HW 3
Implementing Language Model Agents

CS 678 Advanced Natural Language Processing (Fall 2025)
https://nlp.cs.gmu.edu/course/cs678-fall25/

OUT: Oct 14, 2025

DUE: Nov 18, 2025

TOTAL CREDITS: 100 Points
(contribution to your final grade = Points earned $\times$ 20%)

Your name: Riya Elizabet Stanly

Your GID: G01521289

Overview: In this assignment, you will learn to build language model (LM) agents using a Python library called `smolagents`. The assignment includes two parts. In Part 1, you will learn smolaents by building two example agents and answering questions that help you understand how they work. In Part 2, you will then build your own LM-based tool-augmented agent using this library.

Computational Resources: The agent library is pretty lightweight, so it is feasible (at least for Part 1, assuming a recent laptop) to complete the assignment on your personal laptop. However, in any case, GMU's ORC can be a good resource (https://wiki.orc.gmu.edu/mkdocs/Getting_an_ORC_Account/).

Submission Guideline: Like in previous assignments, complete this LaTeX project and compile a PDF.

- Submit the PDF to Gradescope.
- Submit your agent implementation (Part 2) in a .zip file to Canvas.

Important Notes: All answer blocks are set with fixed width and length, and the space should be enough. Do NOT modify them unless you are instructed to (or otherwise your assignment may not be graded properly). Like in all other assignments, AI tools such as ChatGPT and CoPilot are disallowed and violate the university's Academic Standards.

Part 1: Introduction to LM Agents (60 points)

In this assignment, you will learn about LM agents by using an open-source Python library called smolagents. This is a library developed by Hugging Face, which allows you to build an agent in lines of code. In this part, you will follow the assignment instructions to grasp basic concepts about agents. To get started, follow the commands here to install the smolagents library (including `[toolkit, litellm]` as the extras).

smolagents implements two agent types: `CodeAgent` and `ToolCallingAgent`. Both agents are designed to enable tool-calling, but they achieve it in different ways: CodeAgent generates a Python code snippet, while the ToolCallingAgent generates a structured JSON snippet. See here for definitions and comparisons between these two agent types. In this assignment, we will only use CodeAgent.

Finally, smolagents support different LM backends, ranging from API-accessible LMs from OpenAI/Anthropic/etc. to locally deployed, open-weight LMs. See here for instructions for switching the backend.

In this part, you will explore both a vanilla agent and an agent with tool augmentation, called a "tool-augmented agent". The two kinds of agents differ in whether they have access to a pool of available tools. Since we will build CodeAgents, you will see that the agent invokes tools by generating Python code that calls the function interface of the tool. The following questions will get you familiar with some of the smolagents functions, but you may want to check for more from its documentation.

Q1 (40 points): Build a Vanilla Agent

We will start by building a vanilla agent without tool support.

Q1.1 (5 points): Vanilla agent setup. Run the following code. If your environment is set up properly, you should receive a valid output generated by the agent in one step.

```python
from smolagents import CodeAgent, InferenceClientModel

# Initialize a model (using Hugging Face Inference API)
model = InferenceClientModel()  # Uses a default model

# Create an agent with no tools
agent = CodeAgent(tools=[], model=model)

# Run the agent with a task
result = agent.run("Calculate the sum of numbers from 1 to 10")
print(result)
```

Note: InferenceClientModel utilizes Hugging Face (HF)'s facility, which further invokes services from different inference providers for fast inference. Inference Providers need an `HF_TOKEN` to authenticate, but a free HF account already comes with included credits. You need to set the environment variable `HF_TOKEN` or pass the `token` variable upon initialization of InferenceClientModel (i.e., `token= "<YOUR_HUGGINGFACEHUB_API_TOKEN>"`). You can get your token from your HF settings page. Since the free HF account comes with limited credits, make sure to complete Q1 before you do other random exploration.

What Python code does the agent generate? Copy what you observed below:

Your Answer

```python
total_sum = 0
for number in range(1, 11):
    total_sum += number
print(total_sum)
```

What is the LM backend of this agent?

Your Answer

InferenceClientModel – Qwen/Qwen2.5-Coder-32B-Instruct

Record the token counts below:

Input tokens	Output tokens
4,247	174

Q1.2 (10 points): Telemetry setup. Agent runs are complicated to debug. smolagents solves this problem by integrating a telemetry instrument for agent inspection. Follow the instructions here to install the telemetry with Phoenix. Set it up for the vanilla agent and re-run the agent on the same question.

Now, in the graphical interface, you should be able to see your agent run. Under "Step 1", click "InferenceClientModel.generate", and you should see the input and output of the LM. Show the output message generated by the LM:

Your Answer

```
"output": {
    "value": "{\"role\": \"assistant\", \"content\": \"Thought: I will
     use Python code to calculate the sum of numbers from 1 to 10 and
    then return the final answer using the `final_answer` tool.\\n\\n<
    code>\\nresult = sum(range(1, 11))\\nfinal_answer(result)\\n\", \"
    tool_calls\": [], \"token_usage\": {\"input_tokens\": 2036, \"
    output_tokens\": 55, \"total_tokens\": 2091}}",
    "mime_type": "application/json"
  },
```

What is the agent doing with "Thought" and "`<code>`"?

Your Answer

The "Thought" section shows the agent's internal reasoning process where it explains, in natural language, how it plans to solve the task, while the "`<code>`" block contains the actual Python code that the agent generates and executes to perform the computation. In this case, the agent thinks about using Python's 'sum()' function to calculate the total from 1 to 10 and then uses the '`final_answer()`' function to return the result. Essentially, "Thought" represents the model's planning step, and `<code>` represents the executable action it takes to complete the task.

