Towards Role Conversational AI in Systems Development: Experience Developing an Instructional OS

Joao Victor C. Kdouk, Meghavarnika Budati, Antonia Nunley, Tristan Rogers, Ketan Bhardwaj

1 Abstract

Conversational Artificial Intelligence has gotten increasingly popular among software engineers, and is posed to greatly reduce the complexity of software development. Conventional wisdom says that design of Operating Systems (or complex systems in general) is not considered amenable for the usage of Conversational AI, since it requires more than just writing code. Coming up with appropriate abstractions, choosing mechanisms based on underlying hardware architecture, and modularizing the overall design down to different subsystems are common examples of how complex problem solving takes place in the Operating System development scenario. In this paper, we set out to explore the strength of that conventional wisdom. Through the practical attempt to implement an entire operating system, this paper investigates the benefits and drawbacks of using AI in the design and development of Operating Systems. Our experience leads us to argue that, while an imperfect solution, Conversational AIs are already at a point where they can assist Operating System development with basic knowledge, such as x86 and ARM instruction documentation, and high-level design of systems.

2 Introduction

With the release of the Generative Pre-trained Transformer 3 (GPT-3) [1] and subsequent launch of ChatGPT [2] in November 2022, it became increasingly evident that such technology will permeate everyday life including assisting in performing daily tasks [3], and reducing the amount of work required to search for information [4]. While imperfect by many metrics [5], GPT-3 demonstrated for the first time that AI and humans can work as collaborators. For example, through GitHub Copilot [6], AI cemented itself as part of the daily lives of thousands of engineers around the world. Software engineers now have support not only in the process of writing code, but also in designing, debugging, testing, and deploying software.

While the process of writing software greatly improved in the past 40 years, learning about Operating Systems and implementing new features is still a gargantuan task for students at college-level and, depending on the feature at hand, for experienced software engineers alike. When engineers are presented with a hard task, the first step is to research the problem, check for existing solutions, understand implementation details, and devise a viable design. The sole process of researching can take days depending on the problem at hand, and designing a viable solution from a myopic view of the situation can result in security and performance issues [7]. Modern Operating Systems continue to suffer from vulnerabilities that result from the overlook of engineers [8]. We posit that those issues could be avoided if more holistic information was considered in the design process about a given situation or environment beforehand. In this paper, we outline our experience in the design and implementation of instructional Operating Systems, where Conversational AI specifically trained for the purpose of software engineering assists engineers in the development and design process, decreasing research time and improving the reliability of the final product.

3 Story

In the spirit of WACI and conversational AI, we wanted to share the story of how we ended up exploring the idea of using conversational AI for operating systems development as conversations.

SCENE – I

GT CS Department Lobby (2020)

FACULTY 2 Should the kids in elementary school get involved with coding?

FACULTY 1 But are you really sure, kids are gonna need to know to code by the time they grow up? Don't think so!

SCENE – II

Design of OS Classroom (Fall 2022).

STUDENT This OS we are using in labs is too obtuse to understand. We should be able to do better. I am interested in RUST, can I pursue developing an instructional OS as my special project?

FACULTY 1 Sure ... Sounds like a good idea!

SCENE – III

Over Classroom Web Chat Channel (Fall 2022).

STUDENT It is just complicated and hard for me to get the x86 register details to implement the bootloader in RUST.

FACULTY 1 I tried generating a RUST bootloader using Chat GPT - here is what it gave. How about you try to use it get what you need?

SCENE – IV

Design of OS Classroom (Spring 2023).

FACULTY 1 How about you TA for the class and let us try to offer labs in BuzzOS?

STUDENT Sounds great.

FACULTY 1 Now that we have many modules of BuzzOS and their chatGPT logs and many students working on the rest of the BUzzOSS modules. Let us submit our experience to ASPLOS WACI for early feedback?

STUDENT Sounds like a great idea. I will create the video and submit.

SCENE – V

On TA Web Chat Channel (Spring 2023).

STUDENT BuzzOS work has been accepted at WACI!

