Code Generation with AlphaCodium: From Prompt Engineering to Flow Engineering

(paper presentation)

Bijayan Ray
Akhoury Shauryam

May 17, 2025

Contents

Overview

- The paper addresses the challenge of enhancing code generation by LLMs, which struggle with syntax accuracy and handling problem-specific details.
- Code generation differs from typical natural language tasks due to the need for exact syntax, handling edge cases, and following detailed specifications.
- Techniques successful in natural language generation may not work well for code generation.

Overview (Cont)

- The authors propose AlphaCodium, a test-based, multistage, code-focused iterative method for improving code generation by large language models.
- AlphaCodium was evaluated on the CodeContests dataset, which includes competitive programming problems from platforms like Codeforces.
- The approach significantly improves performance; for instance, GPT-4's pass@5 accuracy rose from 19% (with a single prompt) to 44% using AlphaCodium.
- The insights and practices from AlphaCodium are considered broadly useful for general code generation tasks.

Dataset

- CodeContests is a challenging dataset introduced by DeepMind, sourced from competitive programming platforms like Codeforces.
- It includes 10K code problems for training, and separate validation (107 problems) and test sets (165 problems) for evaluation.
- This work focuses on applying a code-oriented flow to existing LLMs (e.g., GPT, DeepSeek) rather than training a new model, using only the validation and test sets.
- Each problem provides a description and public tests; the model must generate code that passes a hidden private test set.

Dataset

- Key strengths of CodeContests:
  - It includes ≈200 private tests per problem to ensure robustness and prevent false positives.
  - The problem descriptions are intentionally long and nuanced, requiring attention to small but critical details.
  - This setup better reflects real-world coding scenarios, which often involve complex and detail-rich tasks.
  - In contrast, simpler datasets like HumanEval contain shorter, more straightforward problems.
- Effective problem understanding, supported by techniques like self-reflection, improves clarity and increases the likelihood of generating correct solutions.

Dataset

images/problem description reflection.png

AlphaCodium proposed flow

- Common prompt engineering techniques (e.g., single prompts, chain-of-thought) do not yield significant improvements for code generation tasks like CodeContests.
- LLMs often fail to fully comprehend the problem, producing incorrect or overfitted code that passes public tests but fails on unseen ones.
- Natural language generation flows are suboptimal for code generation tasks.
- Code generation tasks offer a unique advantage: the ability to run and test code iteratively.
- AlphaCodium introduces a dedicated, iterative flow optimized for code generation and testing.

AlphaCodium proposed flow

- The approach consists of two major phases:
  - Pre-processing phase:
    - Reflect on the problem in natural language.
    - Perform public tests reasoning.
    - Generate and rank 2-3 natural language solution strategies.
    - Enrich public tests by generating 6-8 additional diverse AI-generated tests.
  - Code iterations phase:
    - Generate an initial code solution based on the selected strategy.
    - Run the code on both public and AI tests, iterating and fixing errors.
    - Iterate further to fix code based on test failures and error messages.

AlphaCodium proposed flow

- Detailed stages of the flow:
  - Problem reflection: summarize the problem's goal, inputs, outputs, constraints, and rules in bullet points.
  - Public tests reasoning: explain why each input yields the corresponding output.
  - Generate possible solutions: write 2-3 natural language strategies.
  - Rank solutions: select the best based on correctness, simplicity, and robustness.
  - Generate AI tests: create additional tests covering edge cases and large inputs.

AlphaCodium proposed flow

Detailed stages of the flow (continued):

- Initial code solution:
  - Choose a solution, generate corresponding code.
  - Run code on selected tests, repeat until successful or try-limit reached.
  - Use the best-passing or closest-output code as a base.
- Iterate on public tests: run and fix code iteratively using feedback from public tests.
- Iterate on AI-generated tests: repeat the run-fix process using AI tests and test anchors.

AlphaCodium proposed flow

Additional insights:

- The flow supports knowledge accumulation, progressing from easy to hard tasks.
- Pre-processing outputs help the more difficult code generation stages.
- Generating test cases is easier for LLMs than writing complete solutions.
- Additional AI tests improve generalization by targeting underrepresented scenarios.
- Some stages can be combined in a single LLM call using structured prompts.

AlphaCodium proposed flow (continued)

images/alphacodium.png

Code Oriented design concepts

- *YAML structured output:* The use of YAML format, equivalent to a Pydantic class, provides a structured, code-like way to present complex tasks.
  - Simplifies prompt engineering by reducing ambiguity.
  - Facilitates multi-stage, logical thinking processes.
  - Preferred over JSON for code generation tasks due to better readability and structure.
- *Semantic reasoning via bullet points analysis:* Encouraging models to reason using bullet points improves understanding and output quality.
  - Bullet points help divide reasoning into semantic sections (e.g., description, rules, input, output).
  - Leads to clearer and more structured problem analysis.

