Team 16
Software Engineering
Final Report

Keerthana Beligini
Computer Science
Florida State University
Tallahassee Florida US
kb23u@fsu.edu

Likhita Ganipineni
Computer Science
Florida State University
Tallahassee Florida US
lg23h@fsu.edu

Roopa Chowdary Cherukuri
Computer Science
Florida State University
Tallahassee Florida US
rc23m@fsu.edu

1. INTRODUCTION:

In the fast-changing domain of natural language processing and AI, evaluating the quality and applicability of code produced by language models is increasingly important. While ChatGPT boasts several advantages, it is not without its share of challenges. addressing these concerns is instrumental in ensuring that the code produced can seamlessly integrate into real-time software projects.

The motivation to tackle various issues is underpinned by the numerous benefits that this project can yield. For instance, the quality of AI-generated code, exemplified by ChatGPT, profoundly influences the success and reliability of software applications. Therefore, validating and enhancing the quality of responses remains a paramount concern.

This study focuses on three distinct but related research questions that illuminate various aspects of ChatGPT's code generation process.

1) What types of quality issues (for example, as identified by linters) are common in the code generated by ChatGPT?
2) Can we forecast whether a developer will incorporate the code in a production environment by analyzing their interactions with ChatGPT during a conversation?
3) How do the answers regenerated by ChatGPT differ from the ones generated initially?

Our Research scrutinizes the quality of ChatGPT's code, using tools like code linters to spot typical issues, adding to the discussion about the capabilities of AI in code generation. Furthermore, it looks into whether the way developers interact with ChatGPT can forecast the code's likelihood of being integrated into live systems, shedding light on the real-world utility of AI-created code. The study also examines how the responses generated by ChatGPT evolve over a conversation, uncovering potential enhancements or inconsistencies in later responses and emphasizing the changing nature of AI-based outputs.

2. METHODOLOGY:

2.1 Research Question 1 Methodology

2.1.1 Defining Quality Metrics:

This step involves determining the specific criteria or standards that will be used to assess the quality of the generated code. Quality metrics encompass variable naming, syntax errors, code formatting, complexity, and best practices. In our case, we used Pylint Configuration (pylintrc). By configuring the "pylintrc" file to align with our project's coding standards and quality metrics, we can ensure that Pylint enforces consistent coding practices, leading to improved code quality and maintainability. These metrics include settings for indentation, variable naming, error and warning messages, code complexity, import organization, and more. They are crucial for ensuring that code adheres to coding standards, readability guidelines, and quality requirements specific to our project.

2.1.2 Selection of linter:

This step involves choosing a code analysis tool or linter that is suitable for the dataset. The code makes the choice of Pylint as the selected linter for Python-generated code. This is a crucial decision to ensure that the linter is compatible with the code we intend to analyze.

2.1.3 Automated Analysis:

Automated analysis involves using code analysis tools

linters to examine the generated code systematically. Pylint is the chosen linter for Python-generated code. The goal is to analyze the code files using Pylint to identify potential issues. It uses a custom message template to format the linting results, providing detailed information about each issue found. We iterate through each file in the specified dataset folder, run the linter on the code within those files, and store the results in the list.

2.1.4 Finding the number of issues:

The purpose of this is to keep track of the number of issues found in analyzed code file. This information can be valuable for further analysis and reporting. It allows for a quantitative assessment of code quality, which can be useful in managing and improving codebases.

2.1.5 Manual review:

Manual review is an essential part of the methodology to identify nuanced issues that automated tools like linters may miss. We set a threshold to determine when a manual review is warranted. If the number of issues identified by the linter exceeds this threshold for a specific file, it is flagged for manual review.

2.1.6 Linter rating:

Pylint will analyze the python script and provide a report with information about any issues it finds. This report typically includes details such as code quality scores, style recommendations, and specific lines of code where issues are detected. It helps developers identify and address potential problems in their Python code. The code rating is also given out of 10.

