Andrii Zhabrovets
Salmsacherstrasse 11A
8590 Romanshorn
076 525 13 74
anzhabro@ksr.ch

Kantonsschule Romanshorn
Class 4Mez
Matura

# Artificial Intelligence as a Substitute for Software Engineers: A Benchmark-Based Analysis

Subject: Computer Science
Supervisor: Tom Hofmann
Deadline: 6 January 2024

# Abstract

**To Be Done**

# Contents

**1 Introduction** **1**

**2 Research Methodology** **3**

2.1 Description of the Benchmark Framework . . . . . 3

2.1.1 Selection and Categorization of Programming Problems . . . . . 3

2.1.2 Characteristics of the Problem Set . . . . . 4

2.2 Overview of AI Models Evaluated . . . . . 4

2.3 Evaluation Metrics and Criteria . . . . . 5

**3 Results and Analysis** **7**

3.1 Quantitative Results . . . . . 7

3.2 Qualitative Assessment . . . . . 7

3.3 Comparison to Human Performance . . . . . 7

3.4 Statistical Analysis . . . . . 7

**4 Implications and Future Directions** **8**

4.1 Implications of Findings for Software Engineering . . . . . 8

4.1.1 Selection and Categorization of Programming Problems . . . . . 8

4.1.2 Characteristics of the Problem Set . . . . . 8

4.2 Challenges and Limitations . . . . . 9

4.3 Future Potential . . . . . 10

**5 Conclusion** **12**

**List of Figures** **15**

# 1 Introduction

The rapid advancements in Artificial Intelligence (AI) have brought transformative changes to numerous industries, with software engineering being a significant focus. AI-driven systems have moved beyond automation of routine tasks and now exhibit the potential to tackle complex challenges, including programming problem-solving. These developments have prompted critical discussions about whether AI can assume roles traditionally held by software engineers. Understanding the capabilities and limitations of these systems is essential for gauging their potential impact on the software development process.

This study explores the problem-solving capabilities of state-of-the-art AI models by evaluating their performance on programming tasks. Using a rigorous benchmarking framework, we assess these models through a set of common problems sourced from Leetcode, a platform widely acknowledged for its relevance in evaluating programming proficiency. By providing consistent prompts and analyzing their outputs, the research seeks to measure the effectiveness of AI in solving tasks of varying complexity and to draw meaningful comparisons with human performance.

The investigation centers on analyzing the technical skills of AI in problem-solving, identifying strengths and weaknesses, and determining its readiness to operate independently or alongside human developers. In doing so, this study also examines broader questions about the evolving role of AI in software engineering and its implications for the future of the profession.

The paper is structured to provide a comprehensive analysis of the topic, beginning with a review of existing literature to situate the research within the broader context of AI applications in software development. A detailed methodology follows, outlining the design and implementation of the benchmarks used for evaluating AI performance. The results are presented and analyzed to highlight key patterns and insights, culminating in a discussion of their implications for software engineering and AI research. Finally, the study concludes by reflecting on the findings and their relevance to the question of whether AI is capable of substituting software engineers.

By addressing these themes, this research aims to contribute to a deeper understanding of the

capabilities of AI in software development and to inform future directions for improving AI systems in this domain.

# 2 Research Methodology

## 2.1 Description of the Benchmark Framework

Why Leetcode was chosen as a representative platform for programming tasks.

Explanation of the benchmark framework's design to ensure fairness and consistency.

Steps to automate problem feeding and solution evaluation for AI models.

### 2.1.1 Selection and Categorization of Programming Problems

Criteria for selecting programming problems (e.g., diversity in difficulty levels: easy, medium, hard).

Examples of specific problem categories and why they were chosen.

The selection and categorization of programming problems is a foundational step in designing an effective benchmarking framework. For this study, the problems were chosen to represent a diverse and balanced set of challenges, ensuring that the evaluation captures the full spectrum of skills required in software development.

The selection process prioritized problems that reflect common themes in programming, such as algorithms, data structures, and computational logic. These categories were chosen because they are fundamental to both academic and practical software engineering tasks. Problems that demand optimization, reasoning, and debugging skills were included to test the AI's ability to handle both routine and advanced challenges. Care was also taken to include problems of varying complexity, categorized as easy, medium, and hard, to evaluate the models' performance across different levels of difficulty.

Easy problems typically involve straightforward implementations, such as basic arithmetic operations or simple control structures, designed to test whether the AI can understand and respond to basic instructions accurately. Medium-level problems often introduce more complex requirements, such as recursion, intermediate data structures (e.g., linked lists or trees), or algorithmic optimizations. Hard problems, on the other hand, often require deep problem-solving skills,

including advanced algorithms (e.g., dynamic programming or graph theory), intricate logic, and creative approaches to meet strict efficiency constraints.

