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

1.1 Motivation

The rapid growth of artificial intelligence (AI) has sparked widespread enthusiasm about its potential to revolutionize software engineering. Among the many areas poised for transformation, programming stands out as particularly exciting. AI's capability to understand, generate, and debug code presents opportunities to streamline development processes, assist engineers, and potentially take over certain tasks traditionally performed by humans. However, these advancements also prompt important questions about the reliability, efficiency, and adaptability of AI-driven solutions. To address these questions, it is crucial to develop rigorous methods of evaluation. This research is motivated by the need to systematically assess the real-world programming abilities of current AI models, providing a structured framework to gauge their performance.

1.2 Research Objectives

This study proposes a benchmark specifically designed to evaluate AI models' proficiency in handling programming tasks, with an emphasis on solving algorithmic challenges. The benchmark incorporates a curated collection of well-known problems that mimic real-world scenarios encountered by human developers. By using standardized metrics such as execution time, memory usage, and accuracy, the study aims to capture the technical capabilities of AI in coding. While the primary focus is on assessing the models' ability to solve isolated algorithmic problems, the study intentionally excludes broader considerations like code readability, maintainability, or integration into complex software systems. These dimensions require different evaluation methods and are beyond the scope of this work. By concentrating on algorithmic performance, the benchmark offers a clear and focused view of current AI capabilities. This structured approach provides a solid foundation for future research, paving the way for more comprehensive evaluations and applications.

1.3 Scope of the Research

This paper provides an in-depth exploration of the research, starting with the motivations for assessing AI's programming potential. The introduction outlines the benchmark's scope, objectives, and areas of focus, establishing the groundwork for the study. Following this, the next section delves into the background and related work, offering context on AI in software engineering and the role of benchmarks in evaluating AI capabilities. The paper then transitions to a detailed discussion of the benchmark's design and implementation, encompassing problem selection, technological tools, and evaluation metrics. Results and analysis follow, highlighting key findings, identifying patterns, and discussing unexpected insights. A dedicated section addresses challenges encountered during the research process and the adaptations made to overcome them. The paper concludes with a discussion of the broader implications of the findings, limitations of the benchmark, and its potential for future research. Finally, forward-looking recommendations suggest strategies for expanding and improving the benchmark, including exploring additional AI models and refining evaluation methodologies.

2 Background and Related Work

2.1 Overview of AI in Software Engineering

Artificial intelligence (AI) in software development involves using machine learning and automation to streamline programming workflows, improve efficiency, and in some cases, replace tasks traditionally performed by human developers. Over the past few years, AI has rapidly advanced, becoming an integral part of modern software engineering. Tools such as GitHub Copilot, powered by OpenAI Codex, and ChatGPT have gained widespread recognition for their ability to assist in generating, debugging, and optimizing code. In comparison to more standard coding utilities chat like tools, GitHub Copilot integrates directly into development environments, offering contextual code suggestions that help programmers save time and reduce errors during repetitive tasks.

Similarly, advanced models like Claude and Gemini, which are considered one of the main competitors to ChatGPT, despite being general use-oriented models are also designed to handle complex programming challenges, showcasing their capacity for nuanced problem-solving and software design. These developments highlight the growing potential of AI to transform the field, not just as an assistant but also as a tool capable of overtaking a broader range of programming responsibilities, raising rather controversial questions about its role in the future of software engineering.

Adding to this landscape, Devin AI was introduced by Cognition Labs in early 2024 as the world's first autonomous AI software engineer. Devin was advertised as being able to perform tasks such as coding, debugging and project completion on its own, sparking considerable interest in the tech community. However, subsequent analysis revealed inconsistencies between these claims and Devin's actual performance.