2.2 Research Question 2 Methodology

2.2.1 Data Collection:

Collected a dataset containing conversations between developers and ChatGPT. Each conversation includes prompts and corresponding answers.

2.2.2 Sentiment Analysis:

Computed compound sentiment scores by conducting sentiment analysis on the responses using VADER. The apply function is used to apply the VADER sentiment analysis to every response in the DataFrame. The sid.polarity_scores(str(x))['compound'] function uses VADER to determine the compound sentiment score for each answer.

The text's overall sentiment polarity is represented by the compound score, which goes from -1 (very negative) to 1 (most positive). The generated compound sentiment scores are kept in the Data Frame's 'Sentiment' column, a new column.

2.2.3 Feature Extraction:

Separated the prompts and answers using TF-IDF vectorization, which transformed text into numerical features. Extra features like user experience, conversation length, and sentiment scores were extracted.

2.2.4 Labeling:

To generate binary labels for the initial classification task, use sentiment scores.

Use a cutoff point (0.1, for example) to divide sentiment scores into positive (1) and negative (0) categories.

2.2.5 Feature Matrix:

A comprehensive feature matrix was created by combining TF-IDF vectors with other features. Training and testing sets are created from the binary labels (original_labels) and feature matrix.

2.2.6 Dummy Classifier:

Incorporated a Dummy Classifier (uniform strategy) for baseline performance comparison. Evaluated and reported accuracy for the Dummy Classifier.

2.2.7 Classification Model:

For binary classification, we used a Random Forest classifier as the machine learning model. This model is particularly good at supporting non-linear relationships, making feature importance analysis easier, and producing reliable predictions. It makes predictions about whether developers will integrate code into a production environment by analyzing underlying patterns and correlations in the data.

2.2.8 Model Training:

Take on the initial classification task using a RandomForestClassifier. Utilizing the feature matrix and binary labels derived from sentiment analysis, train the model.

2.2.9 Performance Evaluation:

Evaluated the model using a test set.

Calculated accuracy, precision, recall, F1-score, and ROC AUC to assess performance. Analyzed confusion matrix to understand model behavior.

2.2.10 Model Interpretation:

When accuracy is greater than a predetermined cutoff point (e.g., 0.7), the model is deemed predictive of code incorporation.

2.2.11 Visualizations and Analysis:

Visualized sentiment distribution, conversation length, and other relevant factors. Explored correlations between features using correlation matrices. ROC Curve and AUC: Plotted the ROC curve and calculated the AUC to assess the model's overall performance.

2.3 Research Question 3 Methodology

2.3.1 Loading the Data

Importing Data from CSV: The process begins by importing the dataset from a CSV file into a Pandas DataFrame, named chatgpt_data. This dataset is crucial as it includes ChatGPT-generated responses and other key information necessary for the subsequent analysis.

2.3.2 Data Organization

Understanding the Dataset: An in-depth exploration of the dataset's structure is conducted. This involves pinpointing the specific columns that play a vital role in response generation, ensuring a solid foundation for the upcoming analytical steps.

2.3.3 Choosing Evaluation Metrics

Metric Selection: Key metrics such as accuracy and relevance are chosen to assess the quality of the generated responses. These criteria are essential to quantitatively and qualitatively measure the effectiveness of ChatGPT's output.

2.3.4 Analyzing Quantitatively

Similarity Measures: Metrics like BLEU, ROUGE, and F1 Score are employed to objectively evaluate how similar the ChatGPT responses are to expected outputs. This phase provides a quantitative measure of the response quality.

2.3.5 Analyzing Qualitatively

Involvement of Human Reviewers: To add a qualitative dimension, human reviewers meticulously analyze the responses. This step is crucial as it offers human perspectives and insights into the data, complementing the quantitative findings.

2.3.6 Analysis Comparison

Contrasting Different Analyses: We then compare the insights gained from both the quantitative and qualitative analyses. This comparison is aimed at uncovering the diverse aspects of ChatGPT's response quality under various conditions.

