LLM Defect Prediction Analysis

Introduction

In the field of artificial intelligence, understanding the capabilities of advanced language models like ChatGPT-3.5 Turbo and GPT-4 Turbo, particularly in programming problems is crucial. This study focuses on a critical aspect: whether these models have developed a deep understanding of different problems or are merely relying on memorization. Addressing this issue, the research specifically evaluates these AI models' ability to predict defects in Competitive Programming (CP) problem solutions.
While previous research has concentrated on the problem-solving abilities of AI, this paper examines their capacity for solution analysis. The shift from problem-solving to problem evaluation might offer a new metric for AI performance evaluation.
This paper seeks to give fresh insights into AI's depth of understanding in CP.

Related Work

The field of competitive programming (CP) and the involvement of AI, specifically language models, has seen significant developments, as highlighted by the creation and performance of systems like AlphaCode by DeepMind. In this context, the current study's approach, focusing on evaluating the efficacy of language models in determining the quality of CP solutions, offers a unique perspective compared to existing works.
AlphaCode represents a notable advancement in the domain, achieving competitive levels in programming contests. It uses transformer-based language models for generating code and is capable of solving problems that require critical thinking and a comprehensive understanding of algorithms and coding. Its performance was validated in competitions on the Codeforces platform, placing it at about the median competitor level. This achievement underscores the potential of AI in not just generating code, but in tackling novel problems that demand a deep understanding of programming concepts.
However, while AlphaCode and similar systems demonstrate the ability to generate code at a competitive level, the focus has predominantly been on problem-solving capabilities. The current study diverges by specifically investigating the ability of AI models to evaluate and critique code solutions in CP. This is a crucial distinction, as the ability to generate solutions is different from the ability to critically analyze and assess the quality of those solutions.

Árpád Csizmadia

In summary, while existing works like AlphaCode have shown that AI models can successfully solve CP problems, there is a gap in understanding how these models perform in evaluating the quality of CP solutions. The current study aims to fill this gap by offering a novel perspective and contributing to the broader understanding of AI's capabilities in competitive programming.

Materials

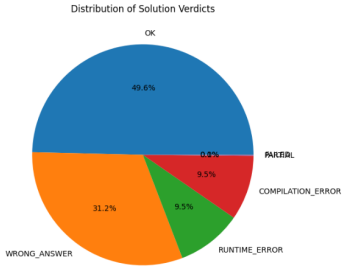
The dataset for this research was sourced from Codeforces, a renowned platform for competitive programming. The data comprises problems and solutions from two distinct subsets within the platform, categorized as follows:

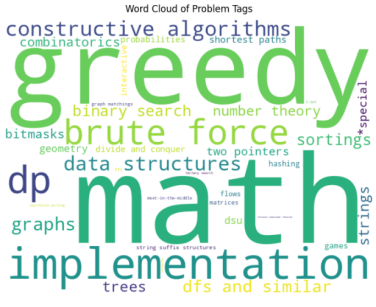
- 'x10' Folder: Contains 1022 problems with at least 10 solutions.
- 'x100' Folder: Encompasses 124 problems with at least 100 solutions.

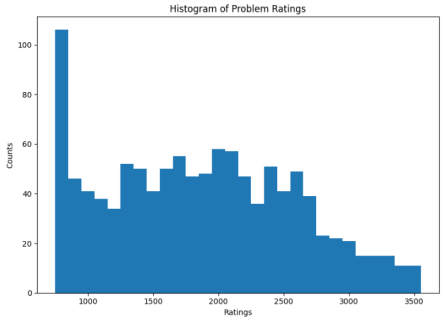
This results in a total of 1146 problems and 24679 solutions. The data extraction involved utilizing Codeforces' API for problem and solution retrieval, supplemented by web scraping techniques to obtain detailed problem descriptions, a task for which a custom Python script was developed.

Each dataset entry includes comprehensive information such as problem ID, contest details, creation time, problem specifics (including name, type, difficulty rating, and associated tags), and author information. Solutions are accompanied by metadata including programming language, verdict, performance metrics (e.g., test counts, execution time), and the source code itself.

Preprocessing of the data was necessary to ensure usability in the research context. This preprocessing involved data cleaning, structuring, and format normalization to facilitate effective analysis and experimentation.







Reproducibility

Árpád Csizmadia

The given program and colab sheet is designed to be repeatable and detailed, allowing reproducibility.
To reproduce the results of this paper do the following:

1. Check out "https://github.com/arpcs/defect-predictor":
    a. git clone https://github.com/arpcs/defect-predictor.git
2. Run the install script
3. Add a file named "api_key.txt" in the parent directory of defect-predictor containing your OpenAI api key.
4. Run the main.py script
5. Run in Google Colab sheet:
    a. "https://colab.research.google.com/drive/1a9DGMOFYa5ISYXD7dFsK7pohdA8GsUNI#scrollTo=56vHt0zOMBWY"
    AI - Defect Predictor.ipynb

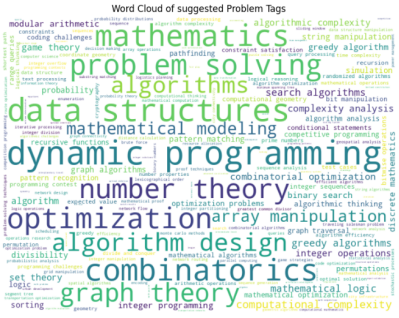
Methods

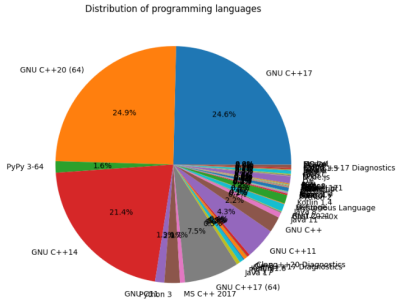
Implementation

The research methodology involved a multi-step process, primarily focusing on the evaluation of AI's ability to analyze competitive programming solutions. The steps were as follows:

Data Collection: The initial phase involved fetching a comprehensive dataset from Codeforces, an online platform for competitive programming. This dataset included a variety of problems and their corresponding solutions.

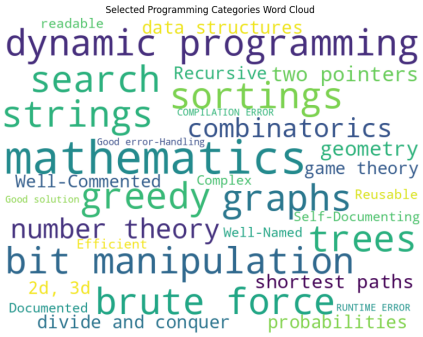
Categorization via ChatGPT-3.5: Utilizing OpenAI's ChatGPT-3.5, categories were suggested for a selected subset of solutions and problems.



