# Ch(e)at GPT: Using LLMs to Learn Memory-Safe Languages

Brady August, *CSM* Francisco Camacho Cervantes, *CSM*
David Ochoumare, *CSM* Zac Wilson, *CSM*

## Introduction / Motivation

Rust is a security-focused programming language that gives programmers fine-tuned control over the little things such as memory layout, which makes it an exceptional language for coding IoT devices and applications [1]. However, Rust is difficult to learn and involves some distinct differences from other popular languages [2], so finding effective methods to ease its learning process is crucial. In this study, we investigated the extent to which ChatGPT, a popular generative AI model, can help programmers learn Rust. Addressing the complexities of Rust through ChatGPT could democratize access to safe programming, contributing significantly to the development of secure applications. The importance of this research lies in its potential to bridge the gap between the need for secure coding practices and the usability of the tools required to implement them. In doing so, we not only enhance individual programmers' capabilities but also elevate the overall security of commercial software. One thing to consider is that ChatGPT has 2 versions, ChatGPT 3.5 and ChatGPT 4.0. It is important to evaluate how advancements from one version to another could influence the learning experience. ChatGPT 4.0 has been designed to have a better understanding of context and deeper conversations. It also has increased knowledge based on a wider range of training data, which in theory allows it to diagnose code and potential errors better. While both versions of ChatGPT can support interactive learning, we aim to determine to what degree each version can support programmers.

## Related Work

Previous work related to Rust has found that although the language is secure and has potential to decrease memory errors, it is difficult to pick up for programmers. Fulton et al. identifies that one of the primary drawbacks to adopting Rust is its steep learning curve, which provides a strong barrier to entry. The ownership system alongside other unique Rust features like lifetimes can be difficult to grasp, but they do provide safety benefits over typical programming languages [2]. Parastoo Abtahi et al. found that programmers typically use the extensive Rust documentation or Q&A sites such as stack overflow to overcome that learning curve of Rust [3]. Our work builds on this by examining the possibility of using AI to soften the steep learning curve of Rust instead of the typical methods of documentation and Q&A.

Prior work on LLMs has largely been focused on the quality of work that can be produced with the assistance of these tools. Sandoval et al. investigates the cybersecurity impact of LLM code suggestions (from GitHub Pilot), finding that, when writing low level C code, the AI suggestions did not increase the incidence rates of severe security bugs [5]. Similarly, Perry et al. writes about the use of LLMs when writing security-focused code. They found that, with access to an AI assistant, users actually wrote less secure code, but perceived their code as more secure than users without access to AI. However, users who took more time generating prompts were able to eventually come to more secure solutions [6]. These studies, and many others, have found that LLMs can easily output insecure, but functional code. With this preliminary work, other researchers are investigating ways to improve the suggestions that these LLMs provide. Latif Siddiq et al. describes SALLM, a framework to evaluate AI-generated code with a security-based test environment and novel metrics, among other things, to provide an analysis of the code generated by different LLMs [7]. Meanwhile, some researchers such as He et al. are investigating methods to allow LLMs to produce more secure code to begin with SVEN, a learning-based approach to perform security hardening to achieve strong security control without modifying the original LM's weights. Many of the common security vulnerabilities stem from the AI-assisted generation of code in memory-unsafe languages, such as C, or C++. Therefore, these vulnerabilities can potentially be avoided by using an LLM to write code in a memory-safe language, such as Rust. This approach has not been explored previously, likely due to Rust's lower profile among students and in industry. Our work looks to explore this vector, analyzing whether or not Rust can assist developers in writing more secure code by avoiding the root cause of the vulnerabilities present in memory-unsafe languages.

## Methodology

To gather data on how well ChatGPT would perform as a teaching tool we first began with constructing our experimental framework. From our understanding that our mentor provided to us we decided the best way to practically test our participants in Rust would be by utilizing three programming problems involving pointers. We constructed three linked-list problems in C since raw pointers are used often and are an important use case of Rust and C.

Task 1: The first task we constructed was a simple linked list, the participant would need to be able to add any arbitrary number of nodes with an integer data value and a pointer to the next node. The participant would also need to be able to iterate through the list and print out each node in order and finish by properly cleaning up any memory that was allocated. Data values could repeat and would still need to properly print to the terminal for this task to be considered fully functional. We also asked that our participants make a function for adding a new node to the list and a function to print starting at the head to make the code more functional and structured.

Task 2: For the second task we asked our participants to add a function to swap two nodes by passing the integer value of the two nodes to swap. Ex: Swap(3,5) swaps node with data value 3 and node with data value 5. For this task to be considered fully functional a participant needed to print the list prior to the swap and again afterwards, as well any two arbitrary nodes could be chosen for the swap. We limited the parameters of swapped nodes to nodes which don't have the same data value and for the first nodes found in the list with the specified integer values to be chosen.

Task 3: The final programming task for our participants was to update their list structure so that a doubly linked list could be created. This task was considered functional if an arbitrary number of nodes could be added, the list could be printed head to tail and tail to head, and any allocated memory is cleaned up at completion. Their previous swap function was not evaluated on the doubly linked list.