Finally, under "Step 1", you should also see a *tool-calling* record from the agent, and the "tool" corresponds to the "`final_answer`" function. For CodeAgents in smolagents, all agents produce an answer by calling the "`final_answer`" function, which returns the argument (e.g., the result from executing "`sum(range(1,11))`" as output to the user.

Q1.3 (15 points): Exploring the capability of vanilla agent. Now that we have set up the vanilla agent, let's test it on more questions!

1. Run the agent on the question "What time is it now?" Use the telemetry instrument to inspect the agent run. Describe what you observed. Does the agent answer the question successfully? Why does the agent need multiple steps in answering this question? (5 points)

Your Answer

Describe the observation below. The agent spent 2 steps in answering this question.
In step 1, it gave the following thought:

```
Thought: I will import datetime to get the machines current local
   time, format it, and then return it.
```

It then generated a code:

```
from datetime import datetime
now = datetime.now().strftime("%H:%M")
print(now)
```

In step 2, it gave the following thought:

```
Thought: Use the printed value as the final answer.
```

It then generated a code:

```
   final_answer(now)
```

(Add more steps if needed. Does the agent answer the question correctly? Reflect on what you observed and discuss why the agent has the multi-step behavior.)

The agent answered correctly (my run's final output was 19:34). It used multiple steps because CodeAgent follows a Thought → Code → Observation loop and typically computes first, then calls the built-in final answer tool in a separate step to return the result.

2. Run the agent on a factoid question: "Tell me anything about George Mason University". Similarly, use the telemetry interface to inspect the agent run. Does the agent answer the question correctly? Describe what you observed and discuss why the agent has the multi-step behavior. (5 points)

Your Answer

Describe the observation below. The agent spent 3 steps in answering this question.
In step 1, it gave the following thought:

```
Thought: To get information about George Mason University, I'll use a
    web search to find details about the university. Then I'll
    process the results to extract interesting information.
```

It then generated a code:

```
import re

# Perform a web search for George Mason University
gmu_info = web_search("George Mason University")

# Extract key points
# This is a simple regex approach to extract sentences containing
    interesting keywords
keywords = ["location", "established", "undergraduate", "graduate", "
    enrollment", "programs", "notable"]
for keyword in keywords:
    matches = re.findall(rf'\b{keyword}\b.*?[.!?]', gmu_info, re.
    IGNORECASE)
    for match in matches:
        print(match.strip())
```

In step 2, it gave the following thought:

```
Thought: Since the `web_search` function is not available, I'll use
    the `wikipedia_search` function to get information about George
    Mason University.
```

It then generated a code:

```
   gmu_info = wikipedia_search(query="George Mason University")
    print(gmu_info)
```

In step 3, it gave the following thought:

```
Since neither `web_search` nor `wikipedia_search` is available, I'll
    manually provide some information about George Mason University
    based on my knowledge.
```

It then generated a code:

```
   gmu_info = (
    "George Mason University (GMU) is a public research university
    located in Fairfax County, Virginia, United States. "
    "Established in 1957, it has grown significantly and is known for
    its innovative programs and diverse student body. "
    "The university offers a wide range of undergraduate and graduate
    degrees across various fields including business, "
    "science, engineering, and technology. As of 2021, George Mason
    University had an enrollment of approximately "
    "36, 000 students. It is particularly noted for its cybersecurity
    programs, which are ranked among the best in the nation."
    )
    final_answer(gmu_info)
```

Your Answer

(Add more steps if needed. Does the agent answer the question correctly? Reflect on what you observed and discuss why the agent has the multi-step behavior.) The agent spent 3 steps. In Step 1, it attempted to use `web_search("George Mason University")` but failed since no tools are available in the vanilla agent. In Step 2, it tried `wikipedia_search` and failed again. In Step 3, it generated its own summary of George Mason University and returned it using `final_answer(gmu_info)`. The response was partially correct but based only on internal knowledge. The agent's multi-step behavior follows the Thought–Code–Observation cycle: it planned tool use, handled errors, and adapted to produce an answer without external tools.

3. Let's run just one more question: "Pull out Prof. Ziyu Yao's latest 10 papers". For your reference, you can find the correct list of papers on Google Scholar. Does the agent answer the question correctly? Discuss the *hallucination* problem with this agent and why it happens. (No need to copy the exact agent thoughts and code for this question.) (5 points)

Your Answer

The agent listed ten plausible but incorrect paper titles such as "Towards Deeper Graph Neural Networks" and "Graph Hypernetworks." These are hallucinated results, since the vanilla agent lacks access to Google Scholar or any external data. It generates realistic-sounding outputs using only its internal training knowledge without factual verification. The hallucination occurs because the agent cannot retrieve real-time information, relying instead on language patterns it has learned.

Q1.4 (10 points): Inspecting the input prompt. So far, we have mainly focused on the output of the agent, how it thinks about a question step by step based on the real-time observation and how it generates code as a solution. But how does it do that?

Click the Step 1 "InferenceClientModel.generate" of the first agent run (for question "Calculate the sum of numbers from 1 to 10"). Under "LLM Input", check out the *system prompt* input to the backend LM. If you can't figure out a way to visualize the prompt, the same one can be found on smolagents's open-source library.

The exact system prompt is long. Do not copy it. Use your language to summarize what is included in the prompt. Make sure your summary clarifies how the agent is instructed to show the following behaviors:

- How is it instructed to give a "Thought" before generating code?
- How does it know how to call a tool and what tools are available?
- We have not talked about it, but smolagents also supports multi-agent orchestration. How does it achieve that?
- Any other tricks for the agent to behave well?