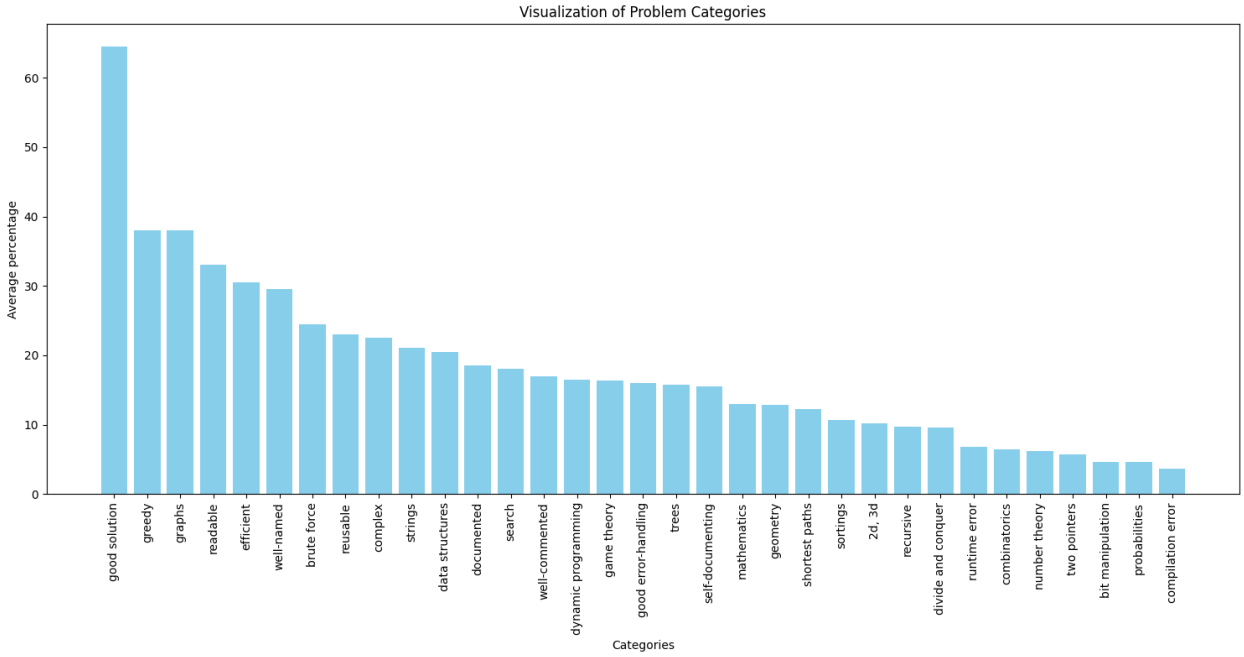


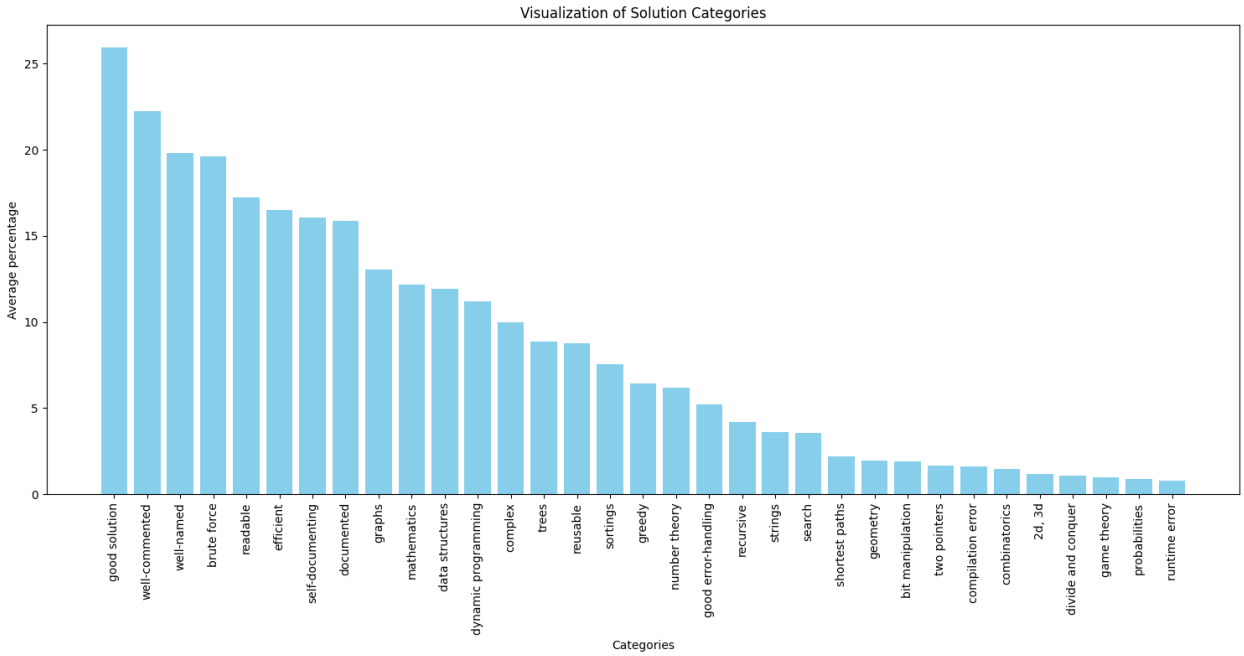
Data Synthesis: The suggested categories from ChatGPT-3.5 were then compared and synthesized with the categories provided by the Codeforces

dataset. Then a manual selection was made based on the generated data, removing duplicates.



Problem and Solution Selection: From the dataset, a specific set of 10 problems, each with 10 corresponding solutions, were chosen for detailed evaluation. All selected problems and solutions were in C++ programming language.