Once we had created the tasks for our participants we created the exit survey they would take once they had either completed all three tasks, expended the two hours we gave them to complete the tasks, or they decided they no longer wanted to work on the tasks. The specific questions that we asked our participants can be found in Appendix A. The most important information we gathered from this survey were the 3 questions on a 7 point likert scale about how well our participants would feel redoing the tasks without ChatGPT's help. These 3 questions indicate to us how well we believe ChatGPT performed at "teaching" our participants. If ChatGPT had performed well our participants would've felt comfortable redoing the tasks without it. We also utilized the exit survey to collect our participant's chat logs with ChatGPT and collected the Rust code they had created, as well as some free response answers.

After we had created our tasks and survey we created a short document to walk our participants through getting a Rust development environment set up on their machine and also laid out the parameters of the experiment. This included specifying that the participant had 2 hours to work through our tasks and could either use the full time or end sooner if they no longer wished to participate. We also specified that our participants could ONLY utilize ChatGPT 3.5 or ChatGPT 4, whichever they had access to, and wanted to use for the experiment. We also gathered which participants used GPT 4 vs GPT 3.5 within our exit survey since we wanted to be sure of any differences between the two models. This documentation can be found in Appendix B.

With all the preliminary work finished, we finally began recruitment for our experiment. Our participants consisted of 5 Master level Colorado School of Mines students who have taken Operating Systems at Mines and were concurrently taking Advanced Computer Architecture. We also recruited 2 professors that we were acquainted with from classes we've taken from each. In total we had 7 participants with 5 participants using ChatGPT 3.5 and 2 using ChatGPT-4. Of the GPT-3.5 and ChatGPT-4 participants the breakdowns are as follows, 4 of the ChatGPT 3.5 participants were students and 1 was a professor. Of the 2 ChatGPT-4 participants 1 was a student and 1 was a professor.

## Results

Fortunately, two of our participants were able to use ChatGPT-4 to complete the tasks. Although two results isn't statistically significant, we've indicated their results compared to the others who used GPT-3.5 with an asterisk (*). The survey results were able to reveal some qualitative, anecdotal data to support our quantitative results. The first survey question asked about the time taken by each participant. Most of them (4/7) took the entire two hours, which was the limit we asked them to not go beyond. One self-reported that they took longer than the full two hours, and the last two took less than two hours: one took less than 30 minutes*, and the other took between one and a half hours and one hour and 55 minutes (*). Of the seven participants, five completed task 1 (**), one completed task 2 (*), and two completed task three (**) . Notably, the participant who completed all three tasks with GPT-4 was the one who took less than 30 minutes. From this alongside the time taken, we can see that the two participants who used ChatGPT-4 were able to complete more tasks in less time than their GPT-3.5 counterparts. However, only one participant was able to complete all three taks, which

indicates that ChatGPT might not be very helpful in this case. We then asked the participants to rate the help that ChatGPT gave them on a scale of 1 (extremely unhelpful), to 7 (extremely helpful). We found that two participants responded with each of 2, 4, and 5 (*), with one participant rating the help a 7 (*). Once again, the GPT-4 respondents were at the higher end of the spectrum. The one participant who completed all three tasks rated their help from GPT a 7, which is understandable considering the relative ease with which they were able to complete everything. These results are scattered, and the sample size is small, so the conclusion we can draw from this data is that the quality of assistance you receive from GPT is based on how you prompt it.