Thus, as for today, Artificial intelligence is still seen a tool, with professionals in software engineering fields having a very skeptical attitude towards the idea of independent AI-powered utilities that can transform the client's ideas to a fully functional product

2.2 Benchmarking for Programming AI

Benchmarks are crucial for assessing the capabilities and limitations of AI models. They provide standardized metrics to evaluate an AI's performance in specific areas, such as natural language processing (NLP) or programming. In NLP, benchmarks test how well AI systems understand and generate human language, offering insights into their practical applicability. For programming, benchmarks like HumanEval and MBPP (Mostly Basic Python Problems) have been used to measure AI performance. However, these benchmarks often focus on isolated tasks that lack the complexity of real-world programming scenarios. To bridge this gap, this study introduces a tailored benchmark that evaluates AI models on multiple dimensions, including runtime efficiency, memory usage, and solution correctness. By emphasizing real-world coding conditions and incorporating diverse challenges, this benchmark aims to provide a more comprehensive and realistic assessment of AI capabilities in software development.

2.3 LeetCode and Programming Problem Sources

LeetCode is one of the most popular platforms for coding challenges, widely regarded as a gold standard among both aspiring and professional developers. Known for its extensive library of algorithmic and data structure problems, LeetCode has become a preferred resource for honing programming skills and preparing for technical interviews. Its reputation for offering high-quality, industry-relevant problems made it the ideal choice for this study's benchmark. The platform's diverse problem set spans a range of difficulty levels—easy, medium, and hard—allowing for a structured and balanced evaluation of AI performance. By sourcing tasks from LeetCode, this benchmark ensures that the problems reflect practical coding scenarios while maintaining rigorous standards. The categorization of tasks by difficulty further enhances the benchmark's ability to measure the capabilities of AI models across varying levels of challenge, providing meaningful insights into their real-world applicability.

3 Benchmark Design and Implementation

3.1 Problem Selection and Categorization

During the selection process of the problems which would then be used for assessing the capabilities of AI models many aspects could be put at the priority: difficulty, variety, popularity. Taking int account the main goal of this research, the selection of problems should be based solely on how often they appear on the interviews. On the LeetCode platform, problems were filtered by the category Top Interview Questions, focusing on tasks considered crucial for professional software development. From this refined selection, the five most popular problems from each difficulty level—easy, medium, and hard—were chosen. This method ensured that the benchmark incorporated tasks ranging from fundamental algorithms to complex scenarios, as both the usage of only difficult or only simple problems would not represent fully abilities of the models.

Furthermore, choosing well-known and frequently attempted problems enhances the benchmark's credibility and relevance. These tasks have been extensively solved and discussed by the programming community, providing the most refined solutions and this way. This factor eliminates any chance of low-key errors, which therefore makes such solutions perfect for using them as a reference. Emphasizing popularity and a balance of difficulty levels, the benchmark achieves both accessibility for AI models and a robust evaluation of their programming capabilities.

3.2 Technologies Used

A variety of technologies were used to effectively design and implement the benchmark. The programming language Python was chosen for the development of this benchmark because of its wide range of modules for working with data and its widespread use in AI development.

For each, the OpenAI API was used to integrate models such as GPT-3.5 and GPT-4, with plans to include Anthropic's Claude and Google's Gemini in future updates. In fact, the availability of all the required APIs played a crucial role in the choice of programming language.

Of all the Python modules used in this project, one called "pandas" was used most often to organise, process and summarise the benchmark results, which were then exported to Excel for easy review and further visual processing.

Visualisations of performance metrics were created using the matplotlib module.

Execution of both AI-generated and LeetCode-parsed solutions was handled using the subprocess for running isolated blocks of code

Two built-in modules were used for monitoring, tracemalloc for memory tracking, time for runtime measurement.

3.3 AI Models Evaluated

The benchmark evaluated some of the leading AI models in the field, including GPT-3.5 and GPT-4 from OpenAI, Claude from Anthropic, and Gemini from Google. These models were chosen based on their popularity, accessibility, and proven capabilities in code generation and software engineering tasks.

GPT-3.5 and GPT-4: These OpenAI models represent state-of-the-art advancements in natural language understanding and generation.

Claude: Developed by Anthropic, Claude emphasizes safety and a nuanced understanding of prompts, making it a strong contender in programming tasks.

Gemini: Created by Google, Gemini is designed to tackle complex problem-solving tasks with an emphasis on general intelligence.

The inclusion of these models ensures that the benchmark provides a well-rounded and comprehensive comparison of the latest AI systems in a standardized testing environment.

