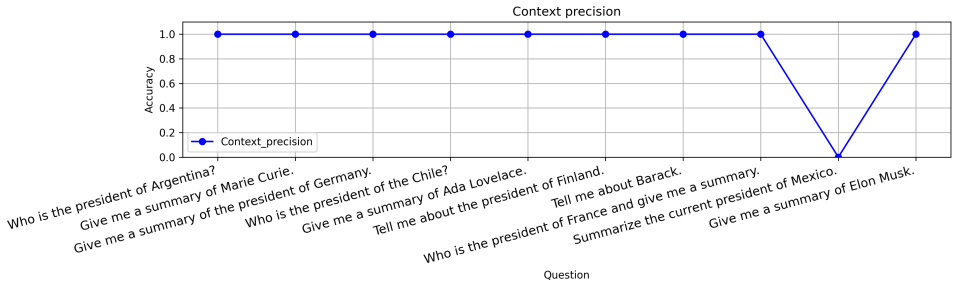

FIGURE 3.10: Context Precision score for each query using a single retrieved context.

3.3.4 Limitations and Targeted Improvements

In order to improve the answers and reliability of the agent a new prompt was designed. The main problem was that the agent was ignoring the tool answer and it was answering the query from its own memory, so the prompt had to be refined to face mainly that.

```
You are a ReAct agent. You must use tools to answer questions - do not assume you
know any answer beforehand.
If the question is like "Who is the president of COUNTRY?":
1) Call the tool ‘get_current_president_of‘ with the country’s name.
2) Use the tool’s output as your answer.
If the question is like "Give me a summary of PERSON":
1) Call the tool ‘get_summary_of‘ with the person’s name.
2) Use the tool output to generate a natural, fluent summary.
3) Your answer must be factually faithful to the tool output - do not invent or
include information from outside sources.
If the question is like "Give me a summary of the president of COUNTRY":
1) First, call the tool ‘get_current_president_of‘ using the country name.
2) Then, take the returned name (the president’s name) as a string.
3) Call the tool ‘get_summary_of‘ with that name.
4) Finally, use the result to write a fluent, factual summary based strictly on the
tool output.
IMPORTANT:
- Do not use the name of the tool as input.
- Only use the actual content returned by the tool (the person’s name). Example:
- Question: "Give me a summary of Barack Obama?"
- Tool output: "Barack Hussein Obama II is an American politician who was the 44th
president of the United States from 2009 to 2017..."
- Your response: "Barack Obama served as the 44th U.S. president and was the first
African American to hold the office..."

Example 2:
- Question: "Give me a summary of the president of Germany."
- First tool call: ‘get_current_president_of("Germany")‘
- First tool output: "Frank-Walter Steinmeier"
- Second tool call: ‘get_summary_of("Frank-Walter Steinmeier")‘
- Second tool output: "Frank-Walter Steinmeier is a German politician serving as
President of Germany since 2017. He previously served twice as Minister for Foreign
Affairs and as Chief of the Federal Chancellery. He is a member of the Social
Democratic Party of Germany."
- Your response: "Frank-Walter Steinmeier is a German politician who has been
President of Germany since 2017. He has also served as Minister for Foreign Affairs
and is a member of the Social Democratic Party."

Don’t call two tools at the same time - first call one, then the next.
IMPORTANT: Your final answer must only include facts that are *explicitly present*
in the tool output. Do not add anything, even if you know it is true. Do not infer,
expand, or include details from memory.

Make sure your response is accurate, relevant, and based only on the tool result.
Always make sure to wait for the result of each tool call before using its output in
the next step. Do not hardcode or reuse tool names as strings. Use only the content
returned by the tool as input for further reasoning or tool calls. result.
```

Once defined this second promt, the same metrics used before can be measured. Firstly, looking at the tool accuracy. As shown in Figure 3.11, the value for tool accuracy has improved compared to the previous one and the agent interprets correctly the queries.

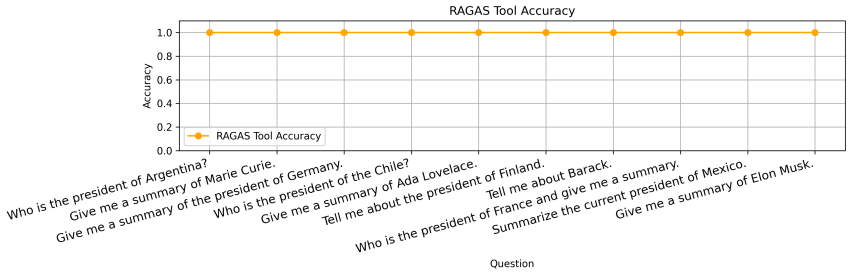

FIGURE 3.11: Tool Accuracy scores for each query using RAGAS.