4 Approach

We took a hands-on approach to our exploration. Currently, we are undertaking the development of a Rust-based Operating System with assistance from both Conversational Artificial Intelligence (specifically ChatGPT [2]) and other resources on the internet like OS Dev Wiki [9] and Blog OS [10]. During the process, we record the chat logs, label them according to a variety of metrics - accuracy, usefulness, performance - and store them for future evaluation. The response from the AI is then used to implement new features of the Operating System. Any questions that are raised throughout the process must be answered by the AI, repeating the evaluation procedure. All the logs and source code can be found here:

- BuzzOS source code github repo [11]
- ChatGPT logs with observations [12]

Below is a glimpse of some of our observations:

- **Bootloader** - Comprehensive procedure of enabling A20, unnecessary complexity based on the prompt. Correct ATA driver code in x86 NASM assembly code to load data from the disk. Skipped bits in the ATA status code. Correctly explained the usage of *rep insw* and *rep stosb*.
- **Scheduler** - ChatGPT produced enough information for someone to blindly implement scheduling. Prior knowledge was needed regarding preemption, since it did not mention enabling the clock IRQ on x86. It was necessary to ask it if it was necessary and how to do it (to which it replied correctly, stating the necessity of clock). It provided external sources on how to implement the scheduler and the clock IRQ. Idea of the trap frame (process internal state, such as register state) was not cited. After asking it for the trap frame, it pointed out correctly that storing the trap frame is a common procedure.
- **Heap Allocator** - ChatGPT correctly lists heap allocator designs, which one XV6 uses, and the correct implementation for a linked list (dlmalloc) allocator. One of the problems of ChatGPT becomes evident when asked about external articles on the matter. All of the five articles provided either do not exist, or are not valid anymore.

At the moment, the method described above is being tested in an academic environment. Students are allowed to perform their Special Project at the "Design of Operating Systems" course at Georgia Institute of Technology with the assistance of ChatGPT and similar Conversational AIs. Conversation logs are then captured, processed, and evaluated. Feedback from students is also collected in order to measure effectiveness of the technology. The recording of the interaction between students and the Conversational AI allows for the measurement of time saved on the research procedure, the performance gained, inaccuracy of the response from the AI, and security flaws that were fixed in the design step.

5 Next Steps

With the results from the experiment above, we will be better suited to evaluate if Conversational Artificial Intelligence is a suitable companion for software engineers in the process of designing and developing complex systems. The practical application of the AI allows for an immediate measurement of the feasibility of the proposed method. Errors and "Hallucinations" by the AI are recorded as an observation on the log files, and are later used in the analytical evaluation of the method.

With the result of the above research, prompt engineering of better AI prompts can be conducted, as well as the drafting of best-practices on the usage of Conversational AIs in the complex software development, improving reliability and reducing time wasted.

References

[1] GPT-3 by OpenAI. https://openai.com/blog/gpt-3-apps. (Cited on page 1.)

[2] ChatGPT. https://openai.com/blog/chatgpt. (Cited on pages 1 and 2.)

[3] Five creative ways people are using ChatGPT. https://www.freethink.com/robots-ai/using-chatgpt. (Cited on page 1.)

[4] Integration of ChatGPT on Bing. https://blogs.microsoft.com/blog/2023/02/07/reinventing-search-with-a-new-ai-powered-microsoft-bing-and-edge-your-copilot-for-the-web/. (Cited on page 1.)

[5] Limitations of ChatGPT. https://www.forbes.com/sites/bernardmarr/2023/03/03/the-top-10-limitations-of-chatgpt/?sh=53e41b758f35. (Cited on page 1.)

[6] GitHub Copilot. https://github.com/features/copilot. (Cited on page 1.)

[7] Human Error in Software Development. https://www.sciencedirect.com/science/article/pii/S1000936117300778. (Cited on page 1.)

[8] Human Error in Software Development. https://www.researchgate.net/publication/220615003_Measuring_analyzing_and_predicting_security_vulnerabilities_in_software_systems. (Cited on page 1.)

[9] OS Dev Wiki. https://wiki.osdev.org/Main_Page. (Cited on page 2.)

[10] Blog OS By Phil Opperman. https://github.com/phil-opp/blog_os. (Cited on page 2.)

[11] BuzzOS Source Code. https://github.gatech.edu/cs3210/BuzzOS. (Cited on page 2.)

[12] BuzzOS Chat GPT Logs. https://github.com/JVKdouk/chat-gpt-os-dev-research. (Cited on page 2.)