3.4 Benchmark Workflow

The benchmark workflow was carefully structured to evaluate the performance of AI and human solutions on selected programming tasks. The process begins with loading problems from the

LeetCode dataset and corresponding test cases stored in JSON files. AI solutions are generated using pre-defined prompts, while human solutions are sourced from existing implementations.

Each generated solution is executed against predefined test cases using subprocess, ensuring isolation and consistent execution. Memory usage is tracked with tracemalloc, and metrics such as runtime duration and error outputs are recorded to assess the correctness and performance of each solution. Aggregated metrics are then analyzed and visualized to facilitate clear comparisons between AI and human performance.

This structured and repeatable workflow guarantees consistency and transparency, enabling fair comparisons between different AI models and human baselines.

3.5 Evaluation Metrics

When evaluating the performance of AI models in solving programming problems, a set of standardized metrics is essential to ensure objectivity and consistency. The benchmark uses a variety of metrics to thoroughly evaluate the performance of both AI and human-generated solutions. These metrics ensure a well-rounded assessment of programming capabilities.

3.5.1 Success Rate

This metric determines how many test cases each solution successfully solves, expressed as a percentage. It serves as the primary measure of correctness and overall effectiveness.

3.5.2 Runtime Performance

This measures the time it takes for each solution to execute, expressed in milliseconds. Faster runtimes indicate better computational efficiency, which is crucial in performance-critical scenarios.

3.5.3 Memory Usage

Peak memory consumption during the execution of each solution is monitored in bytes using tracemalloc. This metric assesses the resource efficiency of the implementations, particularly important for systems with limited memory resources.

4 Results and Analysis

This section discusses the findings from the benchmark, focusing on quantitative outcomes, comparative analysis of AI model performance, and the insights that emerged from the data.

4.1 Quantitive Results

4.1.1 Success Rates

Figure 1 illustrates the success rates of the evaluated AI models and the LeetCode baseline among all the problems. There is no suprise that LeetCode solutions achieved the highest success rate, as all the solutions were added programmably. Among the AI models, GPT-3.5 and GPT-4 performed well, with slight differences in handling edge cases and adhering to given templates. Claude and Gemini followed closely, maintaining comparable success rates, showcasing their reliability in solving diverse tasks.