Secondly, regarding faithfulness (Figure 3.12), the scores remain mostly high, except for *Give me a summary of Marie Curie*, where the agent hallucinates by relying on internal memory and ignoring the response retrieved from the tool. Since some adjectives match, the score is not completely 0. For the other queries, the agent provides factual information based on the retrieved context.

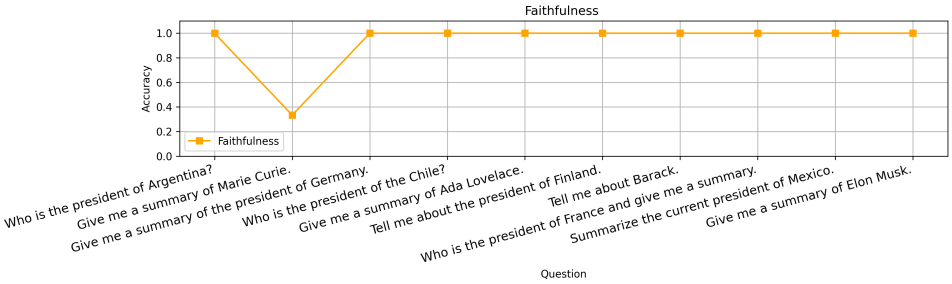

FIGURE 3.12: Faithfulness results for the evaluated queries.

In contrast, answer relevancy (Figure 3.13) shows slightly more consistency, with fewer extreme drops. The prompt adjustment seems to help the agent stay more focused on the user query.

Some cases, however, highlight certain limitations. In *Give me a summary of Marie Curie*, the response includes valid and factual information, but it adds content and partially shifts the focus, which results in a lower score (0.45). Similarly, *Tell me about Barack* also scores poorly (0.46), as the answer is overly general and lacks specific alignment with the retrieved tool output.

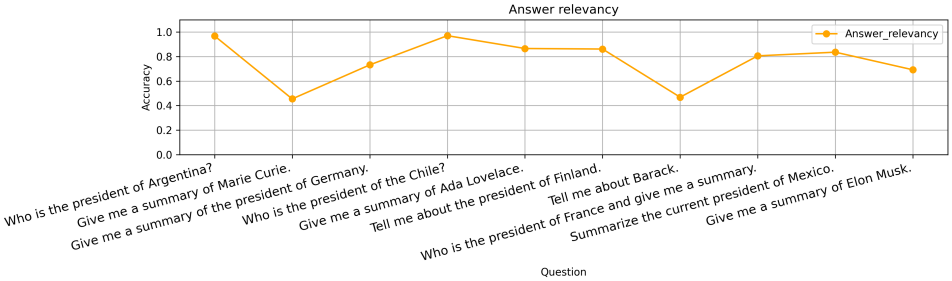

FIGURE 3.13: Answer Relevancy per query.

Lastly, as shown in Figure 3.14, context precision remains consistently high across all evaluated queries, with a perfect score of 1.0 in every case. This indicates that the passages retrieved by the tool are always relevant and correctly aligned with the user's question.

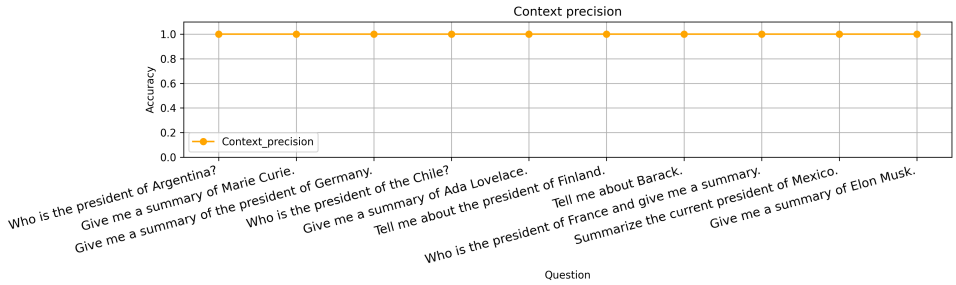

FIGURE 3.14: Context Precision score for each query using a single retrieved context.