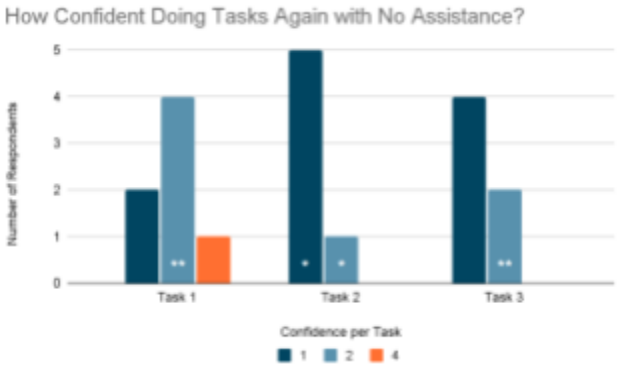



Fig 1: Confidence Repeating Tasks with No Assistance

Figure 1 graphs the results from the next three Likert-scale questions, asking participants to report how confident they would feel if they were asked to repeat the tasks without any assistance, including assistance from ChatGPT. The orange bar in the graph indicates the one participant who rated their confidence repeating Task 1 as a 4, which was the highest number reported by any participant over the three questions. From these results, we can gather that ChatGPT was not a very good teacher. Although many participants were able to complete at least one task, they weren't able to retain the information and details of their implementations very effectively, which is a shortcoming from a learning perspective. Again, this can potentially be attributed to the types of prompts each participant was using. Additionally, this could be because of the shorter time frame, which pressured participants to try to complete the tasks quickly, rather than thoroughly understanding the code they were producing. Next were the free response questions, the first of which asked participants what they thought the largest differences between C and Rust were. We had two major themes in these responses. The first was that rust "seems safer" than C. Five (*) of the seven participants mentioned something that fell into this category of safety differences. The next was an emphasis on pointers and references, which four (*) of seven participants mentioned. This is likely due to the nature of the tasks they were given, which had an emphasis on these differences, so it's reassuring to see that the participants were taking away some differences that we wanted them to see. There were two notable responses, the first mentioned the ownership model, which the participant (*) had heard about in a college course that briefly mentioned Rust. This participant knew the vocabulary of one of the key differences between Rust and C, which could have helped them with their GPT prompts. The second was from the participant who completed all of the tasks (*), and read "To be honest, I took an interesting approach. I did not actually learn rust. I just [kept] asking GPT to fix things until they were fixed. I do not know the differences between rust and c." This participant used ChatGPT very efficiently, but didn't learn anything from the process. This emphasizes the importance of prompt engineering when using AI tools such as ChatGPT, as you can optimize the responses for learning or results based on what you are looking for. The next survey question asked for any other comments, leaving a space for the participants to say anything they thought was relevant. We saw two major themes here as well. The first was learning: a few participants tried to use ChatGPT as a learning tool, this included the participant who reported a four on the no assistance question. The second theme was debugging. Two participants in particular noted that the debugging assistance ChatGPT gave them was unsatisfactory. One participant noted that they actually stopped using ChatGPT to help debug their code in favor of the compiler warnings that Rust was providing them. Finally, the participant who completed all tasks noted that they were able to do so very easily (*).

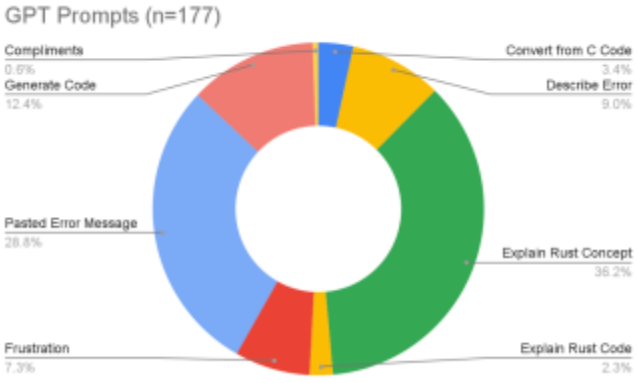



Fig 2: ChatGPT Prompt Coding

The final results from the survey were the ChatGPT logs. We coded each of the 177 prompts the users made into one of eight categories: Convert from C Code, Convert from C Concept, Explain Rust Concept, Explain Rust Code Snippet, Describe Error, Pasted Error Message, Generate Code, Frustration, and Compliments. Describe Error was for

prompts in which the participant typed out a description of the error they were having, as opposed to Pasted Error Message, which was for pasted compiler warnings, with or without the relevant code snippet. Generate Code was for prompts that gave an English description of what they wanted the code to do, and asked ChatGPT to produce Rust code to do so, as opposed to Convert from C Code, which is for prompts that pasted in the provided C code and asked ChatGPT to convert it to Rust. The other categories should be mostly self-explanatory. We can see from Figure 2 that the majority (36.2%) of the prompts were to Explain Rust Concept. This encapsulated prompts such as “Does rust have a class keyword?”, or “Are arguments immutable in rust?” and reasonably encapsulates a lot of the possible questions that the participants were asking. The next largest category was Pasted Error Message at 25.8% of the total prompts. This was also expected, as the participants had never developed in Rust before, so the error messages were likely foreign and participants weren’t able to understand them at first. We saw 13 messages (7.3%) that were frustration-based, in which the participant indicated their frustration to ChatGPT, only some of which had an actual prompt they wanted GPT to respond to. This stands in opposition to one compliment, which was just a simple “Good job!”. This chart gives a breakdown into how the users were using ChatGPT, and based on our personal experiences, this aligns with what we were expecting. Even though most of the prompts were Explain Rust Concept, it seems like the participants were either not able to retain the information well, or the information wasn’t very helpful in the first place, as we saw earlier with the replication questions that had negative results.

Finally, each participant willingly submitted their code to us for evaluation. We were interested to see whether or not ChatGPT fell into the easy solutions to solve some of the tasks, namely using unsafe blocks, and raw pointers. Each of these is a workaround for the built-in memory safety that Rust offers, and would eliminate a lot of the benefit of coding in the language. Fortunately, we only saw one instance of an unsafe block, and one instance of a raw pointer, which were in the same line of code:

```
let node = unsafe { &mut *node_ptr };
```