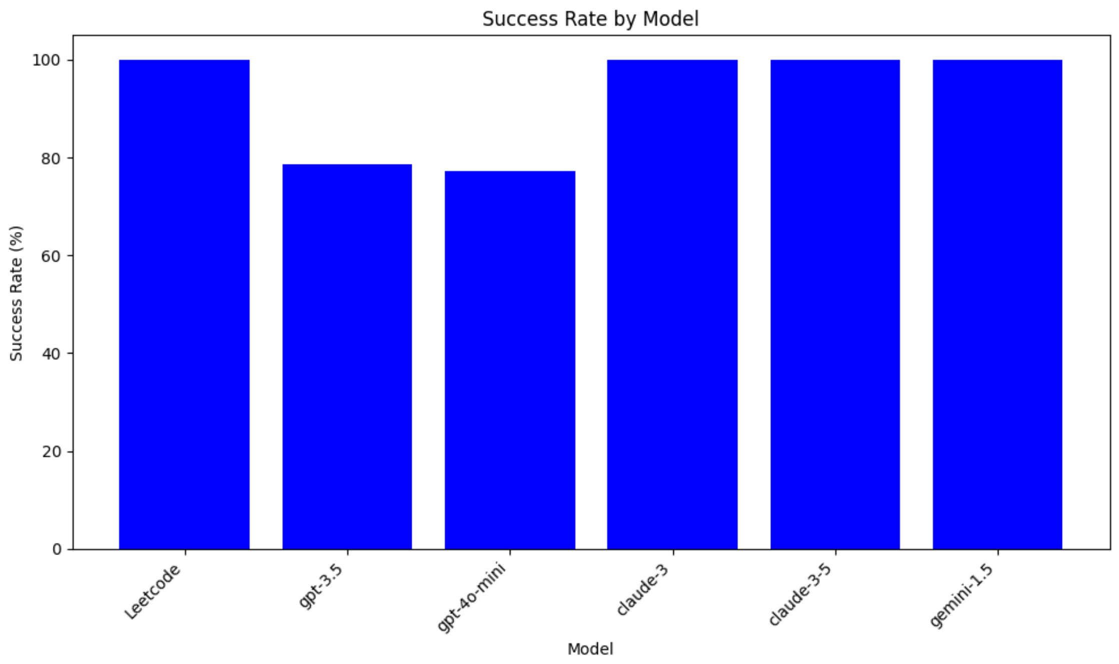

Figure 1: Success Rate by Model

4.1.2 Runtime Performance

As shown in Figure 2, LeetCode solutions exhibited the fastest runtime performance, indicating their computational efficiency. In contrast, GPT-4 exhibited slightly slower performance than GPT-3.5, potentially due to its more complex problem-solving mechanisms. Claude and Gemini demonstrated moderate runtime, emphasizing their balanced optimization between speed and accuracy.

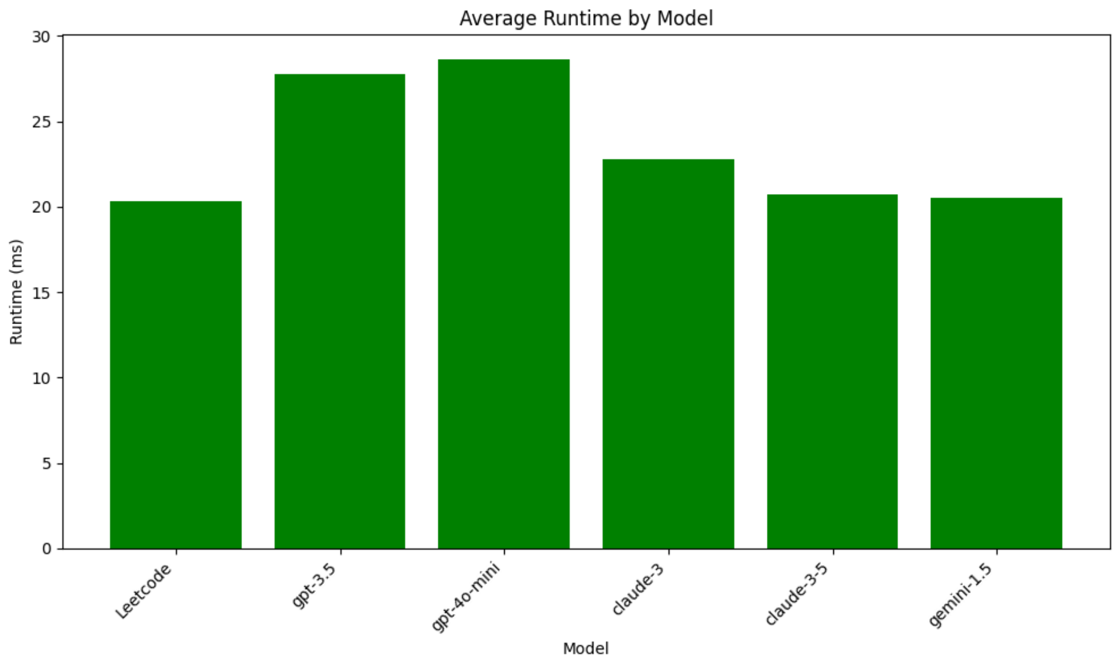

Figure 2: Average Runtime by Model

4.1.3 Memory Usage

Figure 3 highlights the average memory consumption of each model. GPT-3.5 and GPT-4 had the highest memory usage, reflecting their reliance on extensive context processing. Claude and Gemini consumed less memory, showing their efficiency in managing resource utilization. LeetCode solutions required the least memory, indicating their lightweight implementations.