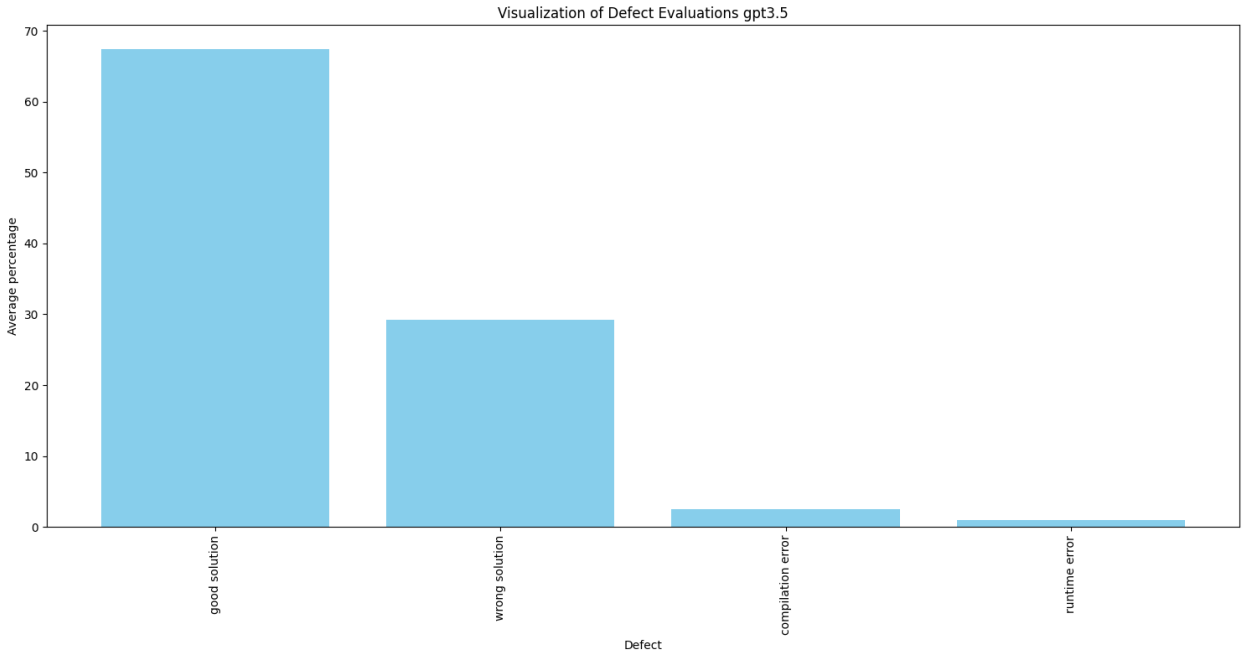


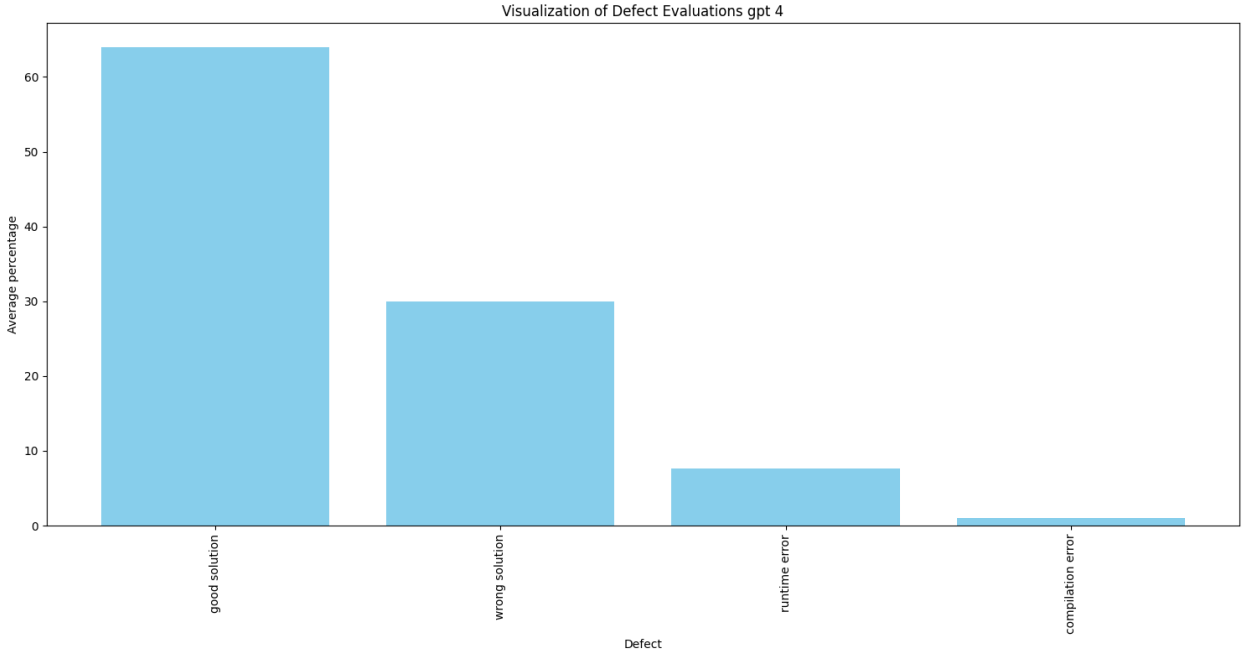


Experimental Setup

The experimental phase was structured to assess the accuracy and depth of AI's evaluative capabilities in relation to competitive programming solutions.

> **Solution Quality Assessment**: The next step involved presenting both the problem and its solutions to ChatGPT 3.5 and 4, asking it to evaluate the likelihood of each solution being correct, or resulting in errors (wrong answer, runtime error, compile-time error).





System and Tool Utilization

Throughout the research, several tools and systems were utilized:

- Codeforces API: Used for data extraction.
- Python Scripting: For data fetching and processing.
- ChatGPT-3.5, ChatGPT-4: Employed for categorization and evaluative analysis.
- Google Colab: Used for statistical analysis and metric evaluation.

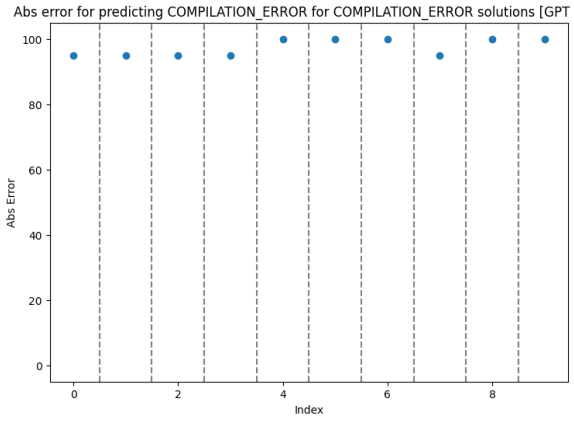
Analysis and Results

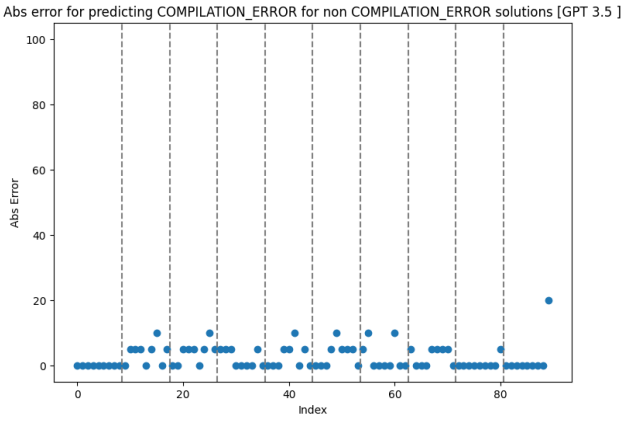
Statistical Analysis and Metrics Evaluation

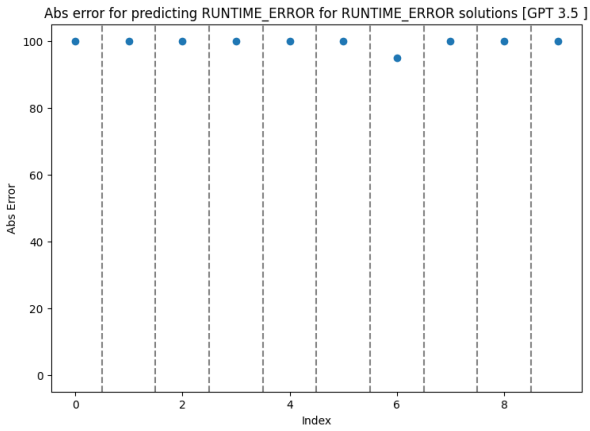
Using Google Colab, a range of statistical analyses and metrics were evaluated to gain an in-depth understanding of the performance of GPT-3.5 and GPT-4 in competitive programming problem-solving.

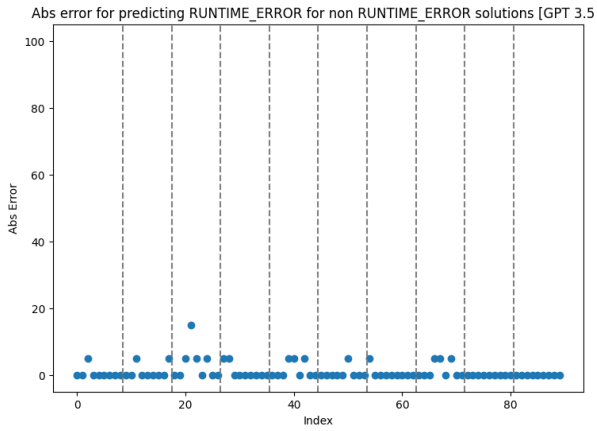
Runtime and Compilation Errors Analysis

- **GPT-3.5**: The model consistently assumed that solutions would compile and run without errors, showing an inability to accurately predict runtime and compilation errors.