This was encouraging to see, as ChatGPT was avoiding using these tools when producing the code for the tasks. Additionally, we noticed that each GPT-3.5 participant had the exact same Node struct. We expect this is because of Rust’s age as a language, and that GPT-3.5’s training data ended early in Rusts’ lifespan, leaving less training data to be analyzed and used in the GPT responses. Finally, we saw functional programming concepts used in each implementation, from both GPT-3.5 and GPT-4 participants. This is also encouraging for the general usefulness of ChatGPT as a tool for functional programming. Interestingly, the code that ChatGPT produced in its responses, and similarly the code that the participants submitted to us was lacking in comments. We expected the user-generated code to be short in comments, but typically in other languages, we have noticed that ChatGPT produces code that is well commented, to help user understanding, but those comments were few and far between in the final code the participants sent to us. Interestingly, one participant implemented a stack rather than a linked list for the first task. While it worked essentially the same for the purposes of this exercise, it was interesting that the participant made that. It’s unclear whether or not that was their choice, a misunderstanding, or a misunderstanding from ChatGPT. The final conclusions from the code review was that ChatGPT generally avoids raw pointers, as well as unsafe blocks in Rust, which is good! Additionally, we saw that GPT-4 has a significant advantage over GPT-3.5, partially due to its constant influx of training data. The two outlier participants who completed more than the first task both were using ChatGPT-4. We also noticed that GPT is very comfortable with functional programming, as it uses those concepts frequently and effectively. Similarly, GPT is good at implementations, as participants were able to complete tasks and many rated the help they got from ChatGPT positively, but it falls short in teaching, as participants could not replicate the tasks with no assistance.

## Limitations

The application of ChatGPT as an instructional tool for Rust programming highlights both the potential and the limitations of current AI technologies in educational settings. This study's insights are primarily derived from a pilot project with a small participant group, which inherently limits the strength and generalizability of the conclusions. However, the observations made provided valuable direction for both the improvement of AI-driven educational tools and the design of future studies. A central limitation noted in this study was the small sample size, which was not sufficient to establish statistically significant results. This issue is common in preliminary studies but is a little worse here due to challenges in recruiting participants interested in learning a complex topic like Rust programming through an AI interface. The recruitment difficulty may partially stem from the target population's availability and willingness to engage in an experimental learning setup, which suggests a need for better incentive structures and clearer communication of benefits in future studies. Despite these limitations, the study provided important insights into the practical use of AI tools like ChatGPT in an educational context. Participants' feedback indicated that while ChatGPT could assist with specific programming tasks, it often fell short of providing a comprehensive learning experience. The varied effectiveness of ChatGPT among

participants also raised questions about the consistency and reliability of AI-driven educational tools. Some users reported beneficial interactions with ChatGPT, noting that it helped clarify programming concepts and offered immediate feedback that they found useful for learning. In contrast, others struggled with the AI's responses, which sometimes were inaccurate or misleading, leading to confusion rather than clarity. This inconsistency highlights the need for ongoing improvements in AI programming and training models to enhance reliability and ensure that the tool is genuinely augmenting the educational process. Moreover, the study revealed significant issues related to user interface and instructional clarity. Participants often found the instructions provided by ChatGPT to be unclear or insufficient for completing tasks effectively. This observation suggests that AI tools need to be designed with a keen focus on user experience, ensuring that instructions are not only clear but also tailored to the users' level of expertise and learning context. The limitations of the study itself—such as its scope and the representativeness of the sample—also merit consideration. These limitations include the constrained time frame and budget, which likely impacted the depth and breadth of the investigation. For instance, more extensive testing over a longer period, or with more resources, might have allowed for a more thorough exploration of different teaching methodologies or the inclusion of control groups to compare the effectiveness of AI-assisted learning against traditional methods. This pilot study serves as a crucial stepping stone towards understanding how AI can be effectively integrated into educational frameworks. It emphasizes the necessity of addressing both technical and design challenges to improve the efficacy of AI educational tools. In sum, while the pilot study revealed promising uses of ChatGPT in educational settings, it also highlighted several areas needing refinement and further investigation. Addressing these areas will be essential for harnessing AI's full potential as a supportive educational tool, rather than as a standalone solution.

## Future Work

This pilot study's findings suggest several avenues for future research to expand and refine the use of AI tools like ChatGPT in educational settings. To enhance the strength of the results and further validate the preliminary observations, future projects should aim to recruit a larger and more diverse participant base. This expansion will help determine the effectiveness of AI-assisted learning across various demographic and educational backgrounds, providing a more comprehensive understanding of its potential and limitations. Moreover, incorporating more complex experimental designs is crucial. Future studies could explore longitudinal methods that track learning progress over time, offering insights into the long-term impacts of AI educational tools. Additionally, establishing comparison groups that utilize traditional learning methods alongside AI tools would provide a clearer measure of effectiveness, helping to delineate where AI can most effectively supplement human instruction. Engaging expert analysis from experienced Rust developers in future research would also be invaluable. Their insights would lend credibility to the assessment of the code quality produced by participants and help evaluate the educational outcomes more rigorously. Through these expanded and refined approaches, future research can better ascertain the role and efficacy of AI tools in education, paving the way for their optimized use in learning environments.

