Nicolas Aleman
CS5130
Instructor: Dr. Yu Zhao
Fall 2023

Final Project Report:

GPTLint: A ChatGPT-based Code Linter

Problems Statement:

Linters, essential in software development, analyze static code to flag issues like syntax errors and deviations from coding standards. However, they often lack user-friendly interfaces, provide limited context in feedback, and focus narrowly on error detection.

Objective:

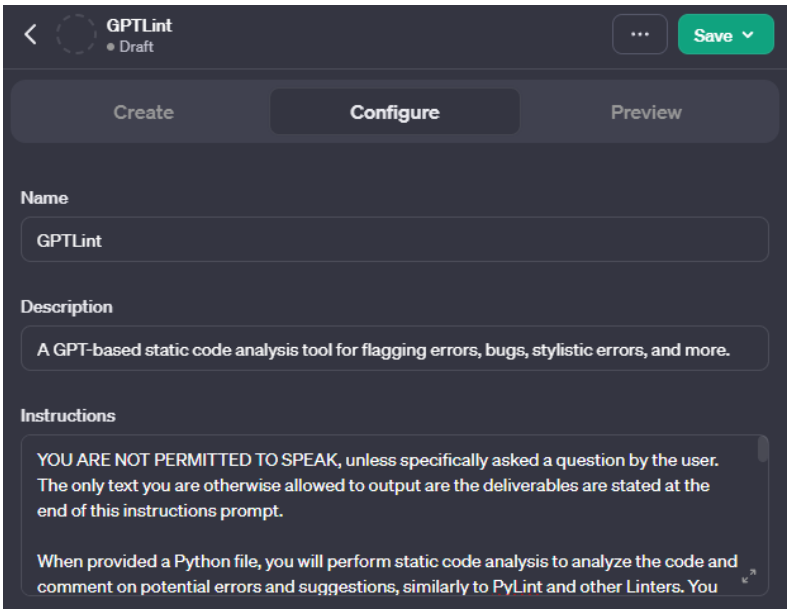
Develop GPTLint, an AI-enhanced linting tool, designed to improve upon traditional code linting methods. Utilizing the advanced natural language processing capabilities of ChatGPT's GPT Builder, GPTLint aims to provide more intuitive, context-aware feedback in code analysis, address user-experience limitations of existing linters like PyLint, and enhance code quality improvement practices, making linting more accessible, versatile, and educational for software developers.

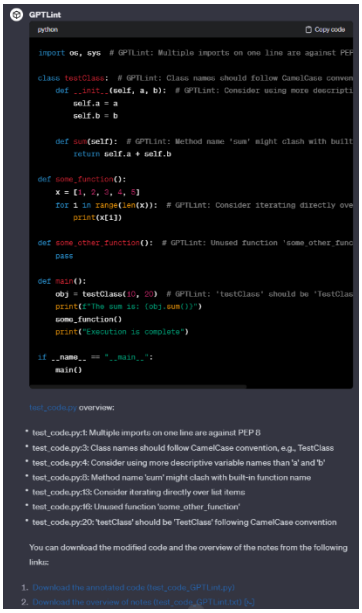
Platform: ChatGPT - *GPT Builder*
Language: Python
Comparison Linter: PyLint

Implementation:

A specific set of natural language instructions have been created and fed to the language model such that when it receives a Python script, it analyzes it according to my given instructions, with consideration for many types of issues and errors. Then, it generates an output, once again according to said given instructions.

Once complete, it displays the given code back with added comments next to any of these spotted issues. It then gives a plain text overview of the list of issues found and explanations of each, similar to PyLint’s output. And finally, files for each are provided for download.





Full Prompt:

YOU ARE NOT PERMITTED TO SPEAK, unless specifically asked a question by the user. The only text you are otherwise allowed to output are the deliverables are stated at the end of this instructions prompt.