To ensure a representative dataset, problems were selected from Leetcode, a widely recognized platform for programming challenges. This source was chosen for its well-structured problem descriptions, comprehensive test cases, and relevance to real-world coding scenarios. Problems with clear success criteria were prioritized to simplify the evaluation process and ensure that AI outputs could be objectively assessed.

Categorizing the problems by type and difficulty also helps identify the specific strengths and weaknesses of the AI models. For example, analyzing the performance across problem types can reveal whether a model excels in certain areas, such as mathematical computations, but struggles with others, such as handling edge cases in dynamic programming problems. This structured approach to selection and categorization ensures a comprehensive and meaningful assessment of AI capabilities in solving programming tasks.

### 2.1.2 Characteristics of the Problem Set

Statistical overview of the problem set (e.g., number of problems, distribution by difficulty).

Importance of including real-world-relevant challenges in the dataset.

Avoiding bias by ensuring the problem set covers a wide range of skills (e.g., coding logic, optimization, error handling).

## 2.2 Overview of AI Models Evaluated

Brief introduction to the AI models tested (e.g., GPT models, Gemini, or other programming-specific models, Claude).

Key capabilities of each model (e.g., natural language understanding, multi-step reasoning).

Why these models were chosen and their relevance to programming tasks.

## 2.3 Evaluation Metrics and Criteria

Metrics used to measure AI performance, such as:

Correctness (did the solution work?).

Efficiency (e.g., runtime complexity or memory usage).

Clarity (how well-structured the solution is).

Evaluating the performance of AI models in solving programming problems requires a clear and rigorous set of metrics to ensure consistency and objectivity. The chosen metrics in this study are designed to assess the correctness, efficiency, and overall quality of solutions produced by AI models, reflecting their applicability to real-world programming tasks.

The primary criterion for evaluation is **correctness**, which measures whether the solution provided by the AI satisfies the problem's requirements. This involves running the AI-generated code against a suite of test cases, including edge cases, and determining if all outputs are as expected. Correctness serves as the most fundamental benchmark, as a solution that fails test cases cannot be considered reliable or functional.

In addition to correctness, efficiency is a critical metric. It assesses the performance of the AI's solutions in terms of runtime complexity and resource utilization. Problems involving large inputs or computationally intensive operations are particularly valuable in highlighting whether the AI can generate solutions that meet the expected efficiency constraints, such as adhering to optimal time and space complexity.

The **readability and structure** of the code are also important factors in the evaluation. While not strictly necessary for functional correctness, well-structured and readable code is essential for maintainability and collaboration in professional software development. This criterion examines whether the AI produces solutions that follow standard programming practices, such as proper variable naming, modular design, and clear logical flow.

To provide a comprehensive assessment, the study also incorporates metrics for **problem-solving completeness**. This involves evaluating whether the AI model effectively interprets the problem statement and generates a fully implemented solution, as opposed to incomplete

or partially functional outputs. Failure to address all aspects of the problem, even if some test cases pass, is considered a limitation of the model.

Finally, consistency is monitored across multiple problem-solving attempts to evaluate the reliability of the models. This ensures that results are not anomalous and that the AI performs consistently across diverse tasks and repeated runs. By combining these metrics, the evaluation framework provides a holistic picture of the AI's capabilities, strengths, and limitations in solving programming problems.

# 3 Results and Analysis

## 3.1 Quantitative Results

Performance metrics (e.g., success rates on easy, medium, and hard problems).

Comparison of models based on metrics such as speed, efficiency, and accuracy.

Visual representation of results (tables and graphs showing success rates by difficulty).

## 3.2 Qualitative Assessment

Analysis of how the models approach problem-solving (e.g., whether they use brute force, optimal solutions, or heuristics).

Examples of well-solved problems and those where AI models failed.

Discussion of common errors (e.g., logical issues, misunderstanding problem statements, or incomplete outputs).

## 3.3 Comparison to Human Performance

How AI models perform in comparison to average Leetcode users (use most common solution).

Cases where AI outperformed humans (e.g., faster execution of simple problems).

Scenarios where humans excelled.

## 3.4 Statistical Analysis

Maybe some scientific methods to analyze the data.

# 4 Implications and Future Directions

How AI models can augment current software engineering workflows (e.g., code generation, debugging, and testing).

Potential changes to roles in the industry (e.g., humans focusing on higher-level tasks while AI handles routine coding).

Use of AI in education or recruitment (e.g., automated assessment of programming skills).

## 4.1 Implications of Findings for Software Engineering

### 4.1.1 Selection and Categorization of Programming Problems

Impact of problem diversity on AI performance.

Lessons learned about problem selection and its role in meaningful AI evaluation.

