A Systematic Evaluation of Code-generating Chatbots for Use in Undergraduate Computer Science Education

Adam Torek
Computer Science
Boise State University
adamtorek@
u.boisestate.edu

Elijah Sorensen
Computer Science
Boise State University
elijahsorensen@
u.boisestate.edu

Natalie Hahle
Computer Science
Boise State University
nataliehahle@
u.boisestate.edu

Casey Kennington
Computer Science
Boise State University
caseykennington@
boisestate.edu

***Abstract*—This research paper focuses on evaluating code-generating chatbots. Chatbots like ChatGPT released in the past three years have proven capable of a wide variety of tasks within a conversational interaction, including writing code and answering code-related questions. With these recent advances, chatbots have many potential uses in education, including computer science education. However, before these chatbots are used in CS curricula, their capabilities and limitations must be systematically tested and understood. In this work, we evaluate the capabilities and limitations of four known, open-source, code-based chatbots in programming tasks by performing a standardized study in which different chatbots are tasked with providing answers for a variety of assignments from Boise State University's computer science program. We found that while all of the chatbots can write code and provide explanations, some do better than others, and each of them work differently in conversations. Moreover, all of them suffered similar and important limitations, which has implications for adoption in curriculum. As a second experiment, we used the Llama chatbot to perform a human evaluation by enabling student novice and experienced programmers to use it as a coding assistant to complete specific tasks in a common software development environment. We found that the coding assistant can help novice programmers accomplish simple tasks in comparable time and code efficacy as more experienced programmers. Given these experiments, and given feedback from participants in our studies, we see a clear picture emerge: new programmers should learn important concepts about programming without the help of code assistants so students can (1) demonstrate their understanding of important concepts and (2) have enough experience to assess code assistant output as useful or erroneous. Then, once intermediate skills are mastered (e.g., object oriented programming and data structures), it seems appropriate to introduce students systematically to coding assistants to help with specific assignments throughout the undergraduate computer science curriculum. We conclude by addressing ethical considerations for the use of code-based chatbots in computer science education and future directions of research.**

***Index Terms*—chatbot, large language models, code copilot**

I. Introduction

Recent advancements in chatbots has attracted great interest from educational institutions and higher learning. Some institutions use chatbots to help with content personalization [1] or use them in online tutoring roles [2]. These chatbots also have the ability to write computer code and solve coding questions [3], which could make them particularly beneficial for computer science (CS) education. CS students have used ChatGPT to help them with programming assignments and have found it to be helpful in debugging and improving code-related thinking skills [4]. While ChatGPT can provide benefits for CS students, there are also potential harms this technology poses, including cheating on exams [5], assignments and projects [6], as well as the potential to make CS students overly reliant on the technology [4].

When presented with these potential benefits and harms, it is clear that systematically evaluating the effectiveness of chatbots on programming assignments is necessary if they are to be applied to CS education. We therefore aim to answer the following research questions in this study:

- What are the capabilities and limitations of code-based chatbots, and how do they affect student answers on questions and problems from real CS assignments?
- How do these capabilities and limitations affect human-chatbot conversations and chatbots' ability to answer questions?
- Can code-based chatbots be integrated in such a way to help novice and experienced programmers perform coding tasks more effectively?

To answer the first and second questions, we conducted an evaluation across multiple recently released open source chatbot models on five assignments, conversing with them to provide a solution for each assignment. We then evaluated these solutions through instructor grading and the percentage of passing test cases. To answer the third research question, we conducted a user study between two groups of programmers to solve a small set of Python problems. One group was given access to chatbots and the other was not. Answering all three of these questions provide insight into how effective and capable coding chatbots are at providing help on CS assignments.

II. Related Work

Code-generating chatbots have systematically been evaluated against benchmarks and have been shown to be effective at solving programming problems [7]. Other work has investigated the potential uses of chatbots in education, both in CS and in other fields. Surveys analyzing different methods for delivery for chatbots [8], [2] have found a variety of ways they can be used in educational settings, including filling different roles and using different interfaces. Other surveys studying the impact of chatbots in educational settings [9], [10] showed that students respond positively to chatbots that are tailored for their education and helped in some areas. Other studies and surveys focused on studying the impact and potential use cases of chatbots in specific fields of education, such as medicine and healthcare [11], engineering and computer science [3], scientific fields such as chemistry [12] and physics [13] and business education [14]. Each of these studies focus on establishing potential use cases, conducting user studies, or finding the capabilities and limitations of chatbots for particular sub-fields of education not related to CS.