## Conclusion

The study of ChatGPT as a tool for teaching Rust programming in this pilot study has provided a glimpse into the potential and limitations of AI in educational settings. The results, albeit preliminary and derived from a small sample size, point towards several key findings that help frame the future of AI-assisted learning. Firstly, the study indicates that while ChatGPT can offer significant assistance in learning programming, its role is best suited as a supplementary rather than a primary educational tool. The AI's ability to provide instant feedback and engage with users on specific programming tasks is valuable, yet the errors and inconsistencies noted in its responses highlight the need for continued development. Participants' experiences varied, with some finding the AI's assistance helpful in clarifying programming concepts, while others were misled by inaccuracies. This variability shows the importance of refining AI technologies to ensure reliability and consistency in educational applications. Additionally, the performance improvements noted with the use of GPT-4 over earlier versions suggest that newer models of AI could enhance learning outcomes more effectively. However, the dependence on the latest technology also highlights the need for ongoing updates and maintenance of educational AI tools, ensuring that they remain effective as teaching aids. The study also revealed critical insights regarding the instructional design and user interface of AI tools. The feedback from participants about unclear instructions and a sometimes confusing user interface emphasizes the necessity of designing AI educational tools that are user-friendly and tailored to the learners' needs. Ensuring clarity in how these tools are to be used is paramount in maximizing their potential to support learning. Furthermore, this pilot study, constrained by resources and time, reflects typical challenges faced in educational research involving technology. These constraints not only impacted the depth of the study but also its breadth, limiting the ability to conduct more comprehensive analysis. Future studies with

more extensive resources could explore additional aspects, such as comparing AI-assisted learning with traditional methods in a controlled setting, to provide more definitive evidence of efficacy. In conclusion, while the use of ChatGPT in teaching Rust programming has demonstrated some effectiveness, the full potential of AI in education remains to be fully realized. The insights gained point towards a promising future where AI could significantly augment educational experiences, provided that the technology continues to evolve and is integrated thoughtfully and strategically into learning environments. The journey of integrating AI into education is ongoing, and this study contributes to that evolving landscape by highlighting both the opportunities and the obstacles that need to be addressed. Moving forward, the focus should be on enhancing AI's accuracy, reliability, and user interaction to truly make it a viable component of educational methodologies.

## References


[1] Verdi, S. 2023b. Why rust is the most admired language among developers. The GitHub Blog.

[2] Fulton, Kelsey R., et al. “Benefits and Drawbacks of Adopting a Secure Programming Language: Rust as a Case Study.” USENIX, www.usenix.org/conference/soups2021/presentation/fulton.

[3] Abtahi, Parastoo, and Griffin Dietz. “Learning Rust: How Experienced Programmers Leverage Resources to Learn a New Programming Language”, Extended Abstracts of the 2020 Chi Conference on Human Factors in Computing Systems, ACM Conferences, 25 Apr. 2020, dl.acm.org/doi/10.1145/3334480.3383069

[4] Perry, Neil, et al. “Do Users Write More Insecure Code with Ai Assistants?” arXiv.Org, 18 Dec. 2023, arxiv.org/abs/2211.03622 .

[5] Gustavo Sandoval, Hammond Pearce, Teo Nys, Ramesh Karri, Siddharth Garg, and Brendan Dolan-Gavitt. Lost at C: A User Study on the Security Implications of Large Language Model Code Assistants. USENIX 2023

[6] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. Do Users Write More Insecure Code with AI Assistants? CCS 2023

[7] Mohammed Latif Siddiq, Joanna C. S. Santos. Generate and Pray: Using SALLMS to Evaluate the Security of LLM Generated Code. arXiv:2311.00889

[8] Jingxuan He, Martin Vechev. Large Language Models for Code: Security Hardening and Adversarial Testing. arXiv:2302.05319

[9] Jung, R. (2020). Understanding and Evolving the Rust Programming Language. Universität des Saarlandes.

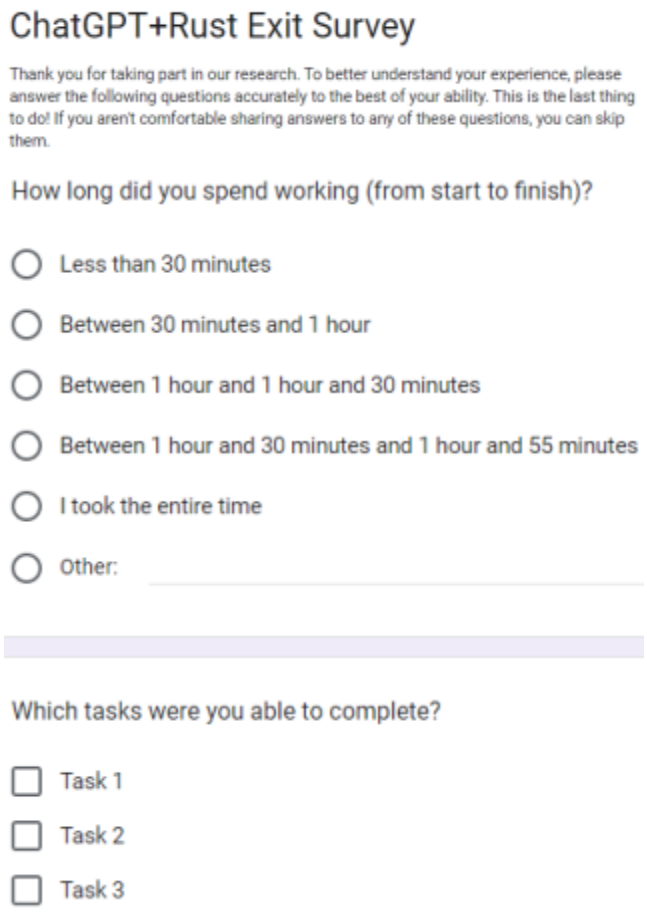
[10] Crichton, W., Gray, G., & Krishnamurthi, S. (2023). A Grounded Conceptual Model for Ownership Types in Rust. SPLASH 2023 OOPSLA.