2.3.7 Statistical Confirmation

Employing Statistical Tests: To statistically validate our results, techniques like t-tests and Chi-Squared tests are utilized. These methods are crucial for determining the statistical significance of our observations.

2.3.8 Visualization of Data

Creating Visual Aids: Using tools such as Matplotlib and Seaborn, we craft visual aids like bar graphs to depict the quantitative differences in the responses. These visualizations help in making the data more comprehensible and engaging.

3. FINDINGS:

3.1 Research Question 1 Findings

Various kinds of quality issues identified include invalid -name, 'line-too-long', 'missing-final-newline', 'missing-module-docstring', 'pointless-statement', 'too-many-lines' , 'undefined-variable'. These can be seen in Fig 3.1.1. It can be observed that most issues are related to exceeding too many limits as defined by the quality metric and Und efined variables.

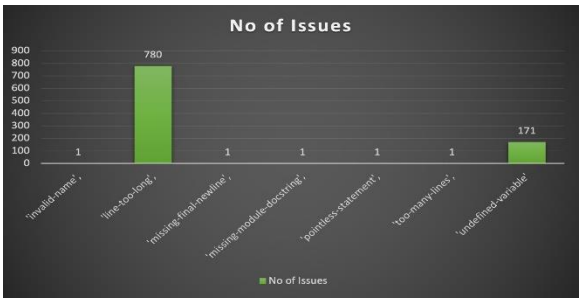

Fig 3.1.1

Finally, the linter rating has been given as 7.5/10 which can be observed in fig 3.1.2.

```
************* Module SE_1
SE_1.py:56:0: C0305: Trailing newlines (trailing-newlines)
SE_1.py:1:0: C0114: Missing module docstring (missing-module-docstring)
SE_1.py:1:0: C0103: Module name "SE_1" doesn't conform to snake_case naming style (invalid-name)
SE_1.py:35:0: C0116: Missing function or method docstring (missing-function-docstring)
SE_1.py:35:17: W0613: Unused argument 'url' (unused-argument)

-----------------------------------------------------------------
Your code has been rated at 7.50/10 (previous run: 7.50/10, +0.00)
```

Fig 3.1.2

3.2 Research question 2 Findings

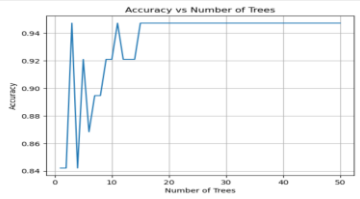

Fig 3.2.1

The fig 3.2.1 illustrates how the performance of the Ran dom Forest model evolves with an increasing number of trees. It helps identify the optimal balance between mode l complexity and accuracy, offering insights into the imp act of ensemble size on predictive capabilities.

```
Evaluation Metrics:
Accuracy: 0.9473684210526315
Precision: 0.9375
Recall: 1.0
F1-Score: 0.967741935483871
ROC AUC: 0.875
```

Fig 3.2.2

The fig 3.2.2 shows that the model achieves an impressi ve accuracy of 94.74%, with a high precision of 93.75% and perfect recall of 100%. The F1-score of 96.77% indi cates a well-balanced precision-recall trade-off. The RO C AUC score of 87.5% confirms the model's effective di scrimination between positive and negative instances. O

verall, the metrics showcase a strong and reliable classifi cation performance.

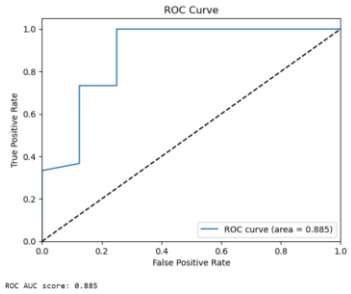

Fig 3.2.3

The fig 3.2.3 ROC curve visually captures the model's di scrimination ability, showing a clear upward trajectory t owards the upper-left corner. With an AUC score of 87.5 %, the curve reflects the model's effectiveness in disting uishing between positive and negative instances across v arying thresholds. This ROC analysis affirms the model's robust performance in binary classification, emphasizing its reliable predictive capabilities.