A closely related work by Liu et al [15] conducted a study at Harvard that encouraged students in their CS50 computer science class to use chatbots like ChatGPT and Bing Chat to help them solve their assignments. Another study done by Jury et. al. [16] used chatbots to generate working examples for students and evaluated them through two surveys, one where the participants were students and the other survey had experts as participants. Both of these studies are similar to ours because they research using chatbots in computer science education. Both of these studies involved students solving problems using chatbots in an educational setting and collecting their feedback, while our study focuses on evaluating multiple open-sourced chatbots systematically using existing assignments.

Other studies focused on the technical details and implementations of chatbots. Some studies [17], [18] provided training data for fine-tuning chatbots for particular assignments or use cases. Other research papers [19], [20] focused on building models and chatbots that were technically designed to fulfill a particular role in an educational setting. Other studies [21] using ChatGPT have found that its particular user experience significantly accelerates completing assignments. Literature reviews [22] and systemic surveys [23] have provided valuable information about how education is being changed by ChatGPT for both students and faculty.

III. Methodology

In this section, we describe the assignments that we selected from Boise State University's CS curriculum, which code-based chatbots were used for Experiment 1, the evaluation process for each chatbot on each assignment. We also explain the objective metrics and subjective metrics used and the infrastructure used to perform these evaluations.

a) Assignments: In order to be useful in any CS curriculum, state-of-the-art chatbots must be able to answer problems from different assignments throughout the curriculum from beginner to more advanced. To thoroughly evaluate the effectiveness of each model, we chose five assignments from three of our core CS classes: CS 1, CS 2, and CS 3. We chose these courses because they each introduce students to core programming concepts on different levels. CS 1 introduces students to the basics of programming, CS 2 introduces students to object-oriented programming, and CS 3 teaches students about data structures and their applications. We also chose these three courses because they provide different complexity and emphases. The assignments are described below (most assignments come with test cases):

Assignments from CS 1:

- TicTacToe: A Tic-Tac-Toe game in Java with a GUI and a basic AI program. Students use 2D arrays to store the game state, write game logic in different Java classes, and implement a basic GUI to play the game and show move history.
- TextBook: A simple console-based social media site called TextBook. The assignment requires functionality for a single user log in with a username (no password requirement), creating posts, printing posts, deleting posts, adding comments to posts, and storing all comments and posts in text files.

Assignments from CS 2:

- Analysis of Algorithms (**AOA**): This assignment introduces students to analyzing code runtime by having them read code snippets and then create growth functions for best, worst, and average case scenarios, and then justify their reasoning afterward. This assignment does not involve unit tests, but it does include a runtime analyzer that provides hints to the code's growth functions.
- Double Linked List (**DLL**): A double linked list in Java, as well as a ListIterator and unit tests. The double linked list must implement get, set, add, and remove methods, as well as all optional and required ListIterator methods.

Assignments from CS 3:

- Cache Project: A basic cache that follows a most recently used scheme with a fixed size in Java. Students are expected to also write a driver class with command line arguments for testing and debugging their cache.

We chose these assignments for evaluation for multiple reasons, including faculty interest, the general nature of these assignments, and the variety of problems they provide. The TextBook and Tic-Tac-Toe assignments, for example, evaluate the ability of each chatbot to solve basic coding tasks, whereas the DLL and AOA assignments evaluate each chatbot's ability to create a complex data structure and do runtime analysis, and the Cache assignment tests each chatbot's ability to create a cache and driver code. Table I shows a synopsis comparison of these assignments.

Assignment Comparison				
Assignment	Class	Category	Test Cases	Answer Key
TextBook	CS 1	Programming	Yes	No
Tic-Tac-Toe	CS 11	Programming	Yes	No
DLL	CS 2	Data Structure	Yes	Yes
AOA	CS 2	Conceptual	No	Yes
Cache Project	CS 3	Data Structure	Yes	No

TABLE I
ASSIGNMENT COMPARISON BETWEEN DIFFERENT ASSIGNMENTS, INCLUDING CATEGORY, CLASS, AND IF TEST CASES AND/OR TEST SUITES WERE INCLUDED WITH THE ASSIGNMENT.

b) Chosen Code Language Models: We chose four code-based chatbots for our evaluation. We chose these chatbots because they are open source, were recently released, and represent the best chatbots the field has to offer thus far.