[11] Roos, J., Kasapovic, A., Jansen, T., & Kaczmarczyk, R. (2023). Artificial Intelligence in Medical Education: Comparative Analysis of ChatGPT, Bing, and Medical Students in Germany. JMIR Medical Education, 9, e46482.

[12] Kuhail, M. A., Alturki, N., Alramlawi, S., & Alhejori, K. (2023). Interacting with Educational Chatbots: A Systematic Review. Education and Information Technologies, 28(1), 973-1018.


## Appendix A: Survey Questions

After completing the tasks, this is the survey the participants were asked to complete. The only required question was the final “Would you be comfortable sharing your ChatGPT conversation history from this project?” question, so we could know whether to show them the following page with instructions or not.



ChatGPT+Rust Exit Survey

Thank you for taking part in our research. To better understand your experience, please answer the following questions accurately to the best of your ability. This is the last thing to do! If you aren't comfortable sharing answers to any of these questions, you can skip them.

How long did you spend working (from start to finish)?

- Less than 30 minutes
- Between 30 minutes and 1 hour
- Between 1 hour and 1 hour and 30 minutes
- Between 1 hour and 30 minutes and 1 hour and 55 minutes
- I took the entire time
- Other:

Which tasks were you able to complete?

- Task 1
- Task 2
- Task 3

How do you feel overall about the help ChatGPT gave you on a scale of 1-7?

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| Extremely Unhelpful | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Extremely Helpful |

How confident would you be if you had to do Task 1 (singly-linked list) again, without any assistance?

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| Not confident at all, nearly impossible | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Extremely confident, it would be easy |

How confident would you be if you had to do Task 2 (swap items in a singly-linked list) again, without any assistance?

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| Not confident at all, nearly impossible | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Extremely confident, it would be easy |

How confident would you be if you had to do Task 3 (doubly-linked list) again, without any assistance?

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| Not confident at all, nearly impossible | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Extremely confident, it would be easy |

From your experience, what do you think are the biggest differences between Rust and C? Are there areas you think Rust is better / worse? (Please don't use ChatGPT to answer this question)

Your answer

Any other comments about your experience or things you want to share?

Your answer

Please email your code to zwilson@mines.edu, and put the email you sent it from below. This will be used to associate your code with your survey response, and will be anonymized as soon as the code and survey results are paired together.

Your answer

Would you be comfortable sharing your ChatGPT conversation history from this project? Your identity will not be shared either way. *

- ○ Yes
- ○ No, I would not like the researchers to see my conversation I had with ChatGPT about this project.

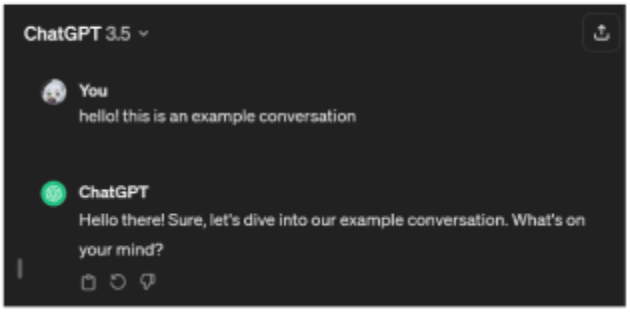
If participants answered 'Yes' on the last question, they would be prompted to complete the ChatGPT log upload. If not, the quiz would end when the participant hit next.

Please share the link to your conversation history from ChatGPT. If you created a new chat before starting, as instructed, this log will only have that conversation's history, and nothing else. Any future messages sent in that chat after the link's generation won't be visible to us, so only the conversation that exists when you export will be viewable.

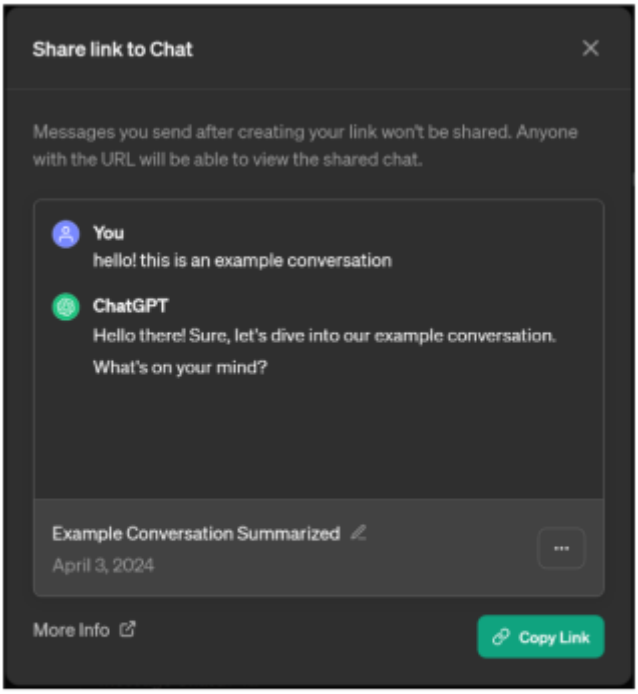
How to Share ChatGPT Conversation

First, click the share button in the top right of the ChatGPT Conversation. Then, use the copy link button to copy, and paste it below.

