Berehulia Mykyta – Generative AI

NLP – Assignment 2

In the related repository you will find notebooks with iterations on how the solution was developed starting from barebone version up to multi agentic one.

Each notebook has a description in the very beginning but I've pulled them up here for convenience:

Iteration 1

The first iteration is just a barebones skeleton of the future solution. It contains a model (gpt-4o for sanity check) and a basic system prompt that asks the model to generate a task in Ukrainian without providing any details or criteria, and compares the model's output with a user-created task.

The dataset has just three fields (task, solution, and answer) and only one task type: open questions that require an answer.

Key characteristics:

- Used GPT-4o model for initial sanity check
- Simple system prompt asking the model to generate math tasks in Ukrainian
- Single task type: open-ended questions requiring calculated answers
- Basic dataset with three fields: task, solution, answer
- Direct comparison of model output against human-created sample tasks

Iteration 2

In the second iteration, it was decided to add one more task type - quiz (the task field in the dataset was renamed to the description, and an additional type field was added), and add some additional rules that the model should follow.

Right now, the model receives a request to generate a task or a quiz for a specific topic.

If it is a task:

- Description: a problem description with a single question asked.

- Solution: an ordered list with a step-by-step explanation of how to get the correct answer.
- Answer: Just a number, or a final equation.

If it is a quiz:

- Description: a problem and a set of options to choose from (always four options enumerated as Latin letters A, B, C, D).
- Solution: an ordered list with a step-by-step explanation of how to get the correct answer.
- Answer: Just a Latin letter corresponding to the answer.

Also, the response needs to be formatted as a valid JSON right now.

The model had a few issues, all related to JSON formatting (adding an ordered list not as a string but as a JSON list, wrapping JSON in a markdown block, and so on). All solved with system prompt modification.

Key improvements:

- Added quiz type support with multiple choice options (A, B, C, D)
- Restructured dataset schema: renamed task to description, added type field
- Introduced JSON output formatting requirement for consistent parsing
- Enhanced system prompt with detailed formatting rules for each content type

Iteration 3

In the third iteration, it was decided to strengthen the solution by adding the following features:

- Before sending the actual question, I added a few sample messages with responses to indicate that it is a few-shot prompt (to eliminate potential formatting issues).
- Added a framework for evaluation. First, it was decided to add a metric for failed JSON responses. Second, it was decided to add an LLM as a judge framework, which has its own criteria for checking whether the task meets the desired acceptance criteria (if the task contains a numbered list in the solution field, if the answer is formatted correctly, if the

proper task type is returned, and so on). It was decided to implement it as an additional agent.

Key improvements:

- Implemented few-shot prompting to eliminate formatting inconsistencies
- Created two-agent architecture:
    - MathTutor: Generates math content based on topic and type
    - AcceptanceJudge: Evaluates generated content against quality criteria

Iteration 4

In the fourth iteration, it was decided to allow the model to work with arbitrary input and to switch from Generator-only mode to a Solver as well.

The idea was to update only the MathTutor class. It should decide right now whether it sees a question or a task request, and how to act. The resulting output is the same: either a new task or the user request reformulated as a task, so we do not need to update the downstream Judge.

Key improvements

- MathTutor now handles arbitrary user input
- Automatic mode detection: topic requests trigger generation, problems trigger solving
- Unified output format ensures downstream Judge compatibility
- Three distinct few-shot examples covering all use cases

Iteration 5

In the fifth iteration, it was decided to refactor the code to use idiomatic LangChain patterns while keeping it simple and linear

Key improvements:

- Added Pydantic models for typed outputs (MathContent, JudgmentResult)

- Used FewShotChatMessagePromptTemplate for few-shot examples

- Used ChatPromptTemplate.from_messages() instead of manual message lists

- Used PydanticOutputParser for automatic JSON parsing

- Created chains with LCEL: prompt | model | parser

Iteration 6

In the sixth iteration, two three improvements were made:

1. The solution field now includes a theoretical explanation (formulas, rules, definitions) before the numbered steps of the solution.

2. Added RAG using math textbooks (grades 5-11) to extract relevant terminology and theory for better context.

3. Model was changed from 4o to more powerful 5.2 (#1 on LMArena)

Key Improvements:

- Solutions now include theoretical explanations (formulas, rules, definitions) before numbered steps
- Implemented RAG (Retrieval-Augmented Generation) using math textbooks
- Created FAISS vector store from Ukrainian math textbooks (grades 5-11)
- Upgraded model from GPT-4o to GPT-5.2 for improved reasoning

Iteration 7

In iteration seven, integration with Wolfram Alpha was added. Right now, the tutor model has a tool to verify the calculations, and here is an additional agent that performs verification of calculations via it's API.

The LangChain tool was not working properly due to some bug with async operations, so the custom one was added. Also, the dataset was extended with more complex questions for undergraduete topics (linear agrebra, analysis and so on).

The model was capable to solve the problems. While some tuning of system prompts was required.

Final Solution

- Wolfram Alpha integration for answer verification
- Three specialized agents working together:
    - MathTutor: Content generation with Wolfram Alpha tool access
    - AcceptanceJudge: Quality and format assessment

    - VerificationAgent: Mathematical correctness verification
- Extended dataset with undergraduate-level topics (linear algebra, calculus)
- LangGraph ReAct agent pattern for tool-using agents

As result on the testing dataset here 59 successes and 1 failure to generate a correct response.

Technology Stack:

- LangChain
- OpenAI GPT-4o (iterations 1-5) → GPT-5.2 (iterations 6-7)
- FAISS with OpenAI embeddings
- Wolfram Alpha (mathematical verification)