3.3 Research Question 3 Findings

The figures visually analyze the dataset using different attributes at two instances. The figure 3.3.1 examines 'Mentioned Sources' and 'Mentioned Authors' to find prevalent sources and frequently cited authors. The figure 3.3.2 focuses on 'Status Codes' and 'URL Domains' to reveal HTTP status code distribution and common domains related to the dataset. These figure-based perspectives highlight the dataset's multifaceted nature, uncovering various dimensions through attribute analysis.

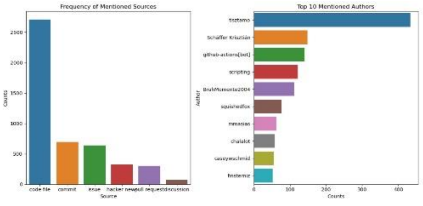

Fig 3.3.1

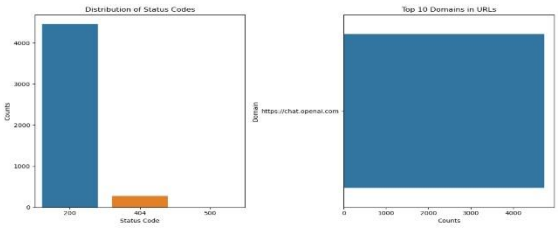

Fig 3.3.2

4. CONCLUSION

The study revealed various quality issues in ChatGPT's code generation, including syntax errors and adherence to coding standards, with linters pinpointing areas for improvement. It also identified factors, such as conversation sentiment and code complexity, influencing developers' decisions to use ChatGPT's code in production. Additionally, the research suggested that the quality of ChatGPT's responses varies between initial and regenerated answers, with analyses indicating differences in accuracy and relevance. These insights are key to understanding how iterative interactions with ChatGPT affect response quality.

5. Contributions

Contributor 1(Roopa Chowdary Cherukuri- RC23M): The contributor worked on Research question 1. She defined quality metrics for code evaluation, selected and configured linters for automated analysis of ChatGPT-generated code. She complemented this with a manual review to identify issues beyond the linters' scope, resulting in a comprehensive report on the code's quality. Finally she analysied the quality issues and obtained the linter rating.

Contributor 2(Keerthana Beligini- KB23U):The contributor worked on Research question 2. Her role involved data preparation, feature engineering using sentiment analysis and TF-IDF, and model training with a Random Forest Classifier to predict ChatGPT code usability in production. They also tuned model parameters and conducted thorough performance evaluations using metrics like accuracy, precision, recall, F1 score, and ROC AUC, culminating in a comprehensive assessment of the model's predictive capabilities.

Contributor 3(Likhita Ganipineni- LG23H): The contributor worked on Research question 3. Her role involved a detailed analysis of ChatGPT responses. She loaded and prepared the data, selected and applied quantitative metrics like BLEU, ROUGE, and F1 Score for text analysis, and collaborated with human annotators for qualitative assessment. Then she compared these results, conducted statistical validation using t-tests or Chi-Squared tests, and created visualizations using tools like Matplotlib and Seaborn to illustrate their findings.

6. REFERENCES

1. Hamfelt, P. (2023). MLpylint: Automating the Identification of Machine Learning-Specific Code Smells.
2. Fowler, M. (2018). Refactoring. Addison-Wesley Professional.
3. Pyle, D. (1999). Data preparation for data mining. morgan kaufmann.
4. Tibshirani, H. R., James, G., & Trevor, D. W. (2017). An introduction to statistical learning. springer publication.
5. Bird, S., Klein, E., & Loper, E. (2009). Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.".
6. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in science & engineering, 9(03), 90-95.
7. Chin-Yew, L. (2004). Rouge: A package for automatic evaluation of summaries. In Proceedings of the Workshop on Text Summarization Branches Out, 2004.