Code improvement report

Roblox prime numbers:
Oleh Basystyi
Anna Stasyshyn
Artur Rudish
Anton Valihurskyi
Maksym Zhuk

April 2024

1 Introduction

In this little research, our team researched AI's capability to optimize Python code in different respects: *style*, *memory usage*, *runtime*. In this part, we used only ***ChatGPT-3.5*** and ***Copilot***. Here, we will provide only aggregate conclusions among different types of optimization techniques. Full list of used tasks with some statistical data as *lines of code before and after the optimization, runtime*, and a detailed summary of each task can be found in table on second sheet. Also, we have the GitHub with where you can find original and optimized versions of problems.

2 ChatGPT-3.5 vs Copilot

ChatGPT-3.5 vs Copilot At the beginning of this part, our team decided to use these two AIs. And, on average, we can see that ChatGPT-3.5 fits better for code optimization than Copilot. Especially, you can see that in the first seven rows in the table. So it is better to avoid unnecessary struggles and use Copilot only for code generation.

3 Style optimization

From our observations, we can say that ChatGPT is very good at improving code style and readability. It can analyze the codebase, split it into separate blocks, write code that will be far more readable than the original one. Additionally, Chat- GPT showed a great capability of rewriting code to fit the functional paradigm. Moreover, AI is good at following PEP8, with some exceptions, when it becomes ignorant of pylint messages. However, some issues occurred. When AI rewrites complex or illogically written code, it tends to forget some details, such as writing module docstring or totally forgets about the existence of a particular function. Moreover, if the precedence of operations is crucial in original code; it can rearrange them incorrectly. Despite the listed issues, we think that AI's work with style improvement was good enough, so they can be used to optimize the style of great volumes of legacy code, if it has enough test coverage to avoid the listed AI lags.

4 Runtime & memory optimization

As we have seen in our research, AI struggles with fully independent code optimization. It never gets sent the correct code on the first try. As mentioned many times before, AI just forgot about crucial details to make code more readable, and the output ends up being just totally not-working code, as in the

table #2, rows 14, 18. However, after several manual directions on what was done wrong, we can improve the performance, but it only works with problems with an uncomplicated description and simple code as in the table #2, row 18, but struggles to do the same thing in the task as shown in row 18.

5 Corner cases problem optimization

Furthermore, we have tested how AI can work with Pandas in the table #2, rows 8-9 and OOP, row 10. In conclusion, ChatGPT manages to understand the purpose of the functions and tries to improve them where possible. In both Pandas problems it improved the readability of long filters by dividing them into separate blocks but, as was said before, sometimes it changed the order of the operations and, as a result, broke the program. In addition, ChatGPT managed to find unnecessary parts of the code and eliminate them correctly, which improved code memory usage and run- time.

In OOP optimization, we noticed the following AI behaviors:

- It tends to select parts of the script (classes) where optimization is mostly needed and sends only the changed version of those parts - and this is very convenient. On the other hand, when AI does that, it usually forgets about the existence of other classes in the script.
- It tends to forget to include all methods in the optimized version and to inherit all parent classes.
- It does not totally understand OOP principles and can move methods from base to child classes.

Therefore, when you try to optimize OOP through AI, these issues should be taken into consideration. It's better to use the following prompts:

- Please optimize this code **{*code*}** this is the problem description **{*problem_statement*}** and the required test cases **{*test_cases*}**.
- Optimize code, do not remove distinct classes.
- Do not copy methods from parent classes to child classes.

6 Conclusion

To summarize, as of today, AI is not capable of optimizing code independently and correctly. It always forgets something important or is unresponsive to different prompts and sends the same solution with unchanged bugs. In spite of this, AI can make code far more readable, but it is not guaranteed that it will 100% retain its validity. However, generally AI can give the developer relevant and useful hints at what small changes can be made in order to make significant improvements. In order to use AI for code optimization, you should follow these Recommendations:

- In the beginning you should send all of **problem description, code, and test cases**.
- Try not to give general promts like *"Optimize this code"*, but specify what part of program must be optimized.
- If AI struggles to solve the bug after several prompts (sends the same code) you should provide **function name** or **class name** where the error occurred.
- Sometimes AI struggle to solve even the PEP8 errors from error signatures, so you should send prompt like *"Split the documentation into two lines"*, because AI can be irresponsible to prompt like *"10th line is to long, make it shoter"*.
- Do not rely on AI completely: you should only use it to search for optimization hints but not as complete tool for code improvement.