- CodeLlama-34B-Instruct-HF: A 34 billion parameter Llama-2 Chatbot model further trained on code-based tasks and fine-tuned through instruction-based human feedback [24].
- Phind-CodeLlama-34B: A 34 billion parameter CodeLlama model (not trained with instructions) fine-tuned by a company called Phind on a closed-source dataset of code-based instruction-answer dialogue [1].
- OctoCoder: A fine-tuned version of StarCoder, a 15-billion parameter code-based LLM trained by HuggingFace and ServiceNow on public GitHub repositories, which was then fine-tuned on GitHub commits [25].
- WizardCoder: A version of StarCoder fine-tuned by WizardLM using a custom fine-tuning method called Evol-Instruct, which used a genetic algorithm to increase the complexity and diversity of code-based problems and solutions, using five metrics to both measure the efficacy of generated responses and create data for further model tuning [26].

Both CodeLlama and Phind-CodeLlama use Llama-2 as their foundation model [27], while WizardCoder and OctoCoder use StarCoder as their foundation model [28].

We decided not to use closed (e.g., ChatGPT) chatbots for two main reasons: Privacy concerns from professors who did not want their assignments uploaded to third parties and the inherent variability of live closed-source chatbots being constantly adjusted in production. We chose to host open-source chatbots on local hardware to address both of these concerns by preventing data from being submitted to third parties and giving us direct control of the environment these chatbots are used for consistency purposes.

c) Evaluation Process: We evaluated each code-based chatbot in a conversational trial-and-error manner to make solutions for each assignment. In total, we conducted 20 conversations between all 4 chatbots on all 5 assignments. Each conversation started with using part the assignment description (as given by the faculty teacher) as input, providing it to the chatbot, letting the chatbot generate a response, evaluating that response for quality and completeness based on answer keys, test cases and human judgement, then selecting a new part of the assignment as input to the chatbot. This continued until we found a complete solution or we could not get a complete solution after a maximum of 30 dialogue (back-and-forth) turns. A graphic showing a visual example of this process can be found in Figure 1.

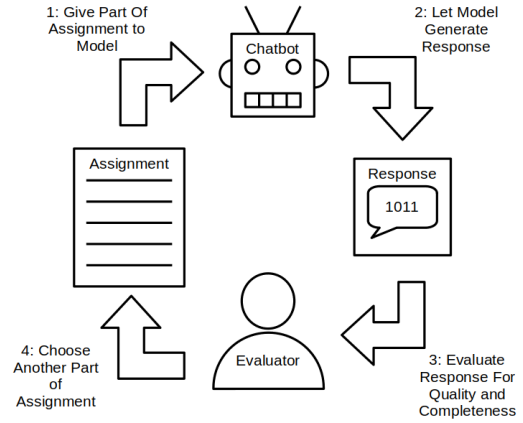

Fig. 1. Diagram of Trial-And-Error Conversation Methodology, step by step. Step 1: Use part of assignment as input to the model. Step 2: Let the model generate a response. Step 3: Evaluate the response for quality and completeness. Step 4: Give another part of assignment based on the given response. Repeat Steps 2-4 for up to 30 turns or until the model produced an adequate response.

After the conversation was complete, each faculty who was actively teaching the course for each assignment graded each as if it were submitted by a student in the class. This sometimes included running test cases to arrive at a final success rate.

For the AOA assignment (which is more focused on meta thinking rather than coding), we followed the same procedure, and included each sub-question on the assignment for each chatbot and asked it to generate a response. If we did not get an appropriate response after two turns, we moved onto the next sub-question.