Your Answer

The system prompt guides the agent to follow a structured Thought–Code–Observation loop, where it first explains its reasoning in a "Thought" step and then writes executable Python code between code tags. It lists available tools as callable Python functions, instructing the agent on how to use them with proper arguments. The prompt also supports multi-agent orchestration by describing how the agent can assign tasks to "team members" through function-style calls with task descriptions and arguments. Additionally, it enforces behavioral rules—such as using only defined variables, approved imports, and persistent state across steps—to ensure safe, logical, and consistent execution.

Q2 (20 points): Build a Tool-augmented Agent

A tool-augmented agent is simply a vanilla agent that can access pre-specified tools.

Q2.1 (10 points): Tool-augmented agent setup. Next, we will set up a tool-augmented agent that has access to the DuckDuckGoSearchTool. This is one of the built-in tools in smolagents so we can directly add it to the tool pool of the agent without further engineering. However, open-weight LMs such as the default one we have been using so far often have difficulty in parsing and understanding long outputs generated by the search engine. Modify the following code and replace the default model with a "`gpt-4.1-nano`" model from OpenAI (you will need to set up an OpenAI API key; **making sure that you don't include the key in your submitted PDF or code**).

```python
from smolagents import CodeAgent, InferenceClientModel, LiteLLMModel,
    DuckDuckGoSearchTool

model = None # TODO: modify this line
agent = CodeAgent(
    tools=[DuckDuckGoSearchTool()],
    model=model,
)

# Now the agent can search the web!
result = agent.run("What is the current weather in Paris?")
print(result)
```

Due to the randomness of output sampling, the GPT-powered agent sometimes can answer the question in one step, but more often it needs multiple steps. You may need to execute the script for multiple times, till you can observe a multi-step agent trajectory.

Describe what you observed by inspecting the thought and code generated by the agent in each step. What is the agent doing when it needs multiple steps to answer the question? Discuss any behavioral patterns you observed. Do you see anything wrong with the agent?

Your Answer

The tool-augmented agent took **two steps** to answer the question "What is the current weather in Paris?". In the first step, it reasoned that it should perform a web search and used the `web_search("current weather in Paris")` command, retrieving multiple snippets from weather websites. In the second step, it analyzed those results, inferred that the weather was clear with a temperature of around 16°C, and produced the final output "Clear, 16°C". This multi-step behavior reflects the agent's reasoning process—first gathering external information using a tool, then synthesizing it into a concise answer. While the response is plausible, it may not always be accurate due to unstructured or outdated search data, showing that the agent's reliability depends on the quality and clarity of the web search results.

Q2.2 (10 points): Understanding how the tool-augmented agent works. Answer the following questions to gain a deeper understanding of the implementation of the tool-augmented agent.

1. Inspect the input prompt in each step. Other than the system prompt, what other information is sent to the LM? (5 points)

Your Answer

Other than the system prompt, the language model also receives several key pieces of contextual information during each step. This includes the **user's task or question** (e.g., "What is the current weather in Paris?"), the **agent's previous thoughts and code executions**, and the **observations** produced by running those code snippets (such as search results or print outputs). These components collectively form the step-by-step context that guides the LM's reasoning and next action. Therefore, each step's input prompt to the LM combines the system prompt, task description, previous thoughts, code, and observations, enabling iterative reasoning and tool use.

2. How does the agent understand the meaning of each tool, its corresponding function name, the available arguments, etc.? Hint: look at the system prompt carefully and find the corresponding source code implementation from smolagents' GitHub repository. (5 points)

Your Answer

The agent understands tools because the system prompt provides Python-like definitions for each one, including the function name, arguments, and description of what it does. These definitions are automatically added before the model runs, allowing it to learn how to call each tool correctly. The smolagents library also keeps a tool registry that links these names to real Python functions, so when the model calls a tool, it is executed and the result is returned for the next reasoning step.

Part 2: Build Your Own Agent (40 points)

Your showtime! Use the smolagents library to develop your own *tool-augmented* LM agent.

Requirements:

- Build something meaningful, something useful to yourself or others, something you would enjoy and would proudly share on your personal profile.
- The smolagents library implements a lot more functions than what we touched on in the previous questions, such as customized tool implementation (self-implemented or from MCP servers), multi-agent orchestration, human-in-the-loop agent planning, etc. The documentation has also included examples such as building RAG (Retrieval-Augmented Generation) agents, building Web browsing agents, etc. You are encouraged to explore them and build your agent on top of them.
- The grading will be based on how carefully and thoughtfully you approach this part, as well as your actual coding effort. For example, simply copying the example agent implementation from the smolagents documentation, or implementing only a trivial extension of the example agent (say, adding one built-in tool to the agent's configuration), will NOT give you the credit. At a minimum, we expect you to build a tool-augmented agent with at least one self-implemented tool. Other explorations are also welcome as long as they involve a reasonable amount of coding effort and show your care and thought.

Describe your agent below. (For questions in this part, you are allowed to increase the height of the answer box if you need more space.)

1. What agent did you implement? What is your motivation, and how is the agent meaningful/useful?

Your Answer

Agent Implemented: AI Term Explainer Agent
I implemented a tool-augmented `CodeAgent` that explains any AI/ML concept at three difficulty levels (Beginner, Intermediate, Expert). The agent is centered around a self-implemented tool, the `DefinitionExpanderTool`, which retrieves Wikipedia summaries and contextual information (related terms, categories) using the Wikipedia REST API.
Motivation: When learning AI/ML, students often encounter unfamiliar terms but face difficulty finding simple, structured explanations. My agent bridges this gap by producing layered explanations that adapt to different expertise levels.
Usefulness:

- Helps students quickly understand unfamiliar concepts.
- Supports multi-step reasoning (definition $\rightarrow$ context $\rightarrow$ synthesis).
- Enables interactive refinement through human-in-the-loop feedback.