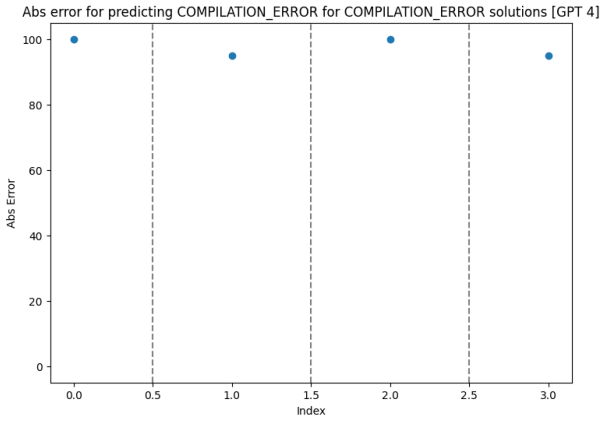


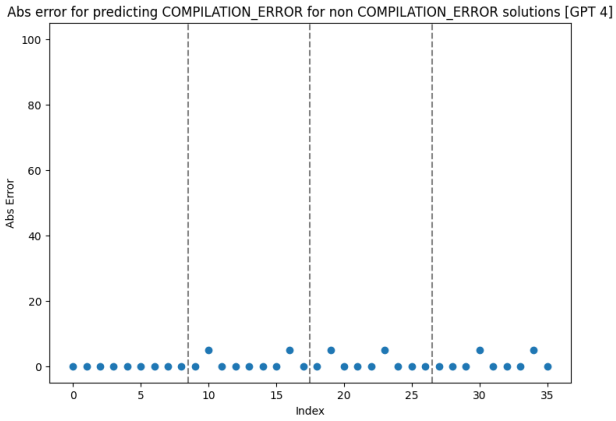


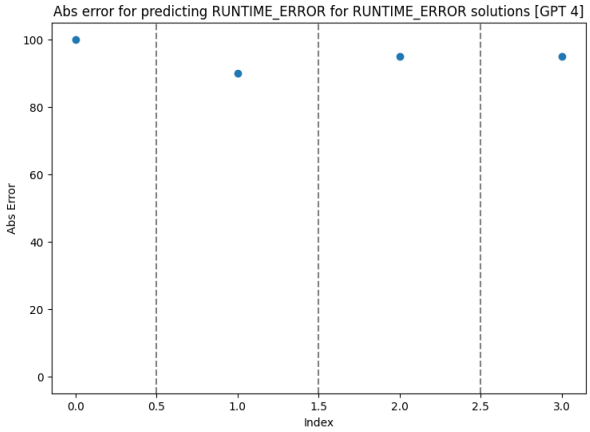


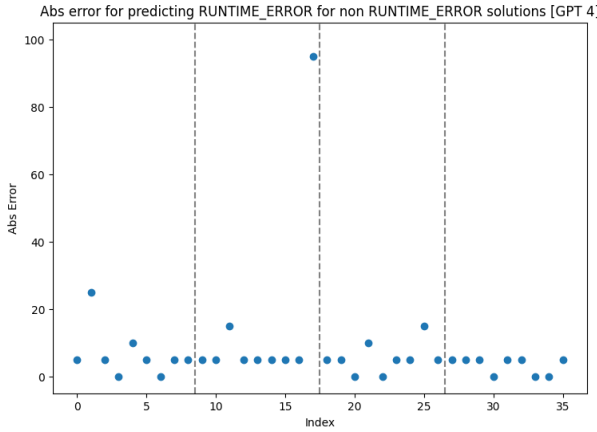


- **GPT-4**: Contrarily, GPT-4 demonstrated a tendency to incorrectly assume code correctness less frequently. However, it struggled in identifying faulty codes, indicating a weakness in error prediction.



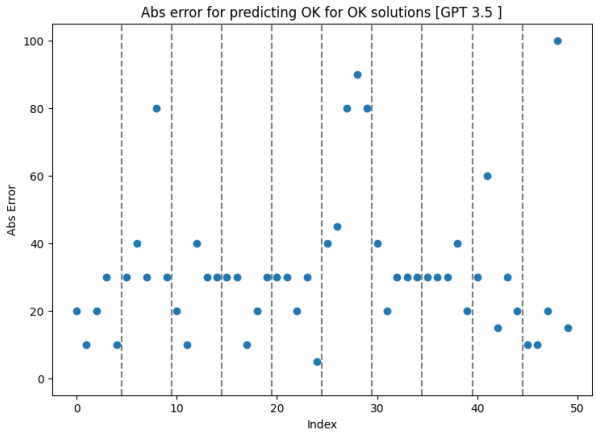


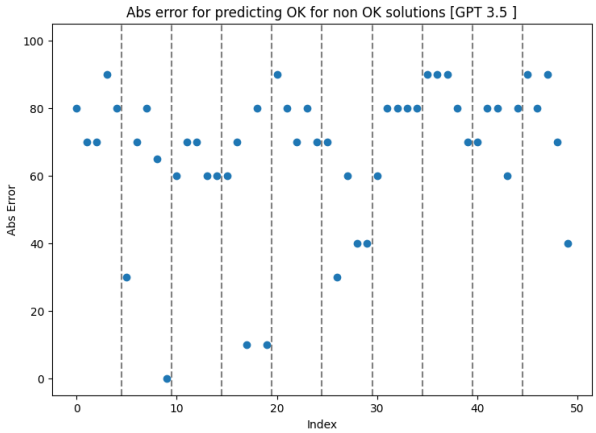


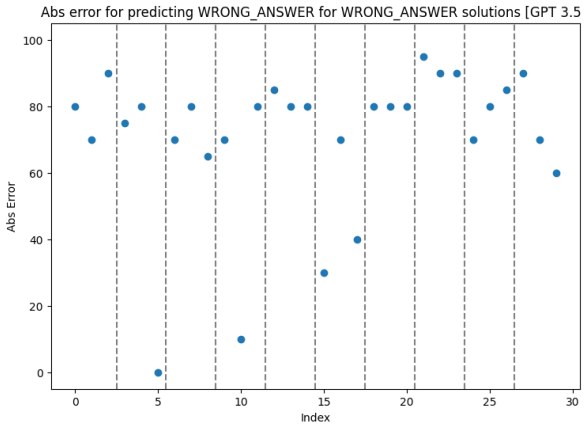


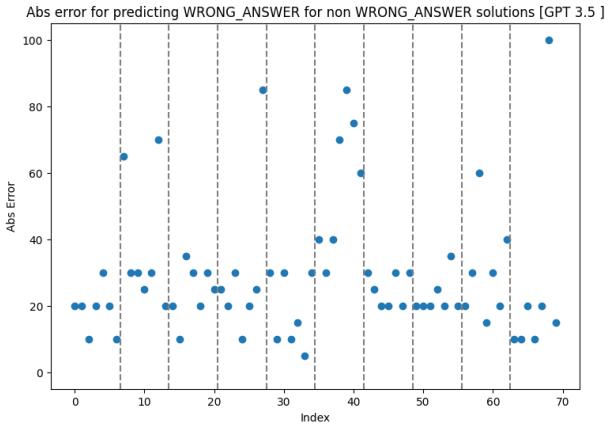
Solution Quality Assessment

- **GPT-3.5**: The variance in error prediction was significant, and overall, GPT-3.5 showed a limited capability in differentiating between correct and incorrect solutions.