Share button shown in the Upper Right of this screenshot




Then, click the Copy Link button at the bottom




Please paste your link(s) here!

Your answer

# Appendix B: Participant Documentation

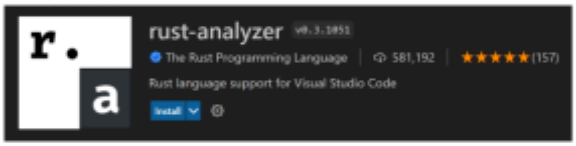
This documentation was provided to participants by email.

## Task 0: Installing + Introduction

Thank you for agreeing to participate in our research. This is the documentation for your use in setting up and completing the research tasks we've made for you. Please read the directions carefully, and complete the tasks to the best of your ability. When completing the programming tasks, you should only use ChatGPT, in any way you would like, to assist you. Don't Google anything, or go to any other websites. When using ChatGPT, please create a new chat for this project rather than using an existing one.

Before beginning, please install Rust on your machine, as well as set up the Rust analyzer in Visual Studio Code. Here are some step-by-step instructions to do so:

- You can install Rust from https://rustup.rs
- After installing Rust, open VSCode and install the rust-analyzer extension as shown below




- Once this is installed, open a new terminal window in VSCode (Ctrl+Shift+`)
- Navigate to where you want to save the files for this exercise
- To create a new project, type `cargo new folder_name` into the terminal
- Then, you should move into that new folder with `cd`
- To run your program, type `cargo run` into the terminal

If you had any trouble with any of these steps, full documentation to set up Rust in VSCode is at https://code.visualstudio.com/docs/languages/rust .
Also, please open ChatGPT, you can find it at https://chat.openai.com/

You can complete all three of the following tasks in src/main.rs, or you can solve each task in its own project, if you prefer. The goal of this research is to evaluate ChatGPT's ability to help individuals write code in Rust who have no prior experience. You will be given three tasks to complete, you should attempt them in order, moving on to the next task when you have completed the prior one, beginning with Task 1. Please take no longer than two (2) hours on this, if you are unable to complete all the tasks, that's ok! After you finish all three tasks, or the time has expired, or you don't want to continue, please take the exit survey under Task 4. Thank you for your participation, and happy coding.

## Task 1: Singly Linked List

The goal of this task is to create a singly-linked list, and add values to it so your final list is in the following form:

[1] → [2] → [3]

You should do this by creating a node structure, and a function to add a new node after another node, which is passed to the function. Below is some C code for your use. This code solves the problem in C, and is what you want to recreate in Rust. You can use it as you'd like. The specifics of your implementation don't have to perfectly match how it's done in C, but the functionality should be the same.

```c
#include <stdio.h>
#include <stdlib.h>


// Define the Node structure
typedef struct Node {
    int data;
    struct Node* next;
} Node;


// Function to add a node at the end of the list
void append(Node** head_ref, int new_data) {
    Node* new_node = (Node*) malloc(sizeof(Node));
    Node* last = *head_ref;

    new_node->data  = new_data;
    new_node->next = NULL;

    if (*head_ref == NULL) {
       *head_ref = new_node;
       return;
    }

    while (last->next != NULL) {
        last = last->next;
    }

    last->next = new_node;
    return;
}


// Function to print nodes in a given linked list
void printList(Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

//free memory
void cleanup(struct Node* node){
    struct Node* temp;
    while (node != NULL) {
        temp = node;
        node = node->next;
        free(temp);
    }
}


int main() {
    Node* start = NULL;

    // The list is: 1 2 3
    append(&start, 1);
    append(&start, 2);
    append(&start, 3);


    printf("Linked list: \n");
    printList(start);

    cleanup(start);
    return 0;
}
```

## Task 2: Singly Linked List - Swap

The goal of this task is to create a singly-linked list, and add values to it so your final list is in the following form:

[1] → [2] → [3]

After creating this list, you should swap the [2] and [3] elements using a swap function, without re-constructing a new list, so your list ends in the form:

[1] → [3] → [2]