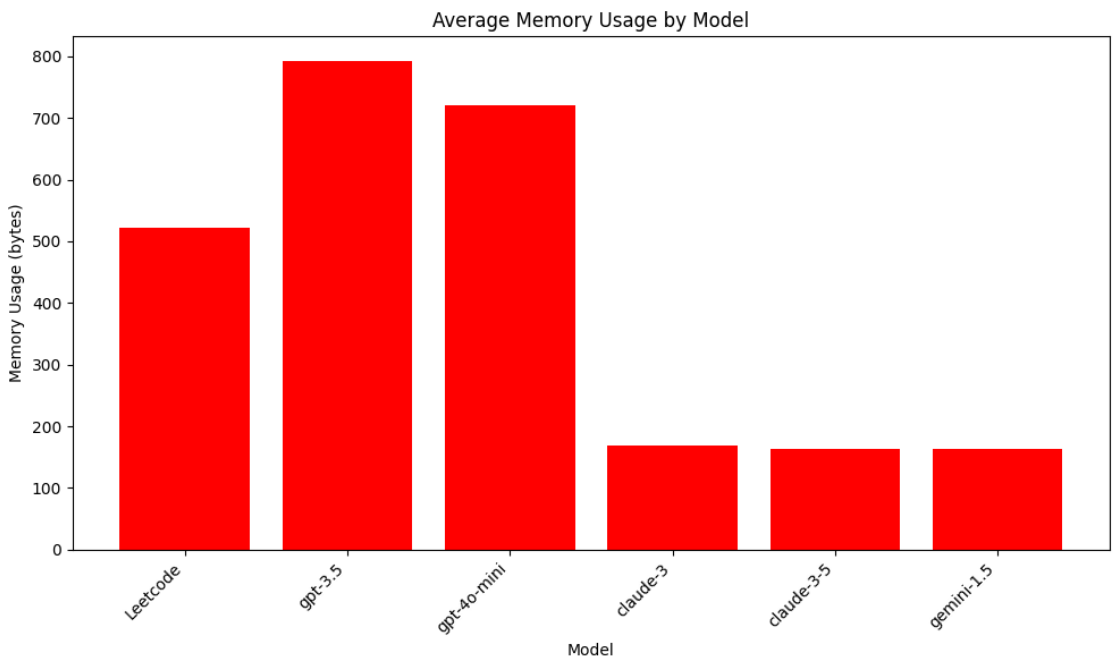

Figure 3: Average Memory Usage by Model

4.2 Comparative Performance

The comparative analysis revealed the trade-offs between speed, accuracy, and memory efficiency across the models. LeetCode solutions set a robust baseline with high accuracy and optimal resource utilization. GPT-3.5 and GPT-4 stood out for their advanced code generation capabilities but required higher computational resources. Claude and Gemini provided competitive performance with slightly lower resource demands, making them suitable for scenarios where memory efficiency is critical.

4.3 Patterns and Insights

Several patterns emerged from the analysis. First, the AI models demonstrated consistent success rates across easy and medium problems but showed variability on harder problems, particularly when edge cases or complex logic were involved. This indicates that while AI models excel at standard tasks, they may struggle with unconventional scenarios that require deeper reasoning or specialized knowledge.

Second, runtime and memory usage varied significantly between models. Models like GPT-4, which prioritize contextual understanding, tend to use more memory and exhibit slower runtimes compared to leaner models like Claude. These differences suggest that model selection should align with specific task requirements, balancing efficiency and accuracy.

Overall, the results underscore the growing potential of AI in programming while highlighting areas for improvement. The insights gained from this benchmark provide a foundation for refining AI systems and optimizing their application in real-world software engineering contexts.

5 Challenges and Adaptations

5.1 Encountered Issues

During the development of the benchmark, several challenges were encountered that required adaptations to the initial plan. The primary issues were related to the adaptation of prompt formats, the handling of completely non-functional solutions, and the execution of generated code.

5.1.1 Generation of Unnecessary Text

The main problem that was spoiling the results of AI models was the generation of unnecessary text. This issue was encountered in the case of gpt-3.5 and gpt-4o AI models. The models were always tring to generate additional unnecessary inroduction message, despite of all the restrictions given in the prompt. Even after improving the restrictions, GPT models were giving out the code solution with Markdown syntax, which is used for inserting code blocks in Markdown files.

```
‘‘‘python
def twoSum(nums, target):
    num_to_index = {}
    for index, num in enumerate(nums):
        complement = target - num
        if complement in num_to_index:
            return [num_to_index[complement], index]
        num_to_index[num] = index
‘‘‘
```