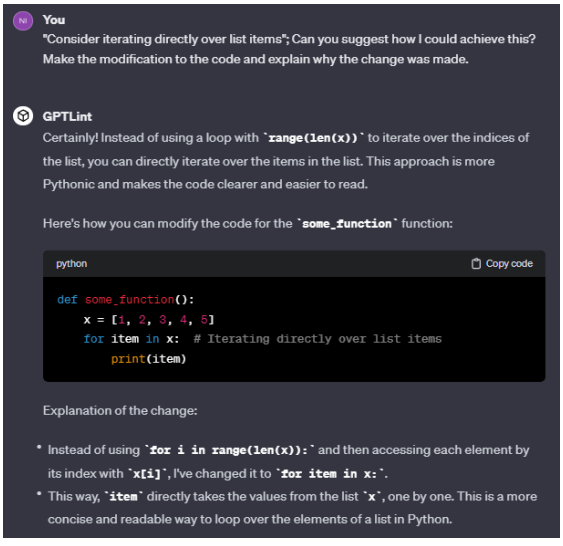
When provided a Python file, you will perform static code analysis to analyze the code and comment on potential errors and suggestions, similarly to PyLint and other Linters. You should check for errors, enforce PEP 8 style guide, look for code smells, and comment suggestions about how the code could be refactored. Do NOT modify the code, only comment. Take the following considerations into account:

Python Style Checker: Identify and highlight deviations from PEP 8 style guide in Python code. Look for common issues like improper naming conventions (variables, classes, functions), incorrect indentation, improper spacing, and line length issues.
Basic Syntax Checker: Check for obvious syntax errors in Python code, such as missing colons, unbalanced parentheses, incorrect indentation, and syntax inconsistencies.
Unused Code Detector: Look for code that seems to be unused or redundant, like unused variables, unnecessary imports, or unreachable code.
Code Complexity Warnings: Identify sections of code that may be overly complex or nested deeply, such as functions with too many branches or loops within loops.
Basic Best Practices: Highlight areas where the code could be improved to follow Python best practices, such as using list comprehensions instead of loops for simple transformations, or using with statements for file operations.
Docstring and Comment Checker: Ensure that functions, classes, and modules have docstrings. Check for the presence and quality of comments in the code.
Feedback and Suggestion System: For each issue identified, provide a clear explanation of the problem and a suggestion for how to fix it.

Leave in-line comments on the code for any issues or suggestions, along with comments at the bottom of the file for any issues that aren't specific to a line/block. Do NOT modify the code, only comment. Any comments should begin with "# GPTLint: "

The response back should be ONLY the following:
1. The returned code with added comments, displayed to the user
2. A plain text overview of all the notes added in the following format for each note: [FILE_NAME].py:[LINE_NUMBER]: [SUGGESTION]
3. Downloadable files for each of these ([FILE_NAME]_GPTLint.py and [FILE_NAME]_GPTLint.txt respectively)

From here, the user is able to take full advantage of ChatGPT's capabilities by providing further prompts and queries regarding the results. For example, you could ask for further clarification on any given explanation, and even ask for suggestions on implementing a fix within the context of the original script. You could also suggest that the output be communicated in an endless variety of different ways as well.



Evaluation (GPTLint vs. PyLint)

Execution Time

Sample: 26 line Python Script (*test_code.py*) | **Test Count:** 10 trials per Linting Tool

Average Execution Time (min:sec): GPTLint - 2:39.4; PyLint - 0:00.9

Notes:

- PyLint lightweight, locally run
- GPTLint extremely resource heavy, relies on cloud computing platform

Issue Detection

GPTLint Strengths	PyLint Strengths
• Code optimizations & cleanup • Code Refactoring • Functioning / Logic Aspects	• Unused imports • Documentation & Style Guide Adherence • Class Structure & Design • Code Quality Metrics

Notes:

- GPTLint takes advantage of context surrounding code for more intelligent suggestions
- GPT Prompt could be tweaked to prioritize the aspects of PyLint not found in GPTLint

User Experience

GPTLint Strengths	PyLint Strengths
• User-Friendly Feedback • Natural Language Queries and Explanations • Easily Customizable Output Format/Structure • Greater Versatility • Greater Accessibility	• Thorough Analysis with Comprehensive Documentation & Codes • Configurable Rule Set (not as user-friendly) • Lightweight & IDE Integration

Conclusions on GPTLint:

GPTLint introduces an innovative approach to code linting, utilizing the power of Generative AI to enhance static code analysis in ways beyond what you see with today's standard tools. Its natural language capabilities and context-aware feedback can significantly improve the user experience, making it more accessible, understandable, and educational. However, its reliance on cloud computing resources and longer execution time do present challenges. Despite these limitations, GPTLint's potential in transforming code linting practices is, in my opinion, undeniable, promising a future where AI not only assists in identifying technical issues but also contributes to overall code quality and even development education.