3.4 Agent 3: Metal-Focused Agent with PDF, Finance and Wikipedia Tools

This third agent is meant to be a more complex assistant than the first one, this too being related to metals. It can do three main things: look up technical info about a metal (like its description, industrial uses and the 2023 reference price) from a local PDF; get the latest market price from Yahoo Finance [11] ; and fetch general background info from Wikipedia[12] . It's mainly useful for people like metal investors, students or engineers who want quick details about how metals are used, and even for chatbots in industrial or educational websites.

3.4.1 Available Tools

The agent includes the following tools:

- `get_metal_info(metal)`: returns the metal's description, industrial uses and its 2023 reference price by parsing a local PDF file.
- `get_metal_price_yfinance(metal)`: gets the latest market price of the metal using Yahoo Finance data, limited to a small set of supported symbols.
- `describe_metal(metal)`: retrieves a general description from Wikipedia, first trying the "(metal)" page and falling back to the base term if needed.

3.4.2 Implementation Notes

This is the prompt used for the agent:

```
You are a ReAct agent specialized in answering questions about metals.
You have access to the following tools:

1. ‘get_metal_info(metal_name)‘: Retrieves a technical description, industrial
uses, and the 2023 reference price for a specific metal from a local PDF document.
2. ‘get_metal_price_yfinance(metal_name)‘: Retrieves the most recent market price
(USD/ounce) of common metals like gold, silver, platinum, palladium, and copper.
3. ‘describe_metal(metal_name)‘: Provides a general encyclopedic description of a
metal from Wikipedia.

You must follow this step-by-step reasoning:
1. First, identify exactly which tool matches the user’s request.
- Use ‘get_metal_info‘ if the question refers to "the document", "technical data",
"description", "uses", or "2023 price".
- Use ‘get_metal_price_yfinance‘ if the question is about current or market price.
- Use ‘describe_metal‘ only if the user is asking for general knowledge.
2. Call the tool.
3. Then summarize or quote the tool output explicitly in your final answer. Do not
invent information. Do not skip this step.

Your answers must be clear and informative. Do not write anything until the tool has
responded. Always base your answers on the tool output.
```

3.4.3 First Evaluation Results and Observations

After the prompt is defined the agent is tested with 10 queries. It has been evaluated with the same metrics as the previous agent but, this time the metrics fit better as the answers are longer strings.

[11] https://pypi.org/project/yfinance/
[12] https://en.wikipedia.org/api/rest_v1/

The agent achieves high faithfulness scores in most cases (3.15), with values of 1.0. However, some drops reveal specific issues. *Describe gold* receives a faithfulness score of 0.5, as the model provides only a vague closing question ("Is there anything else...") instead of using the retrieved information. In *Can you give me the description and price of silver from the document?*, the score drops to 0.0 since the model generates a generic tool failure message, without any retrieved content, the answer cannot be evaluated as grounded.

What's the technical description of silver? receives a relatively low faithfulness score. Although the response is factually correct and related to the source, the way it is reformulated apparently weakens the traceability to the retrieved content lowering the faithfulness score.

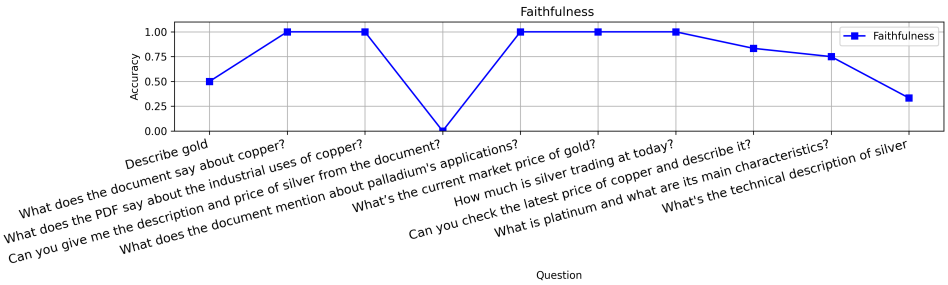

FIGURE 3.15: Faithfulness results for the evaluated queries.