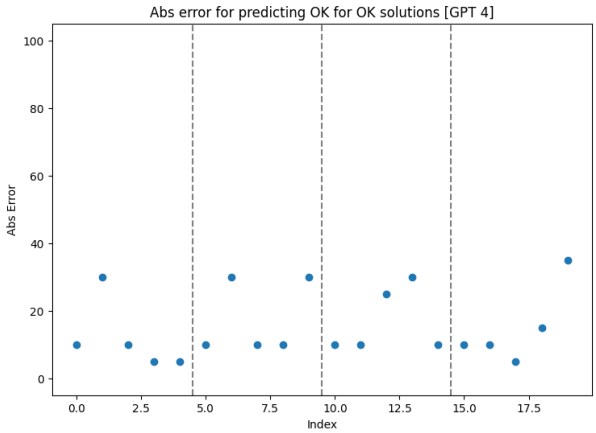


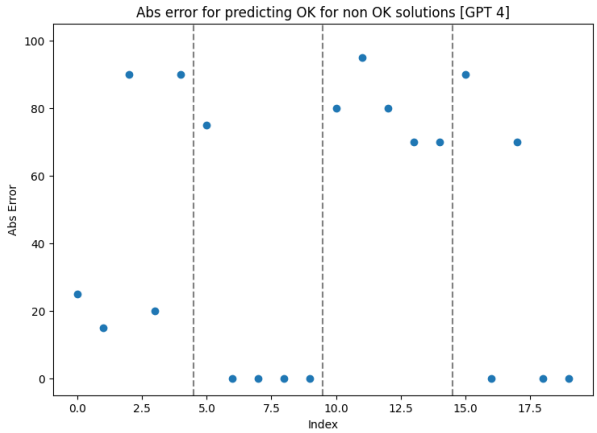


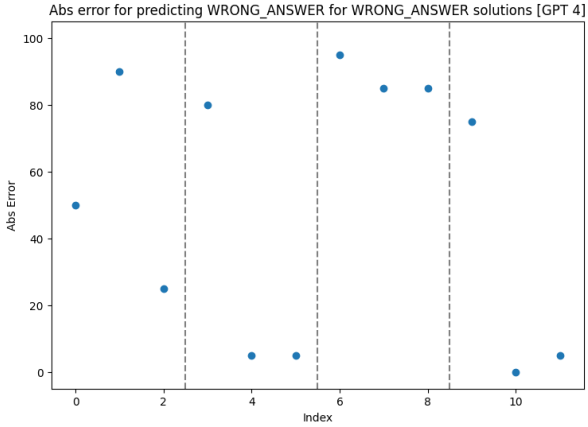


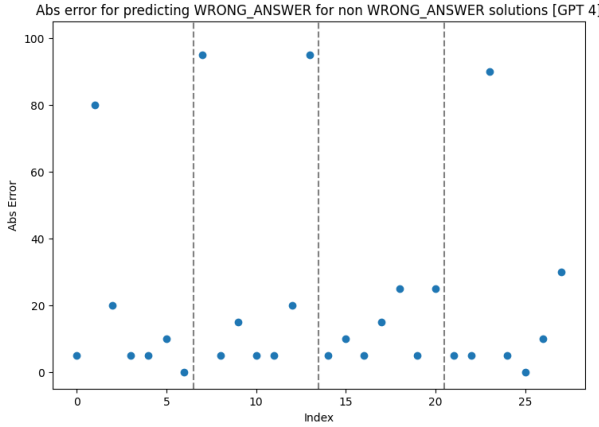


- **GPT-4**: There was a noticeable improvement in GPT-4's performance. It accurately differentiated between correct and incorrect solutions for certain problems. Nevertheless, notable exceptions were observed where it confidently and incorrectly assessed the quality of solutions.



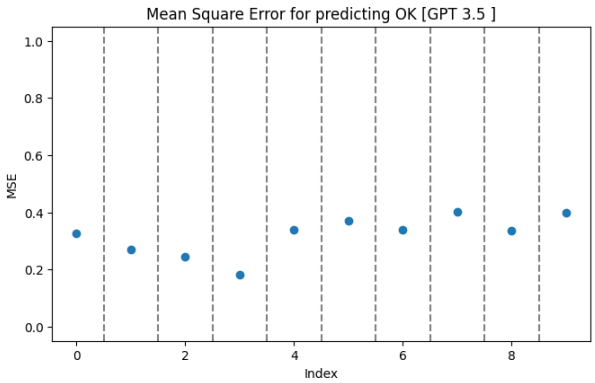


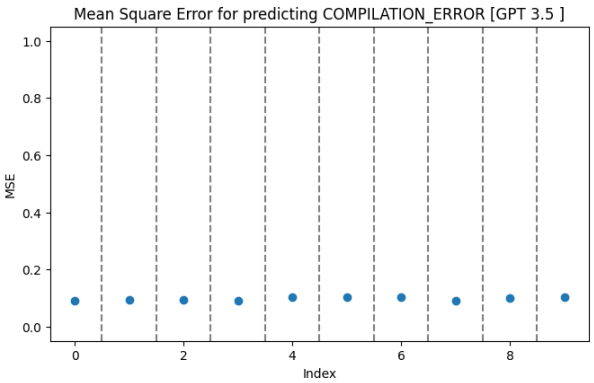


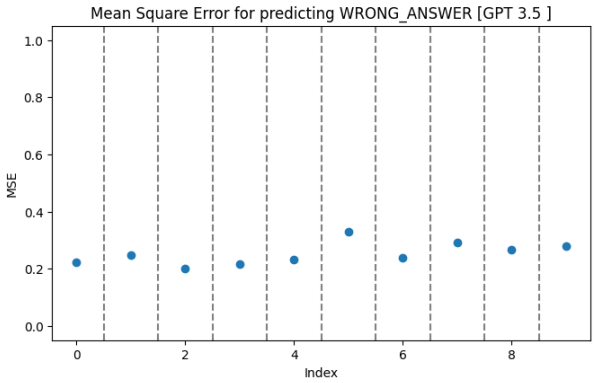


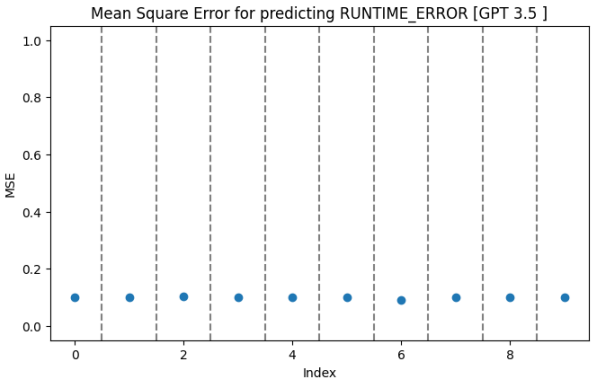
Mean Square Error (MSE) and Average Error Comparison

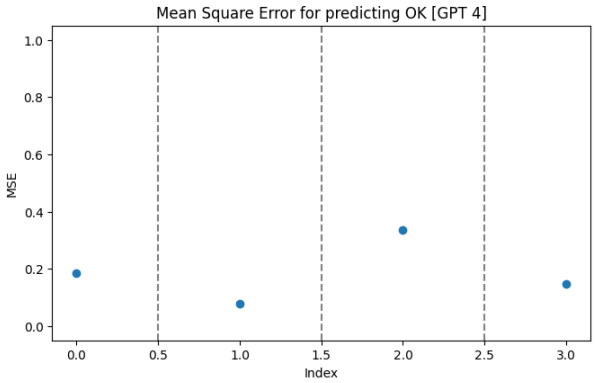
- **MSE**: Despite GPT-4's improved performance in some areas, the presence of outliers resulted in similar MSE for both models.

