AI Math Tutor Agent

Course: ITAI 2376 - Deep Learning

Student: SM Tawhid

Project Overview

The AI Math Tutor Agent is a rule-based, interactive assistant built to help users practice solving simple algebra equations. It is designed to operate in a Jupyter Notebook environment, using symbolic computation and string processing to respond to basic algebraic problems. The primary goal of this agent is to simulate a math tutoring experience by allowing users to either solve equations directly or take randomly generated quiz questions. The agent processes user input, checks correctness, and provides appropriate feedback. This project focuses on combining foundational programming logic with symbolic math tools to build an easy-to-understand, functional NLP agent without relying on large language models.

System Architecture

The system consists of three core modules: the main loop controller, equation solver, and quiz generator. The loop controller manages the interaction with the user, reading the input and determining the appropriate response type. If the user enters a command like "x + 2 = 5," the system parses it and routes it to the solver module. If the command is "quiz," it generates a random linear equation, waits for user input, and validates the answer.

The equation solver uses the SymPy library to interpret and solve algebraic expressions involving one variable. The quiz generator uses Python's `random` module to generate equations in the form `a*x + b = 0`, where `a` and `b` are integers. The agent checks if the user's quiz answer matches the correct float result (within a small

tolerance), allowing for rounding and formatting differences. The system prints responses back to the user, creating a complete interactive loop.

Implementation Details

The entire agent is implemented within a single Jupyter notebook file using standard Python 3.10 libraries. The use of SymPy allows us to easily parse and solve equations symbolically without writing complex math logic from scratch. We use regular expressions to extract user commands, and simple control structures (`if`/`else`) to determine the response path.

For quiz generation, the function `generate_quiz()` creates linear equations by selecting random integers `a` and `b`, then returns both the equation string and the correct answer. The user response is captured using `input()` and compared to the solution using a tolerance-based method with `abs(user_input - answer) < 0.01`.

The `agent_response()` function handles the main logic and routes input to either the equation solver or the quiz module. If the command is unrecognized, the agent responds with a default message prompting the user to try supported commands like "=" or "quiz." There is also a termination command (`quit`) that gracefully ends the session.

No external APIs, databases, or web tools are integrated. This was a deliberate decision to keep the implementation lightweight, fast, and easy to run in a controlled

notebook environment. All dependencies are either part of the Python standard library or available via `pip` in common Python environments.

Evaluation Results

The agent was tested using a variety of valid and invalid inputs. For equation solving, user input such as "x + 3 = 7" consistently returned the correct result (x = 4). We also tested cases with negative coefficients and floating-point solutions to ensure the solver handled these without issues.

The quiz module was tested by entering correct and incorrect answers for auto-generated problems. When the input was numerically correct but formatted differently (e.g., "x = -4/5" instead of just "-0.8"), the system marked it incorrect due to strict type comparison. To address this, we adjusted the input validation to allow float-type inputs and compare them using tolerance. This resolved the issue and ensured the agent accepted all valid numerical inputs regardless of formatting.

Overall, the system performed reliably under various test scenarios. The interface is intuitive, responses are quick, and the logic is robust for the limited task domain.

Challenges and Solutions

The biggest challenge was managing user input formats and handling unexpected input during quizzes. Initially, the agent would reject correct answers like "-1/3" or "x = -8/9" because it expected a float input. To solve this, we adjusted the quiz input logic to accept only numeric answers and convert them to floats before checking.

Another challenge was maintaining clean control flow in the `agent_response()` function. We simplified the logic by limiting command recognition to "=," "quiz," and "quit," and avoided natural language understanding. This kept the scope manageable and reduced bugs caused by ambiguous input.

We also avoided implementing unnecessary features like explanation or web integration, since the focus was on functionality and stability within a notebook environment.

Lessons Learned

This project provided hands-on experience with building a task-specific NLP agent from scratch. We learned how to structure interactive agents using simple rule-based logic and how to integrate symbolic computation for solving algebra problems. The use of SymPy made it easier to focus on the agent logic rather than writing complex solvers.

Another important takeaway was the value of handling edge cases and user input errors gracefully. Providing helpful feedback when input is invalid enhances the user experience and reduces frustration.

Finally, this project reinforced the importance of simplicity. By keeping the features limited and the architecture lightweight, we were able to finish implementation and testing quickly without introducing unnecessary complexity.

Future Improvements

If given more time, the agent could be improved in several ways. First, support for fractional and symbolic answers in the quiz module would make the agent more flexible. This could be achieved by parsing string input with SymPy rather than converting directly to float.

Second, a GUI or web interface would enhance accessibility and make the tool more usable outside the notebook environment. Integration with a voice assistant could make the experience more natural.

Lastly, we could expand the agent's domain beyond linear equations to include quadratic equations, inequalities, or systems of equations. This would involve additional logic and validation but would make the agent more powerful as a tutoring tool.

For this assignment, the agent meets all the basic requirements and demonstrates the core concepts of interactive agent design, symbolic computation, and user input handling in Python.