As shown in Figure 3.16, answer relevancy scores are generally high, with most responses achieving values above 0.75. Although, *Can you give me the description and price of silver from the document?* scores below 0.5, as a tool failure prevented the model from retrieving any meaningful content, resulting in a fallback response that does not fully address the original intent. *Describe gold* receives a relatively high score (0.75) even though the response does not directly address the question. As the response stays on topic and refers implicitly to the requested entity, it is still considered thematically relevant, despite lacking informative value. This highlights one of the limitations of the metric when evaluating vague or evasive answers.

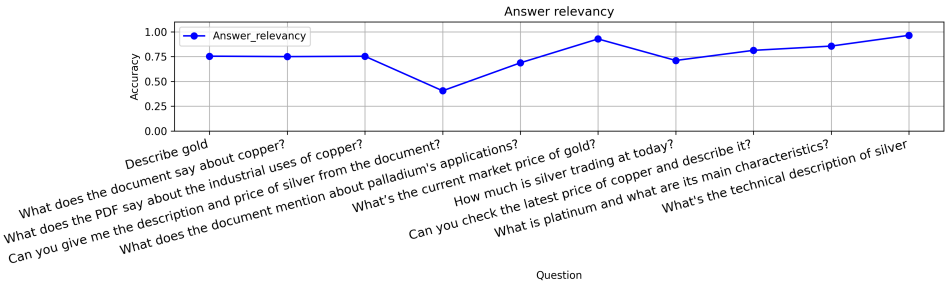

FIGURE 3.16: Answer Relevancy per query.

Context precision (3.17) is perfect for most queries. However, there are two clear exceptions. In *Describe gold*, context precision drops to 0.0 because, although the system retrieved a relevant definition of gold, the model did not use it at all in the response. The generated output simply asks a follow-up question without incorporating any retrieved facts, which breaks the link between context and answer.

A similar issue occurs in *Can you give me the description and price of silver from the document?*, where the tool fails and returns an error. Since no content is retrieved, the model generates a generic fallback message, and context precision is set to 0.0 due to the complete absence of usable input.

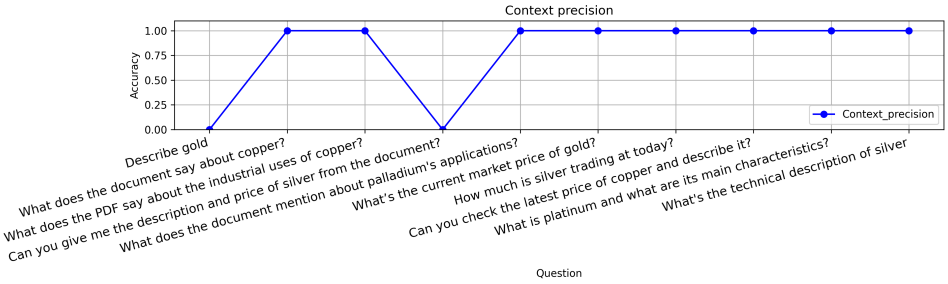

FIGURE 3.17: Context precision per query for a single retrieved context.

3.4.4 Limitations and Targeted Improvements

Looking at the answers received a new prompt was designed. The main goal of this new prompt was to ensure that the model would not answer the query using its internal knowledge. That rule has been emphasised many times along the prompt.

```
You are a ReAct agent specialized in answering questions about metals.
GOLDEN RULE: you MUST only use the information provided by the tools.  NEVER add
information by yourself.

Read carefully the output of the tool to generate a focused and concise response to
the specific question.  Do NOT just repeat or copy the entire tool result.

For example:
- If the question is only about industrial uses, your answer should only summarize
the "Industrial Uses" section from the tool output.
- If the question asks about the 2023 price, your answer should mention only the
price part from the tool.

Avoid including unrelated parts (e.g., don't mention the description if it wasn't
asked).  Be precise and stay on topic.

Summarize, don't copy-paste.  Your job is to extract the relevant information and
rephrase it clearly.

You have access to the following tools:
1.  'get_metal_info(metal_name)':  Retrieves a technical description, industrial
uses, and "2023 price" for a specific metal from a local PDF document.
2.  'get_metal_price_yfinance(metal_name)':  Retrieves the most recent market price
(USD/ounce) of common metals like gold, silver, platinum, palladium, and copper.
3.  'describe_metal(metal_name)':  Provides a general encyclopedic description of a
metal from Wikipedia.  You must follow this step-by-step reasoning:

1.  First, identify exactly which tool matches the user's request.
- Use 'get_metal_info' if the question is about the 2023 price, something from the
PDF, technical stuff, industrial uses, or if it says "according to the document".
- Use 'get_metal_price_yfinance' ONLY if the question says something like "current",
"today", or "latest market price".  Never use this tool for 2023 prices - it's wrong.
- Use 'describe_metal' only if the user is asking for general knowledge.
2.  Call the tool.
3.  Then summarize or quote the tool output explicitly in your final answer.  Do not
invent information.  Do not skip this step.

Your answers must be clear and informative.  Do not write anything until the tool has
responded.  ALWAYS base your answers on the tool output.
```