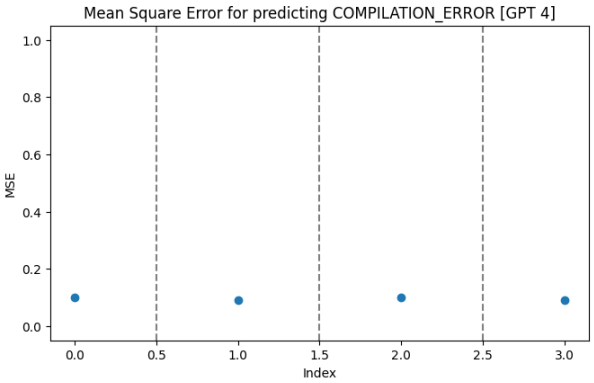


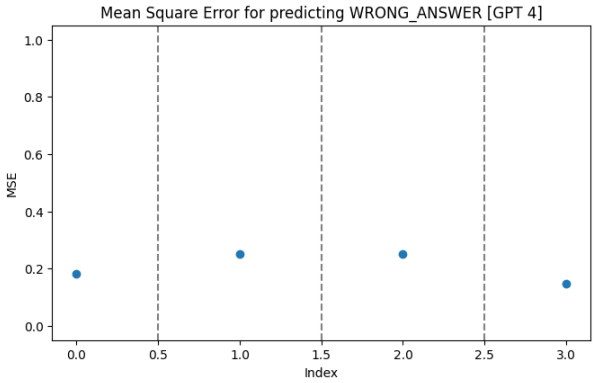


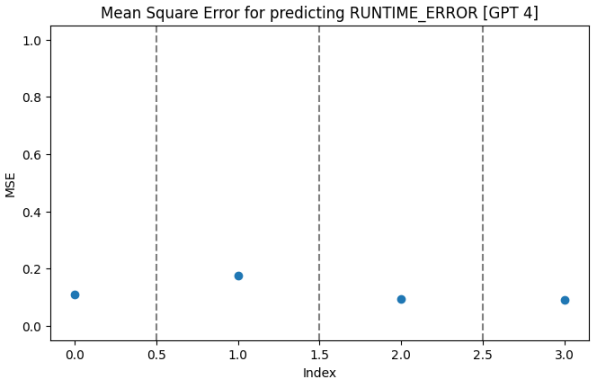






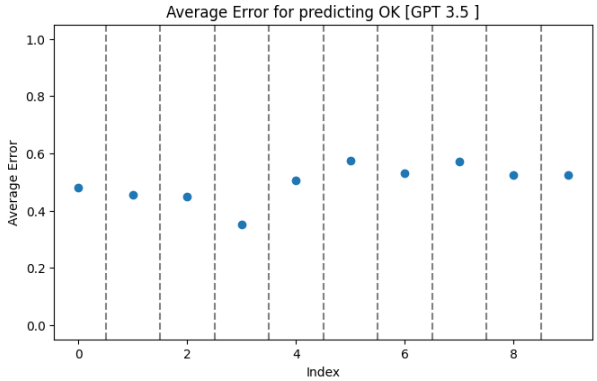


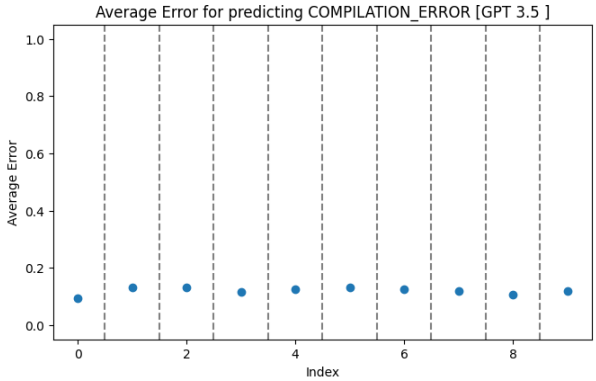


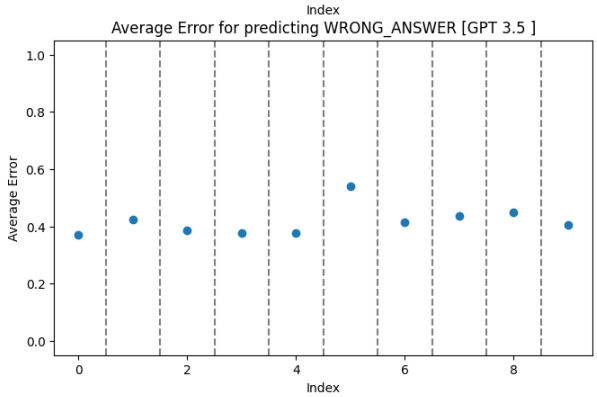


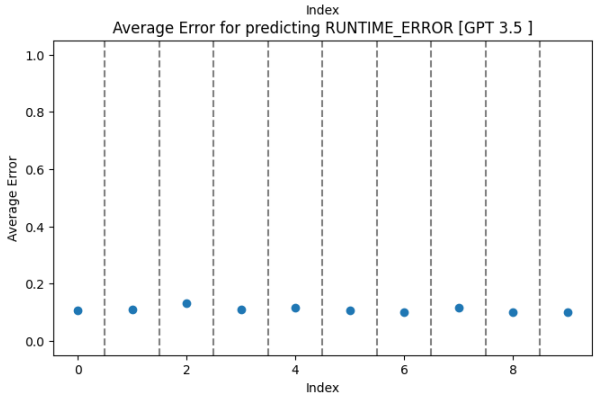
- **Average Error**: In terms of average errors, GPT-4 showed better performance in distinguishing between good and bad solutions but was less effective in

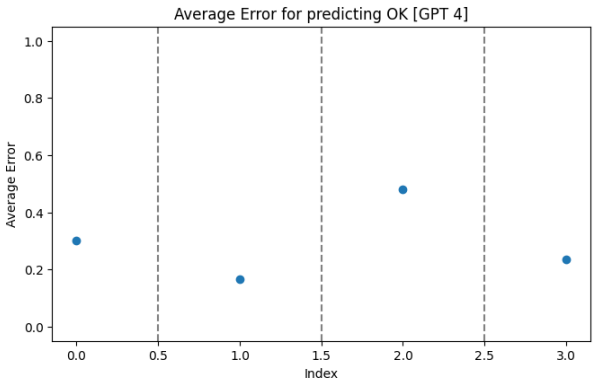
identifying compile and runtime errors.