The agent provides a practical educational tool that can be expanded into a study assistant or integrated into course material.

2. Please give more details about your implementation. Highlight the additional coding work you perform if the agent is developed on top of an example agent on smolagents's documentation.

Your Answer

My system consists of three main components: (1) a custom tool module, (2) the tool-augmented `CodeAgent`, and (3) an interactive user interface.

1. Custom Tool Implementation (self-developed work). I implemented a new Python module `custom_tools.py` containing the tool:

- `fetch_wikipedia_definition(term: str)` uses the official Wikipedia REST API to retrieve a definition, cleans/truncates the output, and returns a safe JSON string.
- `get_term_context(term: str)` fetches related categories and keywords to improve contextual reasoning.

Both functions use the `@tool` decorator from `smolagents` and return structured content for tool-calling.

2. Agent Implementation. I built the agent using:

- `CodeAgent` (not ToolCallingAgent) to allow the model to generate Python code that invokes tools in multi-step sequences.
- A chosen LLM backend (`gpt-4.1-nano` through LiteLLM or alternatively Qwen2.5 through HF Inference).
- `max_steps=5` to enable the sequence: search → context → explanation → refinement.

3. Interactive Mode and HITL. The file `main.py` contains an interactive loop that:

1. asks the user for an AI/ML term,
2. triggers the agent to fetch a definition and generate three explanation levels,
3. asks the user whether refinement is needed,
4. performs a follow-up reasoning step if requested.

Additional Coding Effort beyond documentation examples:

- Full custom Wikipedia integration (search + summary).
- Robust JSON cleaning, error handling, and truncation logic.
- Multi-step explanatory synthesis (Beginner/Intermediate/Expert generation).
- Human-in-the-loop feedback mechanism.

3. If you are given more time on this agent, how do you plan to improve it?

Your Answer

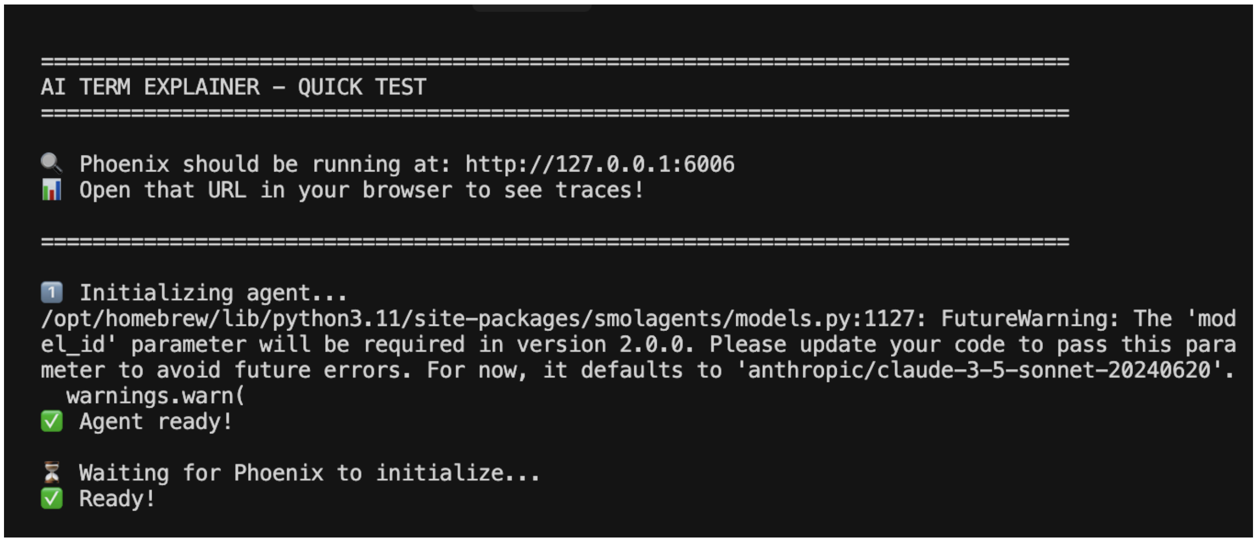
If given more time, I would improve the agent in several meaningful directions:

- **Add RAG (Retrieval-Augmented Generation):** Integrate a vector database (FAISS or Chroma) so the agent can retrieve definitions from multiple sources, not only Wikipedia.
- **Add multi-agent collaboration:** Introduce a "Reviewer Agent" that critiques explanations and a "Teacher Agent" that adds examples or quizzes.
- **Improve factual reliability:** Add verification steps using multiple sources or cross-checking definitions.
- **Caching layer:** Avoid re-querying Wikipedia for repeated terms.
- **Web UI:** Build a Streamlit or Gradio interface for polished user interaction.
- **Expanded toolset:** Add tools for math rendering, diagram generation, or dataset lookup.

These enhancements would make the system more robust, interactive, and suitable for deployment as a real AI-learning assistant.