IV. EXPERIMENTS

A. Experiment 1: Systematic Evaluation of Open-source Coding LLMs

We conducted all of the evaluations on the same PC, which was running Ubuntu 22.04 LTS, an Nvidia RTX 3090 GPU with 24 GB VRAM, a 16-core 32-thread AMD Ryzen CPU, and 64 GB of RAM. Each chatbot was run on the RTX 3090 using 4-bit floating point quantization to shrink memory requirements and make models larger than 7B parameters fit inside GPU memory. All the chatbots we tested were given no pre-prompts. We used same text generation parameters for all models except for input size and context length (both Phind-CodeLlama and Meta-CodeLlama were limited to an

[1]https://www.phind.com/blog/code-llama-beats-gpt4

Results for CS 1 Assignments						
Code-Based ChatBot	TextBook			TicTacToe		
	Grade	Tests Passed	Turns	Grade	Tests Passed	Turns
Meta-CodeLlama	36%	44.5%	31	30%	60.2%	32
Phind-CodeLlama	76%	94.5%	30	51.3%	95.9%	29
OctoCoder	4%	0%	29	3.33%	0%	30
WizardCoder	1.33%	0%	31	0%	0%	31

TABLE II
GRADES, CONVERSATION TURNS, AND TEST CASES PASSED FOR BOTH TEXTBOOK AND TICTACTOE

Table 4.2: Results for CS 2 And CS 3 Assignments								
Code-Based ChatBot	DLL			AOA		Cache		
	Grade	Tests Passed	Turns	Grade	Turns	Grade	Test Passed	Turns
Meta-CodeLlama	21.7%	21.9%	31	20%	31	86.7%	100%	32
Phind-CodeLlama	86.7%	97.6%	31	20%	33	87.8%	100%	21
OctoCoder	0%	0%	31	20%	30	0%	0%	31
WizardCoder	0%	0%	32	20%	30	0%	0%	31

TABLE III
GRADES AND CONVERSATION TURNS FOR AOA; GRADES, CONVERSATION TURNS, AND TEST CASES PASSED FOR BOTH DLL AND CACHE

input length of 1536 tokens and output length of 2048 tokens respectively because longer input length resulted in severe bottlenecks and time-outs; WizardCoder and OctoCoder were given an input length of 2048 tokens and an output length of 4096 tokens because these chatbots tended to generate very long outputs). We used the following software for loading the models, user interface, and coding.

- HuggingFace's text-generation-inference engine[2] to run each chatbot on the PC
- HuggingFace's chat-ui[3] as the conversation interface
- MongoDB[4] to store conversation data
- Visual Studio Code[5] for running code outputted by the chatbot

B. Evaluation Metrics

For the experiments, we used the following objective evaluation metrics for evaluating the overall quality of each generated solution:

- Number grades reflected as percentages from course instructors
- Percent of test cases passed (for assignments with test cases)
- Number of back-and-forth dialogue turns

These evaluation metrics were chosen because they emulate the evaluation of student solutions. The grade from the instructor is used as a metric for the overall quality of the generated solution, and the percent of test cases passed is used as a metric for measuring the solution's completeness with the requirements. We include the number of dialogue turns in the evaluation metrics to show that some solutions took slightly more turns to generate than others, which can indicate the overall quality per response in each solution.

[2]https://github.com/huggingface/text-generation-inference.git
[3]https://github.com/huggingface/chat-ui.git
[4]https://www.mongodb.com/
[5]https://code.visualstudio.com/

For consistency, the same person performed the evaluations for each assignment for each model. The chosen individual is a student in our CS undergraduate program, nearing graduation. They have taken the courses in question and have completed the assignments previously. We use insights gained from this individual for additional subjective, qualitative results that we include below.

1) Objective Experimental Results: Tables II and III show the grades, the percentage of test cases passed, and number of dialogue turns for each each chatbot on each assignment. It is clear from both tables that Phind-CodeLlama and Meta-CodeLlama both did better in both test cases passed and grades than WizardCoder and OctoCoder.

Both Meta-CodeLlama and Phind-CodeLlama were able to write solutions that partially passed test cases and got partial credit. Of the two better performing chatbots, Phind-CodeLlama did the best by far, passing more than 90% of test cases and getting above 50% in most grades. Meta-LlamaCode did worse than Phind-CodeLlama but better than WizardCoder and OctoCoder, scoring between 20% to 60% in test case passing rates with grades between 20% and 50% for most assignments. Almost all conversations continued until around 30 dialogue turns except Phind-CodeLlama's Cache solution, which was both of acceptable quality and fully complete at 21 dialogue turns.

WizardCoder and OctoCoder both failed to pass all test cases, getting a 0% passing rate across all test cases and got grades close to or at 0% for most assignments as well. They did poorly across all assignments because their code was full of errors and would not compile. The only assignment all chatbots did poorly in was AOA, with all of them getting a grade of 20% due to numerous logical errors and too much irrelevant information.