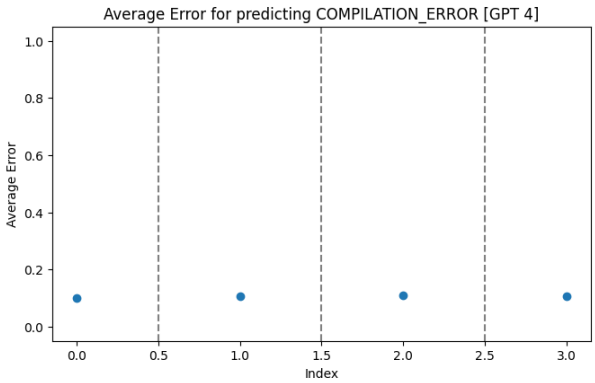


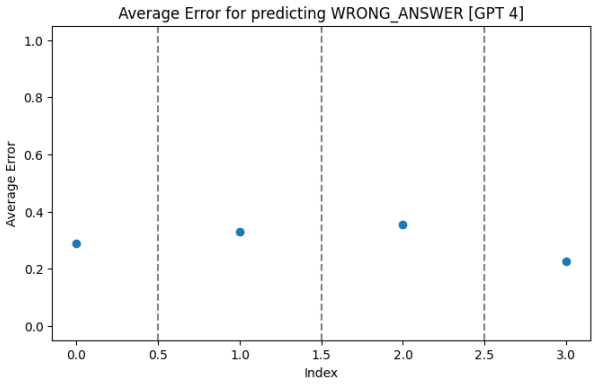


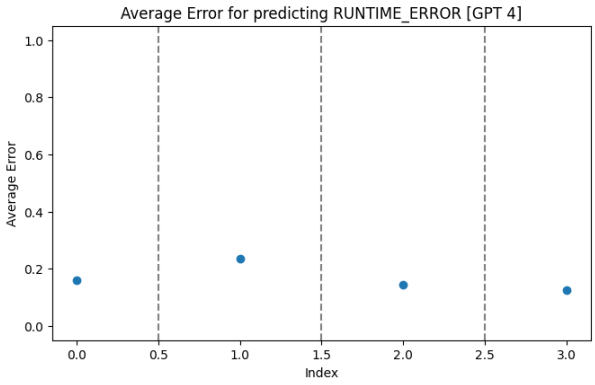


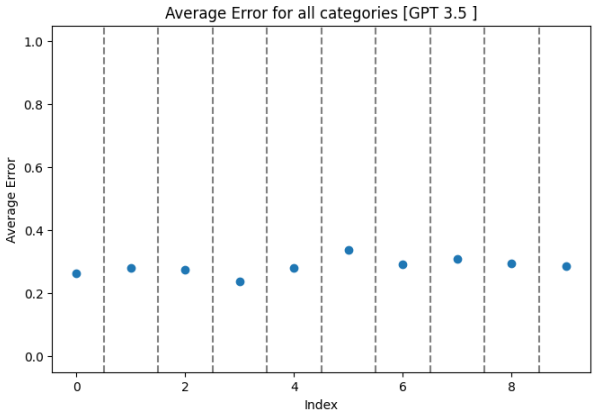


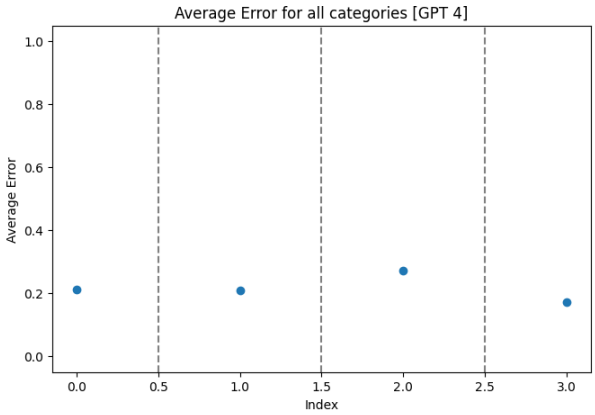






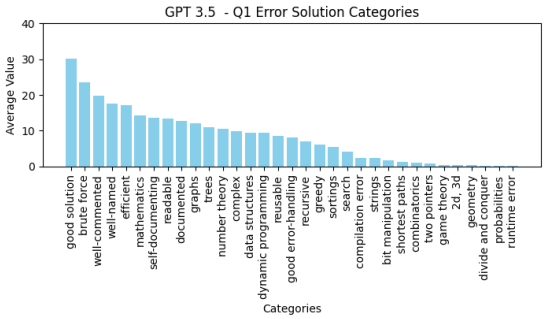


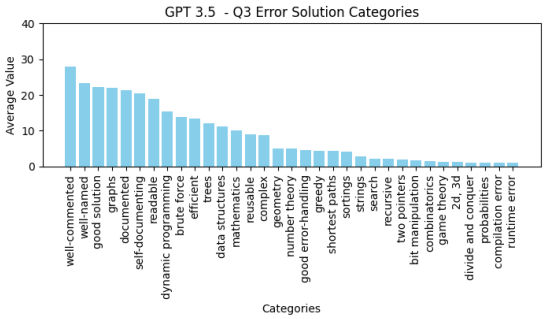


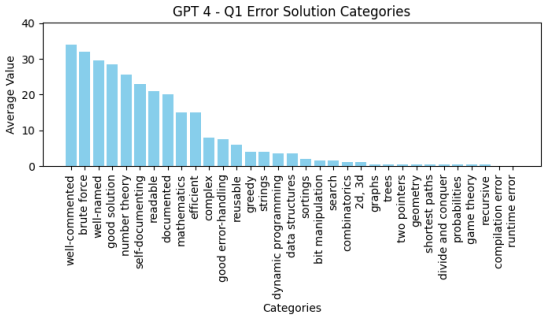


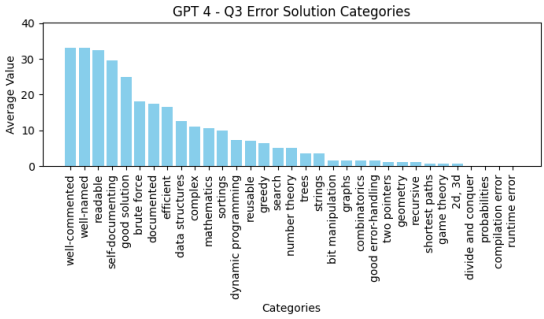
Error Analysis by Solution and Problem Categories

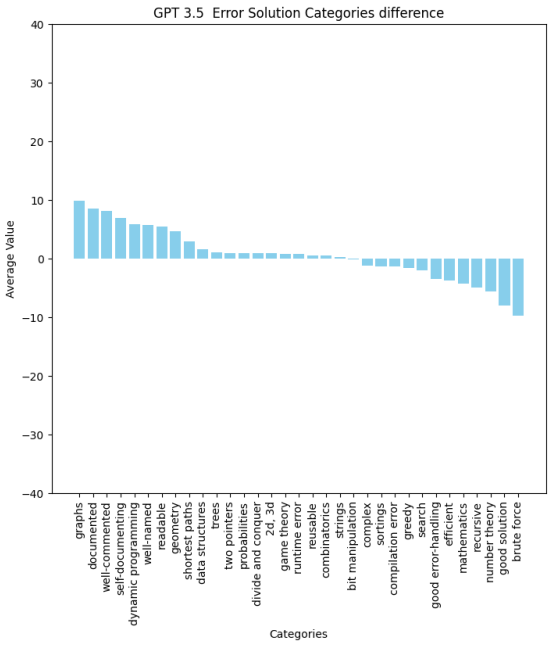
- Solution Category Analysis:
  - For **GPT-3.5**, well-documented and commented codes often led to incorrect assessments, indicating a false sense of accuracy based on code presentation.
  - In contrast, **GPT-4** showed better predictive power with well-commented and structured codes, suggesting improved performance in evaluating code quality.