Code Oriented design concepts

- *Modular code generation:* LLMs perform better when asked to generate code in modular sub-functions.
  - Reduces logical errors and bugs.
  - Enhances the effectiveness of iterative fixing by localizing errors.
- *Soft decisions with double validation:* To address hallucinations and errors in complex tasks, AlphaCodium uses a double validation step.
  - The model is asked to re-generate and correct its own output instead of being queried with binary (yes/no) correctness questions.
  - Encourages deeper reasoning and self-correction.

Code Oriented design concepts

- *Postpone decisions and leave room for exploration:* Avoid asking the model direct questions about complex problems too early.
  - Adopt a gradual process:
    - Start with self-reflection and reasoning about public tests.
    - Proceed to generate AI tests and explore possible solutions.
    - Only then generate the code and perform run-fix iterations.
  - Instead of selecting a single solution, rank multiple and explore iterations from top-ranked options.
  - This reduces the risk of hallucinations and premature commitments.

Code Oriented design concepts

- *Test anchors:* Designed to address the uncertainty of whether a failed test is due to incorrect code or an incorrect test.
    - Begin with public tests (known correct) to form initial anchor tests.
    - Iterate through AI-generated tests, adding passing ones to the anchor list.
    - For failing tests, assume the code is incorrect, but ensure that the fix still passes all anchor tests.
    - This process protects against overfitting to faulty AI-generated tests.
    - An additional optimization involves sorting AI-generated tests from easy to hard to build the anchor base early.

Code Oriented design concepts

images/prompt_structured_output.png

Experiments

- Single direct prompt using the pass@k metric:
  - AlphaCodium significantly outperforms the direct prompt approach across both validation and test sets.
  - For example, GPT-4's pass@5 score on the validation set improves from 19% to 44%, a 2.3x improvement.
  - The improvement is consistent for both open-source (DeepSeek) and closed-source (GPT) models.
- Comparison with prior works:
  - AlphaCodium outperforms CodeChain when using the same model (GPT-3.5) and metric (pass@5).
  - AlphaCode employs a brute-force-like approach: fine-tuning an unknown model, generating up to 100K code solutions, clustering them, and submitting the top K clusters.

Experiments

- Comparison with prior works (continued):
  - Despite AlphaCode's large-scale generation strategy, AlphaCodium achieves better top results using significantly fewer resources.
  - Neither AlphaCode nor CodeChain released reproducible open-source code or evaluation scripts, while AlphaCodium provides a full reproducible solution to support consistent future comparisons.
  - Evaluation subtleties, such as handling multiple correct solutions or timeouts, are addressed in AlphaCodium's released framework.

Experiments

- Computational efficiency:
  - AlphaCodium requires around 15–20 LLM calls per solution; thus, a pass@5 submission uses approximately 100 LLM calls.
  - AlphaCode's pass@10@100K setup involves generating 100K solutions and selecting 10, leading to an estimated 1 million LLM calls.
  - AlphaCodium achieves superior performance with four orders of magnitude fewer LLM calls.
  - AlphaCode2, using a fine-tuned Gemini-Pro model, claims over 10,000$\times$ greater sample efficiency than AlphaCode.
  - Both AlphaCode2 and AlphaCodium achieve similar efficiency improvements over AlphaCode, but AlphaCodium relies solely on general-purpose models without extra training or fine-tuning.

Experiments

images/experiments.png

images/experiments_another.png

Conclusion

- The paper presents AlphaCodium, a code-oriented iterative flow that improves code generation by running and fixing generated code against input-output tests.
- The flow is divided into two main phases:
  - Pre-processing phase: natural language reasoning about the problem.
  - Code iterations phase: iteratively refining code using public and AI-generated tests.
- AlphaCodium incorporates several effective design practices:
  - Structured output in YAML format.
  - Modular code generation.
  - Semantic reasoning using bullet point analysis.
  - Soft decisions validated by double checks.
  - Encouragement of solution exploration.
  - Use of test anchors to guide iterations.

Conclusion

- The approach was evaluated on the CodeContests dataset, a challenging benchmark for code generation.
- AlphaCodium consistently improves performance across both closed-source and open-source models.
- It outperforms prior works while using a significantly smaller computational budget.

Reference

- Ridnik, Tal, Dedy Kredo, and Itamar Friedman. "Code generation with AlphaCodium: From prompt engineering to flow engineering." *arXiv preprint arXiv:2401.08500* (2024).

Thank you!

Questions?