As shown, the prompt has been extended mainly to prevent hallucinations and to ensure it answers using the output from the tools rather than generating its own response. Once the new prompt has been applied these are the results obtained.

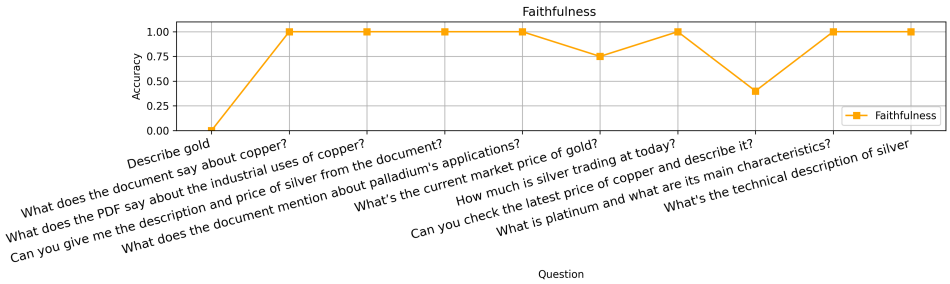

FIGURE 3.18: Faithfulness results for the evaluated queries.

The agent improves faithfulness (Figure 3.18) compared to the previous version. Some deviations are observed, such as in *Can you check the latest price of copper and describe it?* (faithfulness 0.4). Although the model does not introduce any unsupported information, the response reformulates and simplifies the retrieved content to the point where the connection to the original context becomes less explicit. This weaker traceability results in a lower score. Similar to the previous agent version, *Describe gold* gets a low score. Although relevant information was retrieved, the model ignores it and replies with a vague follow-up question.

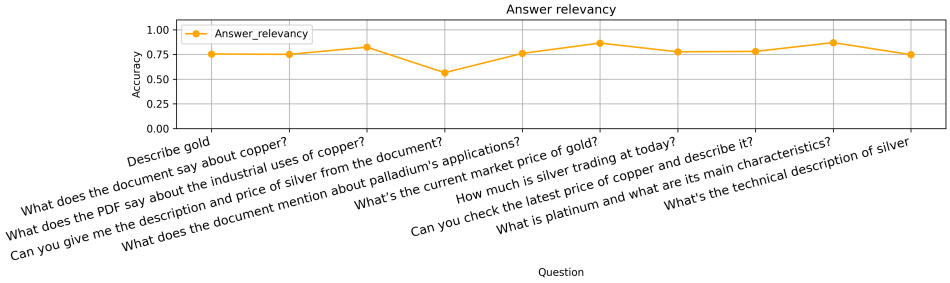

FIGURE 3.19: Answer Relevancy per query.

For answer relevancy (Figure 3.19), the agent achieves consistent performance across most queries. As happened in the previous case (Figure 3.16), *Describe gold* achieves a score around 0.75 despite not directly answering the prompt.

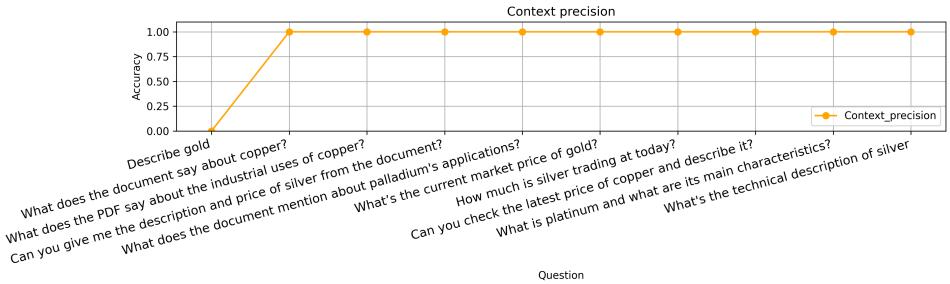

FIGURE 3.20: Context Precision per query

As seen in Figure 3.20, context precision remains perfect across nearly all questions, correcting previous errors except for *Describe gold*.