5.1.2 Wrong Code Placement

Another issue that has been encountered was the unwillingness of the AI models to generate code that would be able to solve the problems in a proper place. Despite, writing the code, which could potentially work, AI models were completely neglecting the given template with of which they where supposed to work.

```python
def twoSum(nums, target):
    pass

def two_sum(nums, target):
    num_to_index = {}
    for index, num in enumerate(nums):
        complement = target - num
        if complement in num_to_index:
            return [num_to_index[complement], index]
        num_to_index[num] = index

```

5.2 Solutions Implemented

Despite a seemingly difficult nature of these models' behavior, the solutions to these problems were quite simple in concept, An additional set of restrictions was set on all the prompts to make sure no extra text was added to the solution.

The issue with Markdown syntax was solved by adding a simple function that would check the first and the last line of the generated code and remove any unnecessary text.

The misplacement problem was solved by adding a comment to the template that states that the code should be written in it's place. Example:

```python
    def twoSum(nums, target):
        # Write your code here

```

5.3 Dropped Approaches

During the development of the benchmark, several ideas were tested on how to execute the code solutions generated by the AI models. Adhering to the principle of using as few external modules as possible for the sake of easier codebase support in the future, the first idea was to import each solution as a module. However, the implementation of even the most basic syntax error detection proved to be unnecessarily complicated. After the decision was made to measure memory usage as well as runtime, the idea of importing each solution as a module was dropped. The main reason for this was that the memory usage of the imported module would be affected by the main process. This would make it impossible to measure the memory usage of the solution itself. That's why the final approach was to run each solution in a separate sub-process, which would allow the memory usage of the solution itself to be measured. In addition, this approach made debugging much more convenient.

6 Discussion

6.1 Capabilities and Limitations of AI Models

The benchmark showcased AI's ability to handle algorithmic challenges effectively but highlighted limitations in adaptability and robustness. Models like GPT-4 excelled in generating syntactically correct code but struggled with edge cases and template adherence.

6.2 Implications for Software Engineering

The results of the benchmark showed an almost perfect success rate for all problems and AI models. However, there are still many problems to consider. As mentioned earlier, the fact that all the problems were taken from a popular internet source may have played a crucial role in this research. However, it would be premature to say that these results prove that AI is about to overtake all software development jobs.

A parallel can be drawn with the real interview process. Can we consider a student who has practised a lot of programming problems to be a suitable candidate for a developer role? This is definitely not an easy question to answer, as the job of a software engineer requires other skills, many of which need to be tested in a more complex way. An example of this would be the ability to not only write correct code, but also to work with other people's code, find and fix bugs in it, and so on. In addition, a developer often has to work with external modules, which requires the ability to analyse their documentation.

Ultimately, the results of this benchmark have only revealed the upper level of the AI model's capabilities, and taking into account its success in this test, the next stage of experimentation can and should be carried out.

6.3 Limitations of the Benchmark

The benchmark's focus on algorithmic problems excludes other critical aspects of software engineering, such as maintainability and scalability. Expanding the scope to include these dimensions would provide a more holistic evaluation.

7 Future Directions and Implications

7.1 Expanding the Benchmark

Expanding the scope of the benchmark is a crucial step for enhancing its relevance and comprehensiveness. While the current benchmark primarily focuses on solving algorithmic problems, programming in real-world settings encompasses a much broader range of tasks. Introducing debugging problems, for example, could assess an AI's ability to identify and fix errors in existing code, a skill that is fundamental to professional software development yet remains underexplored in most AI evaluations.

Another area of expansion could involve problems requiring the use of external libraries and APIs. These tasks would evaluate how effectively AI models can understand and incorporate pre-existing tools and frameworks into their solutions. Additionally, optimization problems, where the goal is not merely to find a correct solution but the most efficient one, would push AI models to demonstrate advanced resource management and algorithmic ingenuity. By diversifying the problem set to include such challenges, the benchmark could offer a more realistic and nuanced evaluation of an AI model's capabilities in practical programming scenarios.

7.2 Evaluating More AI Models

The dynamic nature of AI development necessitates the continuous inclusion of new models to ensure the benchmark remains relevant. While the current version evaluates prominent models like OpenAI's GPT-4, Anthropic's Claude, and Google's Gemini, future iterations should explore additional emerging models. These might include Meta's LLAMA, specialized models optimized for specific tasks, and other state-of-the-art frameworks developed by research institutions and industry leaders.