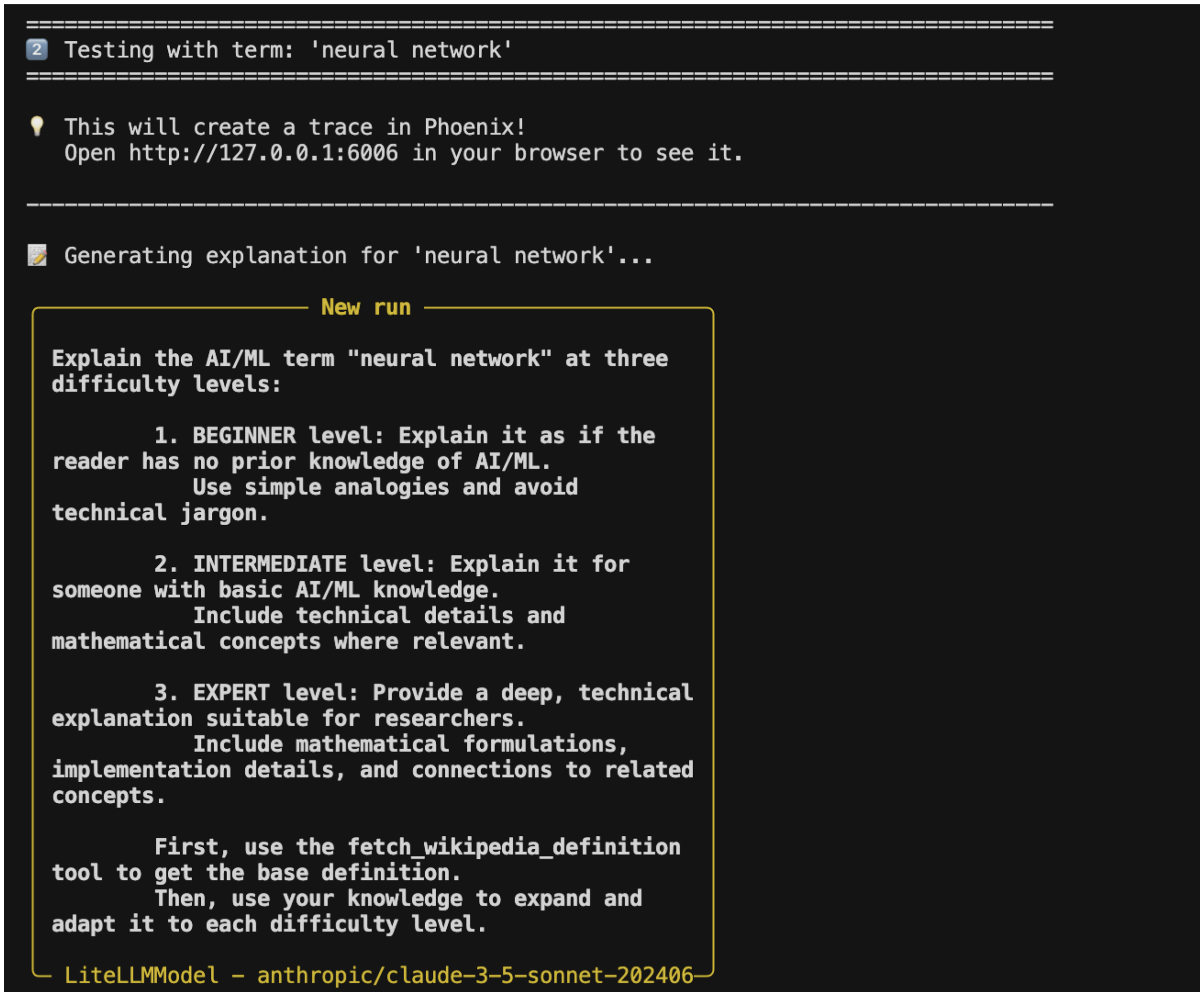
4. Use as much space as you want to add screenshots showing how your agent works. Show a few examples and add necessary textual descriptions to help the grader understand what you built. A basic code for inserting figures:

```
    \begin{figure}[h!]
        \centering
        \includegraphics[width=\textwidth]{your_image_file.jpg}
        \caption{This is the caption for your figure.}
        \label{fig:your_label}
    \end{figure}
```



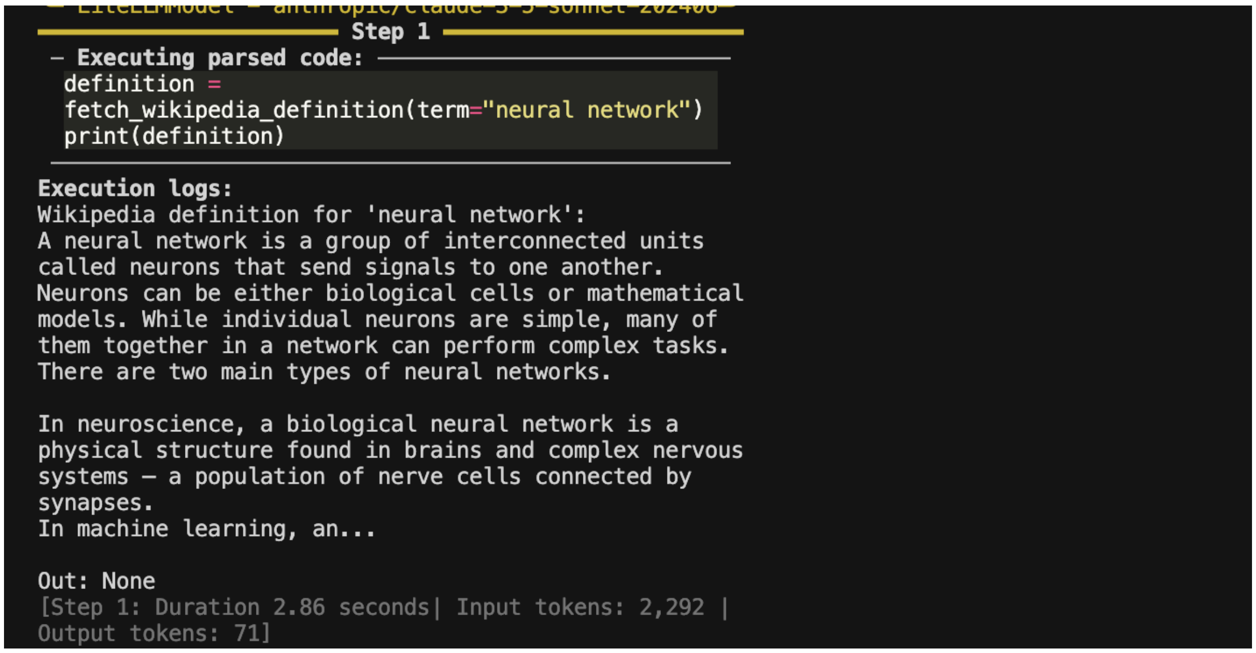
```
================================================================================
AI TERM EXPLAINER - QUICK TEST
================================================================================

🔍 Phoenix should be running at: http://127.0.0.1:6006
📊 Open that URL in your browser to see traces!

================================================================================

1️⃣ Initializing agent...
/opt/homebrew/lib/python3.11/site-packages/smolagents/models.py:1127: FutureWarning: The 'mod
el_id' parameter will be required in version 2.0.0. Please update your code to pass this para
meter to avoid future errors. For now, it defaults to 'anthropic/claude-3-5-sonnet-20240620'.
  warnings.warn(
✅ Agent ready!

⏳ Waiting for Phoenix to initialize...
✅ Ready!
```