Chapter 4

Conclusions

This thesis explored the evaluation of ReAct agents with access to external tools, using both custom rules and RAGAS metrics to assess tool usage, answer relevance, factual consistency and context exploitation.

Evaluation metrics

As agents are still a relatively new paradigm in AI, the need to evaluate them properly has become increasingly important. Unlike traditional language models, agents operate through multi-step reasoning, tool usage and memory, which introduces new dimensions to assess. However, the field of agent evaluation is still in its early stages, and most existing methods are not specifically designed for agentic workflows. As a result, current solutions are limited and often rely on either manual inspection or metrics that were originally built for static question answering systems. This creates a gap between how agents behave in practice and how well we are able to measure their performance.

On the one hand, the most established option for evaluating agents so far is the use of RAGAS metrics. In this work, we applied several of them: *Faithfulness, Context Precision, Answer Relevancy* and *Tool Accuracy*, alongside custom rule-based metrics. Each metric provides a different perspective on the agent's behaviour.

Answer Relevancy has been useful to verify whether the response stays on topic, but it does not guarantee that the content is factually correct. It confirms that the agent understood the theme of the question, but not whether the information provided is accurate. Answer Relevancy is therefore best interpreted with caution: because it measures topical alignment rather than factual accuracy, its score can be high even when the answer is wrong, or low even when the answer is correct, depending on how closely the wording of the response mirrors the wording of the question.

Faithfulness, in contrast, is more helpful when trying to detect hallucinations, since it checks whether the claims in the answer are supported by the retrieved context. Still, if the agent answers from its own memory and happens to align with the context by coincidence, the score can be misleading.

Context Precision could be a powerful metric in scenarios where multiple documents or passages are retrieved, as it measures how focused the retrieved information is. However, in our case, since only one chunk is used per query, the metric simplifies to a binary outcome (0 or 1), which limits its usefulness. Tool Accuracy, by contrast, proved to be particularly valuable, as it allowed us to track the reasoning process step by step and identify exactly where the agent failed or chosed the right tool, passing the correct arguments or combining the results properly.

On the other hand, in addition to RAGAS, custom rule-based metrics were developed for Agent 1 to evaluate specific aspects of task completion. These included checks for correct tool selection, presence of the metal name in the output, and numerical accuracy of the returned price. These metrics worked very well for structured tasks involving numerical calculations or clearly defined entities. However, they are only applicable in scenarios where the expected output can be precisely defined. For more open or descriptive responses, such as summaries or general explanations, these checks become less useful and cannot be validated through exact matching or numeric thresholds.

Problems faced

Apart from evaluation metrics, one of the main takeaways is the importance of prompt design. For both Agent 2 and Agent 3, the tools were called correctly from the beginning, but the model often ignored the tool output or generated an answer from memory. Iterating on the prompt was necessary to make the agent incorporate the retrieved content and remain grounded. In Agent 3, this was particularly important when using domain-specific data from PDFs or financial APIs.

Today, prompt engineering is gaining increasing attention and demand. Being as precise and specific as possible when interacting with a language model has become essential to ensure useful and grounded outputs. It is a growing field, and this project has highlighted its importance in practice. Despite having a correct architecture and working tool integrations, the agents still failed to produce reliable answers until the prompt was carefully refined. In this case, prompt engineering was the main strategy used to fix issues and improve the overall behaviour of the system. As models become more capable but also more sensitive to phrasing, this area is rapidly evolving and opening up new professional opportunities in both research and industry.

Hallucinations were one of the most frequent and difficult problems during the project. Even when the agent used the correct tools, it often ignored the tool output or gave answers based on its own memory. Many of these responses sounded correct but were not really connected to the retrieved information. Fixing this required many small tests and changes until the agent started working as expected. Although it may seem simple in theory, improving and debugging an agent like this was much harder than it looked. Even so, avoiding hallucinations completely is still very difficult, especially in open or general questions.

Another key insight from this project is that LLM-based agents tend to perform poorly when dealing with generic tasks, such as accessing the internet or answering broad questions. In these cases, hallucinations are far more likely, and achieving reliable behaviour requires substantial prompt tuning and validation. In contrast, when the task is narrowly defined and the tools provide clear, structured outputs, the agent performs much more reliably. This suggests that LLM agents are currently best suited for highly specific use cases where the reasoning path and expected output can be tightly controlled.