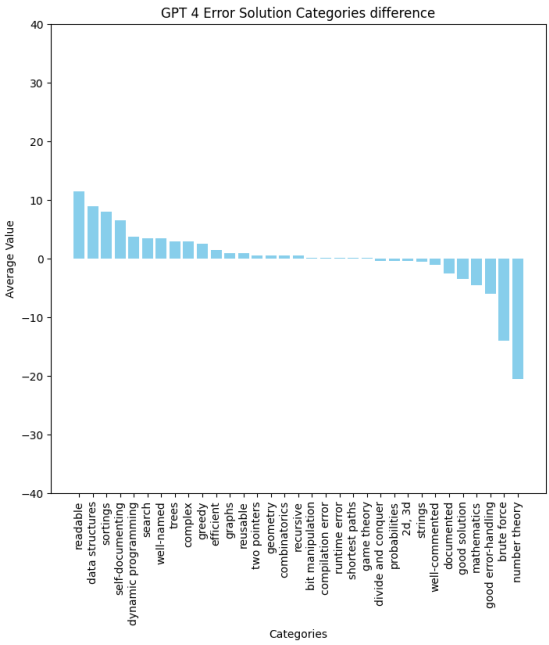






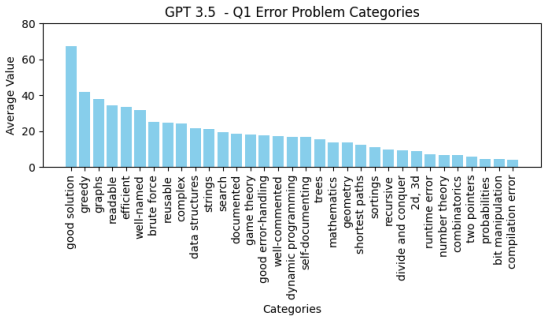


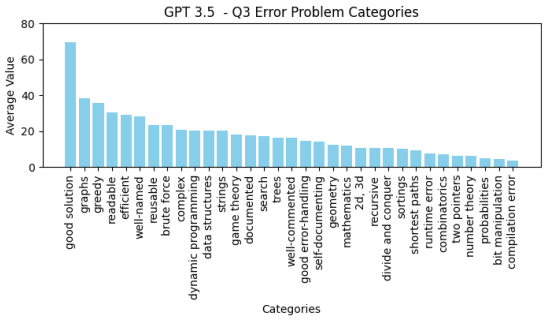


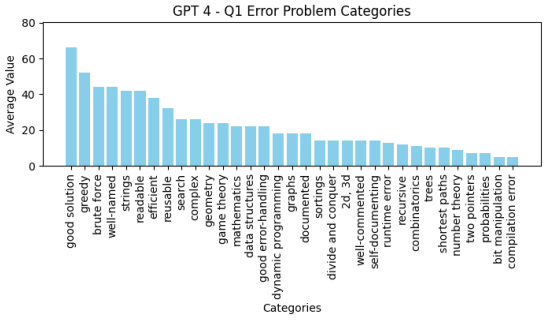


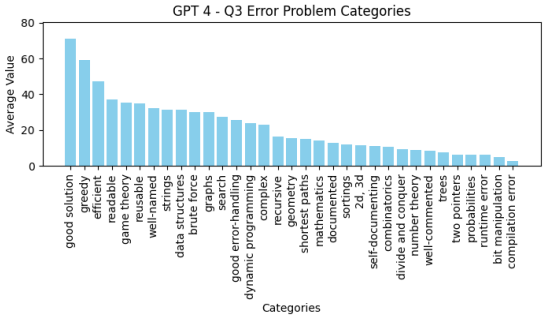
- Problem Category Analysis:
  - For **GPT-3.5**, the type of problem did not significantly impact the model's predictive accuracy, implying a general limitation in understanding and evaluating problems.
  - **GPT-4** exhibited more variance in performance across different problem types. It showed proficiency in categories like brute force, strings,

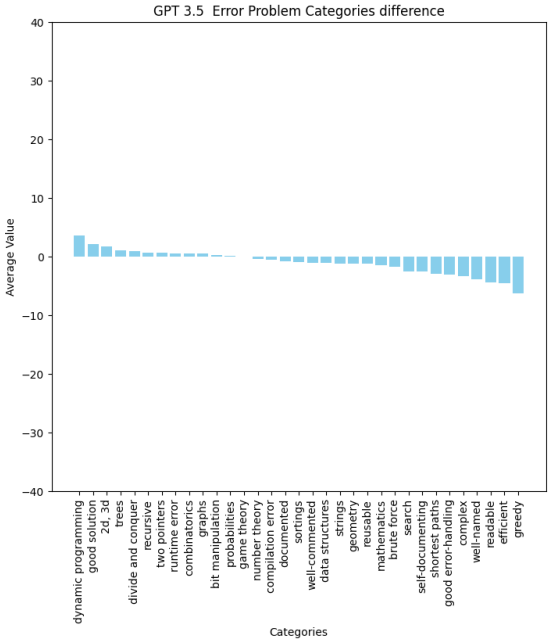
geometry, and general mathematics but underperformed in areas like graphs, data structures, dynamic programming, and recursive algorithms.

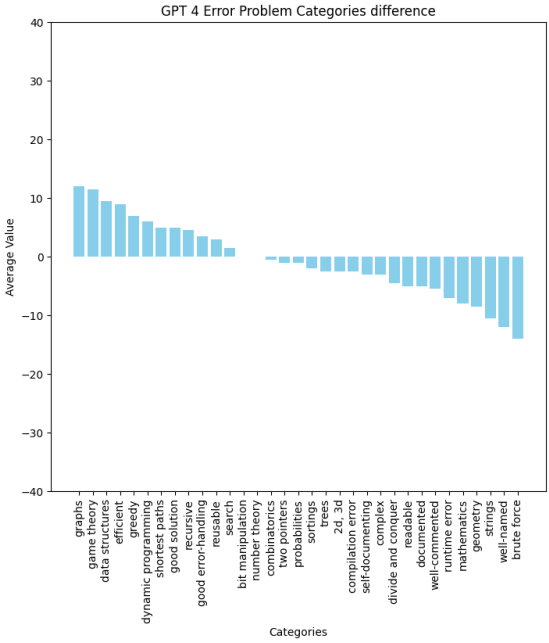












Analysis conclusion

The findings suggest that while GPT-4 shows improvements in certain aspects of problem evaluation over GPT-3.5, both models exhibit limitations in accurately

assessing programming solutions, particularly in error prediction and certain problem categories.

1.5.6 Discussion

This research aimed to evaluate the ability of advanced language models, specifically ChatGPT-3.5 Turbo and GPT-4 Turbo, in predicting defects in Competitive Programming (CP) problem solutions. The results highlight both the potential and limitations of these models in a CP context.

Interpretation of Findings

The findings indicate a marked difference in the performance of GPT-3.5 and GPT-4. While GPT-4 demonstrated an improved ability to differentiate between correct and incorrect solutions in most instances, it, along with GPT-3.5, showed a notable deficiency in predicting and evaluating correctness for solutions with runtime and compilation errors. This suggests that while these models have advanced in certain aspects of problem evaluation, their understanding and analysis of programming code, particularly in detecting technical errors, remain limited. The research revealed that GPT-4, while generally more advanced, does not uniformly outperform GPT-3.5 in all aspects of competitive programming problem analysis. Notably, GPT-4 exhibited weaker performance in predicting runtime and compilation errors compared to GPT-3.5.

Limitations of the Study

A significant limitation of this study was the restricted number of queries made to ChatGPT due to cost constraints. This limitation curtailed the ability to conduct a more granular analysis of the models' performance across varying problem complexities and types. For instance, it would have been insightful to examine how the models' defect prediction abilities varied between simpler and more complex CP problems. Additionally, a deeper investigation into specific problem types where GPT-4 underperforms could have provided more nuanced insights into the models' evaluative capabilities.

Implications and Future Research

The study's findings provide insights in understanding the current capabilities and boundaries of AI in the realm of competitive programming. They suggest that while AI can assist in evaluating programming solutions, its effectiveness is varying based on the nature of the problem, as well as the clarity of the code. The research might suggest

that currently CP problem solutions are more likely to be memorized by the LLM rather than understood at a deeper level.

Future research should aim to overcome the limitations encountered in this study. Conducting a more extensive analysis with a larger dataset and across a broader spectrum of problem types and more queries could provide a more comprehensive understanding of the AI models' capabilities. Moreover, future studies could focus on refining AI models to improve their accuracy in detecting technical errors.

Conclusion

In conclusion, this research contributes to the understanding of AI's capabilities and limitations in competitive programming. This study suggests new avenues for further improving AI's reliability and effectiveness in programming. Unfortunately cost limitations prevented this study from giving a more complete analysis and insights on the observed performance differences between the GPT models and between the categories, but this leaves adequate room for future research.