Code Generation with AlphaCodium: From Prompt Engineering to Flow Engineering

(paper presentation report)

Bijayan Ray (MCS202403)

Email: bijayan@cmi.ac.in

May 25, 2025

Code generation is somewhat different from the natural language generation since it requires precise syntax matching, handling edge cases and requires attention to fine grained problem details. It however has an advantage due to the presence of a compiler enabling testing it easier. This paper introduces an approach named *AlphaCodium* for code generation using large language models. It employs a test-based, multi-stage, code-oriented iterative flow to enhance the performance.

AlphaCodium was tested on *CodeContests* dataset. AlphaCodium significantly improved the accuracy of LLMs. For example GPT-4's "pass5" accuracy. increased from 19% to 44% using this approach.

The CodeContests dataset was introduced by Google DeepMind containing ∼10,000 problems from competitive programming platforms like *Codeforces* designed for training and evaluating LLMs. It comprises training, validation and test sets each including description and public tests as model inputs. The evaluation is based on private test sets (∼200 input-output tests per problem). The paper develops a code-oriented flow for existing LLMs (like *GPT, DeepSeek*) focusing on just validation and test sets. This dataset has large private test sets and the problems are long, complex and detailed simulating real-world scenarios. Problem understanding is essential for a successful code generation and that is achieved through self-reflection and introspection by the model.

The common prompting techniques like single prompts or chain-of-thought prompting are ineffective for complex code generation tasks like those in CodeContests. The traditional NLP style flows are suboptimal. AlphaCodium utilizes self-reflection, test reasoning to increase understanding of the problem. It also enriches public tests with AI-generated test cases increasing coverage of edge cases. The AlphaCodium flow is broadly divided into pre-processing and code iterations phases. The pre-processing phase includes generating problem reflection (breaking down the problem statements to understand) on the input problem description → public tests reasoning → generate possible solutions → rank solutions → generate additional AI tests. The code iterations include iterating the initial code solution on public tests → iterate on AI tests → output solution to the problem.

There are several code-oriented design concepts and tricks that added to the performance of AlphaCodium. YAML structured output was used, since it enables complex, multi-stage outputs in a clear, code-like format reducing prompt engineering complexity. The flow asked the model to output reasoning in bullet points that encourages modular understanding and logical segmentation of the problem thus improving clarity and performance in reasoning tasks. It focused on modular code generation that improves code quality and makes iterative fixing easier and effective. The model was asked to regenerate and self-correct outputs (double validation) which was effective in strict decision-making tasks. The flow encouraged exploration and avoided making early, irreversible decisions in complex tasks rather it used a gradual data accumulation: starting with problem reflection and public test reasoning → generating additional tests and solution strategies → generate and iterate on code. Test anchors were used to deal with uncertainty in AI-generated test correctness. This includes iterating on trusted public tests and set passing ones as anchors then as AI tests pass, add them to anchor set, but when a test fails, only accept the code changes that still pass all the anchor tests. This prevents regression and misdirection due to faulty test cases.

AlphaCodium significantly outperformed direct prompting, like on GPT-4 which showed a 2.3× improvement as per the pass5 metric (pass$k$ metric the percentage of problems solved using k generated solutions per problem). Some of the methods include *CodeChain* and *AlphaCode*. AlphaCodium outperformed CodeChain(using GPT-3.5) consistently with better pass5 results. AlphaCodium performed better than AlphaCode which used a fine-tuned model, despite using fewer ($\sim 10^{-4}\times$) LLM calls. Thus AlphaCodium achieves strong performance improvements in code generation that is more effective than direct prompting, more sample and compute efficient using just the standard large language models.

However, some limitations to AlphaCodium include: injecting last 50 lines of execution trace into prompt didn't improve results, providing last k failed code attempts didn't help steering the model effectively, supplying previous code differences as context didn't show improvement and optimizing long or chain-of-thought prompts didn't improve performance.

Some newer approaches enhancing code generation includes: RepoRift that uses Retrieval Augmented Generation (RAG) powered agents to inject information into user prompts, RLEF that uses reinforcement learning from execution feedback and AlphaVerus that uses formal verification with self improving translation and tree refinement. One can find more about the new improvements in this recent survey on usage of LLMs for code generation.