If you'd like, you can reuse the list you created in Task 1 rather than re-making it to then swap the elements. Your swap function should take the head of the list, and two integers indicating the values to swap, as parameters. Below is some C code for your use. This code solves the problem in C, and is what you want to recreate in Rust. You can use it as you'd like. The specifics of your implementation don't have to perfectly match how it's done in C, but the functionality should be the same.

```c
#include <stdio.h>
#include <stdlib.h>


// Define the Node structure
typedef struct Node {
    int data;
    struct Node* next;
} Node;


// Function to add a node at the end of the list
void append(Node** head_ref, int new_data) {
    Node* new_node = (Node*) malloc(sizeof(Node));
    Node* last = *head_ref;

    new_node->data  = new_data;
    new_node->next = NULL;

    if (*head_ref == NULL) {
       *head_ref = new_node;
       return;
    }

    while (last->next != NULL) {
```

```
        last = last->next;
    }

    last->next = new_node;
    return;
}


// Function to swap nodes x and y in linked list by changing links
void swapNodes(Node** head_ref, int x, int y) {
    if (x == y) return;
    // Search for x (keep track of prevX and CurrX)
    Node *prevX = NULL, *currX = *head_ref;
    while (currX && currX->data != x) {
        prevX = currX;
        currX = currX->next;
    }

    // Search for y (keep track of prevY and CurrY)
    Node *prevY = NULL, *currY = *head_ref;
    while (currY && currY->data != y) {
        prevY = currY;
        currY = currY->next;
    }

    // If either x or y is not present, nothing to do
    if (currX == NULL || currY == NULL)
        return;


    // If x is not head of linked list
    if (prevX != NULL)
        prevX->next = currY;
    else //make y the new head
        *head_ref = currY;

    // If y is not head of linked list
    if (prevY != NULL)
        prevY->next = currX;
    else // make x the new head
        *head_ref = currX;


    // Swap next pointers
    Node* temp = currY->next;
    currY->next = currX->next;
    currX->next = temp;
}


// Function to print nodes in a given linked list
void printList(Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

//free memory
void cleanup(struct Node* node){
    struct Node* temp;
    while (node != NULL) {
        temp = node;
        node = node->next;
        free(temp);
    }
}


int main() {
    Node* start = NULL;

    // The list is: 1 2 3
    append(&start, 1);
    append(&start, 2);
    append(&start, 3);


    printf("Linked list before swapping: \n");
    printList(start);

    swapNodes(&start, 2, 3); // Swap nodes 2 and 3

    printf("Linked list after swapping: \n");
    printList(start);

    cleanup(start);
    return 0;
}
```

## Task 3: Doubly Linked List

The goal of this task is to create a doubly-linked list, and add values to it so your final list is in the following form:

[1] ⇔ [2] ⇔ [3]

You should be able to traverse this list forwards or backwards due to the doubly-linked list functionality. Similarly to the singly-linked list, you should do this by creating a node structure, and a function to add a new node after another node, which is passed to the function. Below is some C code for your use. This code solves the problem in C, and is what you want to recreate in Rust. You can use it as you'd like. The specifics of your implementation don't have to perfectly match how it's done in C, but the functionality should be the same.

```
#include <stdio.h>
#include <stdlib.h>


// Define the Node structure
typedef struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
} Node;

// Function to add a node at the end of the list
void append(Node** head_ref, int new_data) {
    Node* new_node = (Node*) malloc(sizeof(Node));
    Node* last = *head_ref;

    new_node->data  = new_data;
    new_node->next = NULL;
    new_node->prev = NULL;

    if (*head_ref == NULL) {
       *head_ref = new_node;
       return;
    }

    while (last->next != NULL) {
        last = last->next;
```

```
    }

    last->next = new_node;
    last->next->prev = last;
    return;
}

// Function to print nodes in a given linked list
void printList(Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

// Function to print nodes in a given linked list *backwards*
void printListBackwards(Node* node) {
    while (node->next != NULL) {
        node = node->next;
    }
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->prev;
    }
    printf("\n");
}

//free memory
void cleanup(struct Node* node){
    struct Node* temp;
    while (node != NULL) {
        temp = node;
        node = node->next;
        free(temp);
    }
}

int main() {
    Node* start = NULL;
```

```
    // The list is: 1 2 3
    append(&start, 1);
    append(&start, 2);
    append(&start, 3);


    printf("Linked list printed forwards: \n");
    printList(start);

    printf("Linked list printed backwards: \n");
    printListBackwards(start);

    cleanup(start);
    return 0;
}
```

## Task 4: Exit Survey

Thank you for participating in our research. The last thing to do is the exit survey, which can be found at https://forms.gle/oU9oLoYzgmJLwspr5 . After that, you're done!

## Teammate Contributions

**Brady August:**

I worked to recruit 2 participants for the research, wrote a previous version of the code for task 1, created all of the slides that I presented in class presentations, and wrote, or edited where we already wrote, the introduction and related work sections of the paper. I also made sure that our paper didn't exceed the 5 page maximum through trimming.

**Francisco Camacho Cervantes:**

I wrote the C code for task 3 for our participants, tested functionality of our participants' code, created the slides I presented on the Rust code, wrote the methodology section, and helped recruit a participant.

**David Ochoumare:**

I wrote the code for the participant documentation, made the slides that I presented, helped with the coding for the prompts from our results, wrote the limitations, future work, and conclusion section of the final document.

**Zac Wilson:**

I found some participants for our research, wrote the text part of the participant documentation, spruced up and formatted the code in the participant documentation, made all of the slides I presented, created and set-up the final document, and wrote the Results section as well as added images for both appendices.