Figure 0.1: AI Term Explainer startup. Phoenix is launched on `http://127.0.0.1:6006` for tracing, and the smolagents CodeAgent is initialized. This shows that the project is instrumented with OpenTelemetry/Phoenix and that the agent is ready to accept AI term queries.



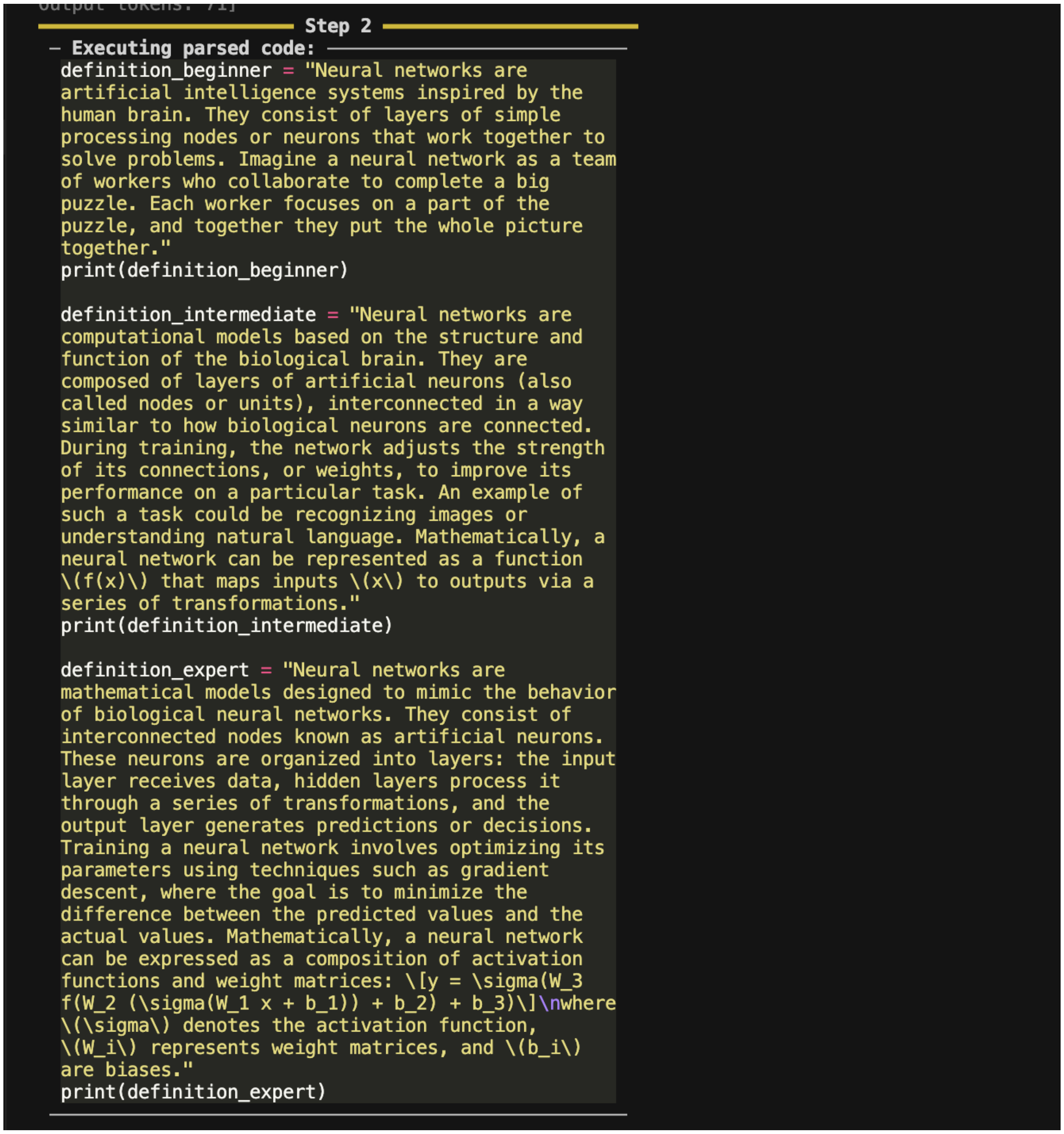
```
================================================================================
2 Testing with term: 'neural network'
================================================================================

💡 This will create a trace in Phoenix!
   Open http://127.0.0.1:6006 in your browser to see it.

--------------------------------------------------------------------------------

📝 Generating explanation for 'neural network'...
╭──────────────────────── New run ────────────────────────╮
│                                                          │
│ Explain the AI/ML term "neural network" at three         │
│ difficulty levels:                                       │
│                                                          │
│         1. BEGINNER level: Explain it as if the          │
│ reader has no prior knowledge of AI/ML.                  │
│            Use simple analogies and avoid                │
│ technical jargon.                                        │
│                                                          │
│         2. INTERMEDIATE level: Explain it for            │
│ someone with basic AI/ML knowledge.                      │
│            Include technical details and                 │
│ mathematical concepts where relevant.                    │
│                                                          │
│         3. EXPERT level: Provide a deep, technical       │
│ explanation suitable for researchers.                    │
│            Include mathematical formulations,            │
│ implementation details, and connections to related       │
│ concepts.                                                │
│                                                          │
│         First, use the fetch_wikipedia_definition        │
│ tool to get the base definition.                         │
│         Then, use your knowledge to expand and           │
│ adapt it to each difficulty level.                       │
│                                                          │
╰─ LiteLLMModel - anthropic/claude-3-5-sonnet-202406──────╯
```

