Let's Code Step By Step: Intelligent LLM Tutors for Competitive Programming

Priscilla Zhu (QZ2531), Diya Nair, Andrea Lopez, Begum Gokmen

Motivation - Competitive programming presents significant barriers to entry, especially for students from underrepresented backgrounds who lack the resources to prepare. While plenty of textbooks cover data structures and algorithms, students often rely on solutions from platforms like USACO or Codeforces when they feel stuck on problem-solving. However, seeing complete solutions at once discourages independent thinking - sometimes, a hint is all that's needed. In these cases, coaches play a crucial role by reviewing students' code and asking questions that inspire them to keep thinking without providing direct answers. We want to **harness the reasoning ability of language models (LMs) to act as this "coach," prompting students with helpful questions at each step when they're stuck.** This would provide AI-powered, personalized assistance for students without a coach, an expensive resource that's hard to access. Our project aims to use chain-of-thought prompting to obtain LLM-generated questions to guide students' algorithmic problem-solving. We believe our pipeline could be adapted to other fields like technical interview preparation or STEM problem-solving, benefiting a broader population in the future. It could also potentially address concerns about LLMs in education by promoting guided problem-solving rather than giving away complete answers.

Related work - There is abundant research on CoT prompting and LLM reasoning. We also found studies that evaluate LLM's code generation ability on competitive programming problems. Although code generation isn't our focus, the datasets provided by these studies are useful for our purpose, such as TACO. We also looked at research on using LLM as teaching assistants, although we didn't find many large-scale evaluations of the efficiency or success of LLM tutoring (we would really appreciate some direction for this!). We listed some papers relevant to our work here.

Action Plan: Methods

1. Feed LLM (potentially DeepSeek) a competitive programming problem + solution
2. Prompt LM to break the solution down into intermediate steps
    a. Example result: [{Step 1: We use a stack to store the nodes expanded by DFS. Step 2: … }]
3. For each step, prompt LM to generate a question to ask the student
    a. Ex: [{Question: The DFS algorithm expands on the last node added to the frontier. What data structure could you use to accomplish this behavior? Expected answer: I can use a stack.} …]

The resulting dataset after the pipeline might look like this:

Coding Problem	Ground Truth Solution	LLM CoT Steps breakdown	LLM Questions	Expected Answers to LLM Questions

From there, we could potentially finetune an LLM for code generation following these generated steps and questions. We can also work on student-LM interaction, such as evaluating the student's answer:

1. If correct, output "correct!" and encourage the student to keep attempting the problem
2. Keep prompting the student with questions until the student gets it correct.

Measures of Success - Qualitatively, we want to attempt a competitive programming problem and see how well our LM prompts us in intermediate steps. We may also get programmers of different levels to attempt problem-solving with the assistance of this LM agent. **Quantitatively**, we can feed the CoT or intermediate step questions generated by our LM to other LLM coders, such as AlphaCode. We can use existing benchmarks on code generation such as Code-Vision (which also evaluates LLMs logic understanding) to see whether AlphaCode generates better quality, more correct, or more reasonable code after being fine-tuned on our CoT dataset. The Code-Vision benchmark contains HumanEval-V, Algorithm, and MATH, which evaluates LLM's basic programming, algorithmic problem-solving, and mathematical problem-solving, respectively. We are primarily interested in the first two metrics and of course the correctness of the solution.

Unknowns and Obstacles - We are still not sure how the final product would prompt the intermediate steps. Should it be integrated into a live coder, and read from user input periodically? For the scope of this project, we'll be satisfied if the LM can perform good question prompting, even with horrible UI. We would really appreciate feedback on better datasets and benchmarks we could use! We are also not entirely sure whether the methods under the Action Plan are the most optimal, or whether it would achieve our goal. We would love any suggestions on how we could improve the streamline of the project methods and have very clear steps! Thank you.

References and Related Work

CoT Prompting & Instruction Fine Tune

- Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

LLM in Competitive Programming / Code Generation

- Evaluating the Performance of Large Language Models in Competitive Programming: A Multi-Year, Multi-Grade Analysis
- Competition-Level Code Generation with AlphaCode
- Benchmarking Language Model Creativity: A Case Study on Code Generation

LLM As Teaching Assistant / Personalized Learning Agents

- AI-TA: Towards an Intelligent Question-Answer Teaching Assistant using Open-Source LLMs
- Beyond Traditional Teaching: Large Language Models as Simulated Teaching Assistants in Computer Science
- TAMIGO: Empowering Teaching Assistants using LLM-assisted viva and code assessment in an Advanced Computing Class
- Navigating the Landscape of Hint Generation Research: From the Past to the Future
- CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs

Instruction Tuning & Prompt Templates

- PromptSource
- Public Pool of Prompts

Potential Datasets

Originally, we planned to web-crawl challenge problems from USACO's training page and historical contest problems. These are the problems our group member Priscilla trained on and is the most familiar with. However, we found a couple of open-source datasets that may have robust potential.

TACO: Topics in Algorithmic COde generation dataset

- The TACO dataset contains 25433 and 1000 coding problems in training and test sets, respectively, as well as up to 1.55 million diverse solution answers. Each problem in the training set has 51.6 test cases, and the test set has 202.3.
- The dataset contains challenge problems, solutions code in Python, input-output examples, test cases(!), and time and memory limits. The last two entries are especially important because they determine which algorithms would be acceptable to use (for example, O(n^2) algorithms won't satisfy the time limit of 1 second if n is greater than ~10^9).
- From the paper, it seems like this dataset features diverse solutions after de-duplication: each problem features 58.21 correct code solutions. One of the concerns we had was that there usually exist multiple solutions to a problem, using various data structures or algorithms. We were worried if the LM coach would be able to identify these multiple ways, if only given one correct, "standard" solution, as the USACO website usually only provides one solution for reference. Having multiple solutions from the TACO dataset might mean the LM can learn to pick up diverse ways of problem-solving.
- We can access the TACO dataset on: Hugging Face, GitHub