By broadening the pool of AI systems under evaluation, the benchmark would provide richer insights into the strengths and weaknesses of different models. This expansion would help in identifying trends across diverse architectures and uncovering areas where further development is needed. Evaluating a variety of models also ensures the benchmark remains a comprehensive tool for analyzing advancements in AI-driven programming.

7.3 Improving Metrics metrics and Evaluation

Refining the evaluation metrics is another essential step in advancing the benchmark's effectiveness. Automating Big O analysis, for instance, could add a deeper layer of assessment by measuring the computational efficiency and scalability of AI-generated solutions. This metric would provide critical insights into the suitability of solutions for real-world applications that involve large datasets or time-sensitive processes.

Stress testing is another promising area for improvement. By running solutions under extreme conditions, such as edge cases or unusually large inputs, the benchmark could evaluate their robustness and resilience. This form of testing would be particularly valuable for assessing how well solutions handle unexpected scenarios, which is often a key requirement in production environments.

Moreover, extending the benchmark to support multiple programming languages would significantly enhance its scope. Evaluating AI models across different languages and coding paradigms would offer a more holistic view of their adaptability and versatility. As modern software engineering often requires proficiency in diverse languages and frameworks, this enhancement would align the benchmark more closely with real-world programming demands.

7.4 AI Agents for Software Development

The emergence of AI agents presents an exciting frontier for the benchmark. Unlike traditional models that generate static solutions, AI agents can dynamically interact with external tools, APIs, and environments to solve problems. For example, an AI agent could conduct web searches to gather additional context, leverage APIs to fetch real-time data, or utilize integrated development environments (IDEs) to execute and debug code interactively.

Testing such agents would require a significant rethinking of the benchmark's structure. New metrics would need to be developed to evaluate their performance effectively, such as the efficiency of multi-step interactions, the accuracy of context-based problem-solving, and the ability to adapt to dynamic requirements. By exploring these advanced capabilities, the benchmark could help illuminate how AI might transition from static problem-solving to dynamic, context-aware programming assistance, further advancing its role in modern software engineering.

8 Conclusion

This research has explored the evolving capabilities of AI in software development, focusing on benchmarking AI models against programming problems to evaluate their potential to substitute or complement human developers. The findings highlight both the strengths and limitations of AI systems like GPT-3.5, GPT-4, Anthropic's Claude, and Google's Gemini. While these models excel at generating syntactically correct and often efficient code, their performance is constrained by challenges such as adherence to given function templates, handling edge cases, and generating solutions that are robust under varying conditions.

The benchmark developed in this study provides a comprehensive framework for evaluating AI performance across a range of problem types and difficulty levels. By incorporating metrics like success rate, runtime, and memory usage, the tool offers a nuanced understanding of AI efficiency and effectiveness. The inclusion of multiple AI models has also underscored the diversity of approaches in this domain, revealing areas where individual systems excel or fall short.

However, the study acknowledges the limitations inherent in both the benchmark and the AI models themselves. While the focus has been on algorithmic problems, real-world programming involves a broader spectrum of tasks, including debugging, integration with external tools, and optimization—areas that remain underexplored in this research. These gaps present exciting opportunities for future enhancements, such as expanding the benchmark to include more complex problem types and evaluating emerging AI systems with advanced capabilities.

The implications of these findings extend beyond academic curiosity. As AI continues to evolve, its integration into software development processes promises to reshape the industry. Developers may increasingly rely on AI not just as a coding assistant but as a partner capable of handling complex and dynamic tasks. At the same time, the limitations observed in this study highlight the importance of human oversight, creativity, and expertise—qualities that remain irreplaceable.

In conclusion, this work contributes to the growing body of knowledge on AI's role in programming by providing a rigorous and adaptable benchmarking tool. It lays the groundwork for further exploration into AI’s capabilities and limitations, offering a foundation for both academic research and practical advancements in the field. The future of AI in software engineering holds immense promise, and this study represents a step toward understanding and harnessing that potential.

„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