Figure 0.2: Quick test with the term *"neural network"*. The wrapper script prints the natural-language task description given to the agent (three difficulty levels, beginner/intermediate/expert) and reminds that the agent should first call the FETCH_WIKIPEDIA_DEFINITION tool before generating explanations.



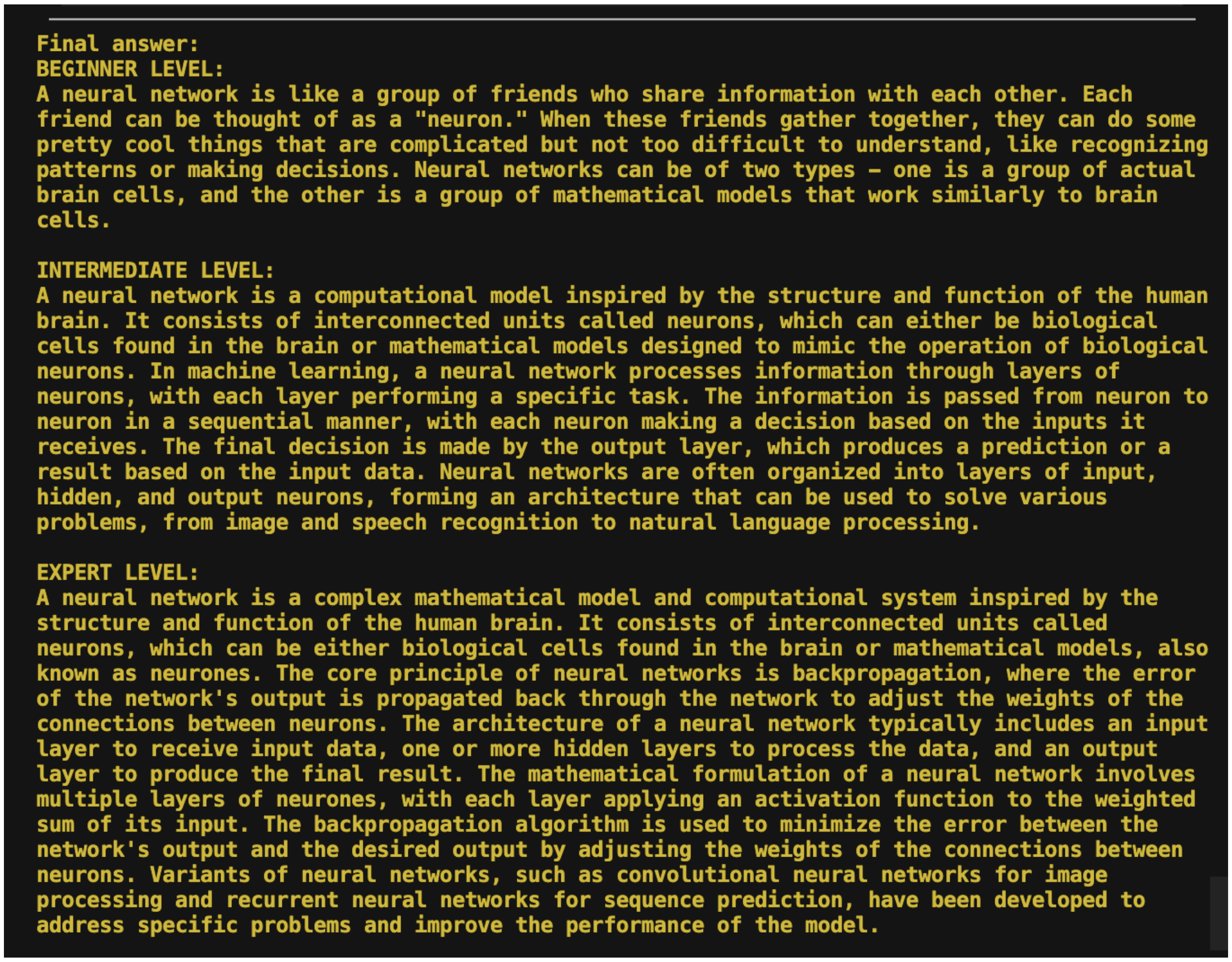
```
─ LiteLLMModel - anthropic/claude-3-5-sonnet-202406─
━━━━━━━━━━━━━━━━━━━━━━━━ Step 1 ━━━━━━━━━━━━━━━━━━━━━━━━
 ─ Executing parsed code: ──────────────────────────────
   definition =
   fetch_wikipedia_definition(term="neural network")
   print(definition)
 ────────────────────────────────────────────────────────
Execution logs:
Wikipedia definition for 'neural network':
A neural network is a group of interconnected units
called neurons that send signals to one another.
Neurons can be either biological cells or mathematical
models. While individual neurons are simple, many of
them together in a network can perform complex tasks.
There are two main types of neural networks.

In neuroscience, a biological neural network is a
physical structure found in brains and complex nervous
systems – a population of nerve cells connected by
synapses.
In machine learning, an...

Out: None
[Step 1: Duration 2.86 seconds| Input tokens: 2,292 |
Output tokens: 71]
```