Future work

Looking ahead, one of the biggest challenges is making sure that an agent can be used in real applications and still behave reliably. In practice, it's very hard to guarantee that it will work well 100% of the time. What is usually done in these cases is to define a small set of key queries that must always work. This way, you can make changes or improvements without breaking the most important behaviours.

As for evaluation, while the metrics used in this project were useful, there are other tools worth exploring. One of them is Giskard[1], which allows you to write tests for your model, check for hallucinations or risky outputs and make sure it behaves as expected in different situations. Using something like that would make it easier to catch problems before putting the agent into production.

In general, what's needed going forward is a more complete way to build and test these agents. That includes writing better prompts and learning how to write good ones, defining what they're supposed to do and checking carefully how they behave. As these systems get more complex, having a solid process for improving and testing them becomes even more important.

Overall, the most time-consuming challenge was getting the agent to generate the correct final answer after using the tools. Once tool calls were reliable, the final reasoning step became the main source of error. Future work should focus on improving how tool outputs are used in the response, and on finding better ways to avoid hallucinations without limiting the agent too much.

[1] https://www.giskard.ai/

Bibliography

Amazon Web Services (2024). *What is an AI Agent?* Accessed: 2025-06-26. URL: https://aws.amazon.com/what-is/ai-agents/?nc1=h_ls.

DeepLearning.AI (2024a). *Agentic Design Patterns Part 4: Planning*. Accessed: 2025-05-27. URL: https://www.deeplearning.ai/the-batch/agentic-design-patterns-part-4-planning/.

— (2024b). *Agentic Design Patterns Part 5: Multi-Agent Collaboration*. Accessed: 2025-05-27. URL: https://www.deeplearning.ai/the-batch/agentic-design-patterns-part-5-multi-agent-collaboration/.

Fauscette, Michael (2024). *Agentic AI vs LLMs: Understanding the Shift from Reactive to Proactive AI*. Accessed: 2025-04-25. URL: https://www.arionresearch.com/blog/agentic-ai-vs-llms-understanding-the-shift-from-reactive-to-proactive-ai.

Google Cloud (2025). *What are AI agents? Definition, examples, and types*. Web page. Accessed: 2025-05-07. URL: https://cloud.google.com/discover/what-are-ai-agents.

Jiang, Zi et al. (2023). "Mistral: Efficient Open-Weight Language Models". In: *arXiv preprint arXiv:2310.06825*.

Microsoft (2024). *Tool Use*. Accessed: 2025-05-28. URL: https://microsoft.github.io/ai-agents-for-beginners/04-tool-use/.

OpenAI (2023). *GPT-4 Technical Report*. Accessed: 2025-05-26. URL: https://openai.com/research/gpt-4.

RAGAS (2024). *RAGAS Documentation*. https://docs.ragas.io. Accessed: 2025-05-26.

RAGAS: Answer Relevance Metric (2024). Accessed: 2025-05-10. URL: https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/answer_relevance/#response-relevancy.

RAGAS: Context Precision Metric (2024). Accessed: 2025-05-10. URL: https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/context_precision/#example.

RAGAS: Faithfulness Metric (2024). Accessed: 2025-05-10. URL: https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/faithfulness/.

RAGAS Metrics for Agents: Tool Call Accuracy (2024). Accessed: 2025-05-10. URL: https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/agents/#tool-call-accuracy.

Touvron, Hugo et al. (2023). "LLaMA: Open and Efficient Foundation Language Models". In: *arXiv preprint arXiv:2302.13971*.

Vidhya, Analytics (2024). *Agentic AI: Reflection Pattern*. Accessed: 2025-05-27. URL: https://www.analyticsvidhya.com/blog/2024/10/agentic-ai-reflection-pattern/.

Wu, Juncheng et al. (2025). *Knowledge or Reasoning? A Close Look at How LLMs Think Across Domains*. arXiv preprint. arXiv:2506.02126v1. URL: https://arxiv.org/pdf/2506.02126v1.

Wu, Qingyun et al. (2023). *AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation*. arXiv: 2308.08155 [cs.AI]. URL: https://arxiv.org/abs/2308.08155.

Yao, Shunyu et al. (2023). *ReAct: Synergizing Reasoning and Acting in Language Models*. arXiv: 2210.03629 [cs.CL]. URL: https://arxiv.org/abs/2210.03629.