Recommendations for creating better problem sets in future research.

### 4.1.2 Characteristics of the Problem Set

How problem complexity affects AI performance.

Importance of testing AI models on problems that mirror real-world software engineering challenges.

The characteristics of the problem set play a crucial role in evaluating the true capabilities of AI models, as the complexity and nature of problems directly influence their performance. The problems used in this study were carefully curated to cover a wide range of scenarios that reflect both theoretical concepts and practical applications in software engineering.

Problem complexity significantly affects AI performance. Simpler problems, often categorized as "easy," typically involve straightforward logic or basic algorithmic steps, such as iterating through an array or performing simple mathematical calculations. These problems serve as a baseline to assess the AI's ability to produce syntactically and logically correct solutions. AI

models generally perform well on these tasks, as they require minimal contextual understanding and rely on commonly observed programming patterns.

In contrast, more complex problems introduce layers of difficulty that challenge the AI's ability to reason, interpret, and optimize. Medium-difficulty problems often require intermediate skills such as recursion, efficient use of data structures like trees or heaps, or solving problems with moderate constraints. Hard problems push these boundaries further by demanding advanced algorithmic solutions, deep contextual understanding, and the ability to manage multiple interdependent variables. These challenges expose limitations in AI models, such as struggles with abstraction, lack of creativity in designing novel solutions, and inefficiency when faced with computationally intensive tasks.

The inclusion of problems that mirror real-world software engineering scenarios is crucial for meaningful evaluation. Real-world challenges often involve incomplete or ambiguous problem statements, require multi-step reasoning, or necessitate balancing competing priorities like performance and readability. By incorporating such problems, the study ensures that the evaluation goes beyond textbook scenarios to assess the AI's practical utility. For example, problems that require handling edge cases, implementing scalable solutions, or integrating with existing systems offer valuable insights into the model's readiness for deployment in professional environments.

This diverse and thoughtfully constructed problem set enables a comprehensive understanding of the strengths and weaknesses of AI models. It highlights not only their technical capabilities but also their potential limitations when faced with real-world complexity, ultimately guiding future research and development in AI-driven programming tools.

## 4.2 Challenges and Limitations

Limitations of the study (e.g., focus on Leetcode, which may not reflect real-world scenarios).

Challenges in evaluating non-deterministic AI outputs.

Biases in the dataset or prompts that may influence AI performance.

## 4.3 Future Potential

The evolution of AI in software engineering presents a range of opportunities for advancing its role beyond current capabilities. As these models grow more sophisticated, they may achieve a deeper understanding of problem contexts, allowing them to address challenges involving ambiguity or incomplete information with greater precision. By refining their ability to interpret complex requirements, AI systems could become indispensable for tackling tasks that demand both technical knowledge and contextual reasoning.

In addition to independent problem-solving, AI has the potential to become a more effective collaborator in team-based programming environments. By integrating seamlessly into workflows, these models could assist developers in real time, suggesting optimizations, identifying errors, and improving overall code quality. Such advancements would position AI as a trusted partner in the development process, augmenting rather than replacing human expertise.

Another promising direction lies in enabling AI to handle large-scale, real-world projects. While current models are adept at solving isolated problems, future systems could extend their functionality to manage multi-file software projects with complex dependencies. This would open doors to applications in fields such as enterprise software development, where AI could assist in creating, maintaining, and scaling systems efficiently.

The expansion of benchmarks and evaluation frameworks will also play a critical role in shaping the trajectory of AI in programming. By introducing benchmarks that better reflect real-world challenges, such as collaborative tasks or system design problems, researchers can encourage the development of models that align closely with industry needs. This will help bridge the gap between academic research and practical applications, ensuring AI evolves in ways that are directly beneficial to software engineering.

The ethical implications of relying on AI for critical systems also demand attention. As these systems become more capable, it will be essential to address concerns such as accountability, transparency, and the potential displacement of human roles. By prioritizing responsible development and deployment, AI can be integrated into software engineering in a way that maximizes its benefits while minimizing risks.

Future advancements in AI for software engineering are likely to reshape the profession in profound ways, enhancing productivity, enabling new possibilities, and driving innovation across the field. As these systems continue to evolve, their role will become increasingly central to the way software is designed, developed, and maintained.

Suggestions for improving AI models (e.g., better training on real-world coding patterns, handling ambiguities).

Expanding evaluations to include team-based programming or larger-scale projects.

Creation of additional benchmarks to assess AI performance in different software engineering tasks (Use my old ideas).

Should it be followed by an Appendix?

# 5 Conclusion

**To Be Done**

„Hereby, I declare that I have independently prepared the preceding work. All passages that have been quoted or paraphrased have been marked as such.“

Andrii Zhabrovets

# List of Figures