Figure 0.3: Step 1 of the agent's reasoning trace in Phoenix. The agent calls the custom tool FETCH_WIKIPEDIA_DEFINITION(TERM="NEURAL NETWORK"). The execution log shows the truncated Wikipedia definition returned by the tool, confirming that the tool-augmented behavior is working correctly.



```
output tokens: 71]
━━━━━━━━━━━━━━━━━━━━━━ Step 2 ━━━━━━━━━━━━━━━━━━━━━━
 ─ Executing parsed code: ──────────────────────────
  definition_beginner = "Neural networks are
  artificial intelligence systems inspired by the
  human brain. They consist of layers of simple
  processing nodes or neurons that work together to
  solve problems. Imagine a neural network as a team
  of workers who collaborate to complete a big
  puzzle. Each worker focuses on a part of the
  puzzle, and together they put the whole picture
  together."
  print(definition_beginner)

  definition_intermediate = "Neural networks are
  computational models based on the structure and
  function of the biological brain. They are
  composed of layers of artificial neurons (also
  called nodes or units), interconnected in a way
  similar to how biological neurons are connected.
  During training, the network adjusts the strength
  of its connections, or weights, to improve its
  performance on a particular task. An example of
  such a task could be recognizing images or
  understanding natural language. Mathematically, a
  neural network can be represented as a function
  \(f(x)\) that maps inputs \(x\) to outputs via a
  series of transformations."
  print(definition_intermediate)

  definition_expert = "Neural networks are
  mathematical models designed to mimic the behavior
  of biological neural networks. They consist of
  interconnected nodes known as artificial neurons.
  These neurons are organized into layers: the input
  layer receives data, hidden layers process it
  through a series of transformations, and the
  output layer generates predictions or decisions.
  Training a neural network involves optimizing its
  parameters using techniques such as gradient
  descent, where the goal is to minimize the
  difference between the predicted values and the
  actual values. Mathematically, a neural network
  can be expressed as a composition of activation
  functions and weight matrices: \[y = \sigma(W_3
  f(W_2 (\sigma(W_1 x + b_1)) + b_2) + b_3)\]\nwhere
  \(\sigma\) denotes the activation function,
  \(W_i\) represents weight matrices, and \(b_i\)
  are biases."
  print(definition_expert)
 ───────────────────────────────────────────────────
```

Figure 0.4: Step 2 of the trace: the agent converts the base definition into three explanations at different difficulty levels. The code cell in the trace shows how it constructs separate `definition_beginner`, `definition_intermediate`, and `definition_expert` strings before returning the final answer.



```
Final answer:
BEGINNER LEVEL:
A neural network is like a group of friends who share information with each other. Each
friend can be thought of as a "neuron." When these friends gather together, they can do some
pretty cool things that are complicated but not too difficult to understand, like recognizing
patterns or making decisions. Neural networks can be of two types - one is a group of actual
brain cells, and the other is a group of mathematical models that work similarly to brain
cells.

INTERMEDIATE LEVEL:
A neural network is a computational model inspired by the structure and function of the human
brain. It consists of interconnected units called neurons, which can either be biological
cells found in the brain or mathematical models designed to mimic the operation of biological
neurons. In machine learning, a neural network processes information through layers of
neurons, with each layer performing a specific task. The information is passed from neuron to
neuron in a sequential manner, with each neuron making a decision based on the inputs it
receives. The final decision is made by the output layer, which produces a prediction or a
result based on the input data. Neural networks are often organized into layers of input,
hidden, and output neurons, forming an architecture that can be used to solve various
problems, from image and speech recognition to natural language processing.

EXPERT LEVEL:
A neural network is a complex mathematical model and computational system inspired by the
structure and function of the human brain. It consists of interconnected units called
neurons, which can be either biological cells found in the brain or mathematical models, also
known as neurones. The core principle of neural networks is backpropagation, where the error
of the network's output is propagated back through the network to adjust the weights of the
connections between neurons. The architecture of a neural network typically includes an input
layer to receive input data, one or more hidden layers to process the data, and an output
layer to produce the final result. The mathematical formulation of a neural network involves
multiple layers of neurones, with each layer applying an activation function to the weighted
sum of its input. The backpropagation algorithm is used to minimize the error between the
network's output and the desired output by adjusting the weights of the connections between
neurons. Variants of neural networks, such as convolutional neural networks for image
processing and recurrent neural networks for sequence prediction, have been developed to
address specific problems and improve the performance of the model.
```

Figure 0.5: Final console output for the term *"neural network"*. The agent returns clearly separated BEGINNER, INTERMEDIATE, and EXPERT explanations, demonstrating that the system meets the assignment goal of an educational AI term explainer with multi-level explanations.