C. Subjective Results

During evaluations, the evaluator found that each chatbot had a unique way of conversing and generating code solutions—a kind of limited, but different *personality* for each model. They found that this *personality* greatly affected the responses and the conversation process:

Meta-CodeLlama: Often followed my instructions closely and carefully, but made mistakes when the input was not in the right format. Whenever I told it the answers that it gave were wrong and incomplete, Meta-CodeLlama would apologize and write a new answer based on the prompts it was given. Meta-CodeLlama rarely rejected to answer my prompts outright, except in one instance where it refused to provide a solution for the DLL in the first prompt because the chatbot recognized that it was a class assignment. I think this happened because of some kind of safety mechanism I triggered, but I did not confirm that. Overall, Meta-CodeLlama tried to be helpful and provide good solutions. Comments were included in some code snippets, however these comments are limited or not generated unless I told it to.

Phind-CodeLlama: I found it to be like Meta-CodeLlama, but was highly code focused, responding with code snippets in the vast majority of cases, and would sometimes give descriptions after each response. I found that Phind-CodeLlama was the most "GPT-like" of all the models I tested and was able to bridge gaps in prompts and respond to non-specific instructions with specific code snippets. I also found that Phind would sometimes freeze and not produce any output when prompted to do something not directly code related and sometimes refuse to fix because convinced itself it had the right solution even if I knew it didn't.

OctoCoder: Wrote very verbose code with detailed comments. I found that OctoCoder always generated lengthy code snippets with a lot of helper methods and internal variables. It would do okay if I gave it short instructions about generic problems, but it often meandered on larger problems so much that its answers were cut off, even with a 4096 token limit. I also found that OctoCoder often hallucinated variables or methods, generated very long names, and got stuck generating the same token sequence over and over again. OctoCoder generated the most descriptive and accurate comments out of all the chatbots I tested, which I found odd considering it had poor quality code overall.

WizardCoder: Likes generating complicated code snippets and explanations using a variety of strategies and moving parts. Like OctoCoder, WizardCoder's answers were often long and complex and were often cut off at the 4096 token limit. Like OctoCoder, I found that WizardCoder did fine on smaller problems but not on larger problems that required specific implementations. It often over-complicated answers to conceptual questions as well by generating verbose answers, sometimes with logical fallacies. Unlike OctoCoder, WizardCoder sometimes complimented me randomly or even refused to generate solutions due to a self-reported lack of confidence, especially when I told to fix a mistake or error in its code. WizardCoder also repeatedly got stuck on the token $< >$ when generating code snippets, even when I explicitly told it not to.

D. Discussion

While Phind-CodeLlama and Meta-CodeLlama did better than WizardCoder and OctoCoder in the objective metrics, all chatbots we tested in this evaluation had similar issues in responding to instructions. They all had incidents where they did not provide answers at all. Some chatbots got stuck on particular tokens (such as WizardCoder getting stuck on $< imp\_sep >$), while others got stuck on particular instructions or formatted input. All chatbots we tested also got stuck on particular problems or kept generating output that was incorrect or nonsensical. Sometimes the chatbots repeatedly generated the same code or answers over and over again, both inside and between prompts. In all cases, the only consistent way we found to keep the chatbots moving toward solutions was to move onto a different part of the assignment in question.

All of the chatbots we tested also generated code or answers that contained so-called "hallucinations" or made up information that was stated as fact [29]. Examples include names of code objects that were nonexistent, including functions, variables, methods, and classes for code based problems. We noticed that this issue was even more pronounced for the AOA assignment, where chatbots consistently invented lines of code, algorithms, runtime functions, constants, and variables where none existed. All chatbots also made logical or arithmetic errors during the evaluation that we could not correct, even if we provided many different forms of the same prompt.

E. Experiment 2: Novice vs. Experienced Programmers

The results of Experiment 1 showed the effectiveness for the Llama models to be used for interactive code assistance. In this experiment, we implemented an extension to a popular programming tool, VSCode, that interfaces directly with the Llama coding model and recruited a group of Novice programmers and another group of Experienced programmers and tasked them with programming several specific problems.

a) Setup: Our chosen model was LLama with 7 billion parameters, with a max token response of 512 tokens. We used Streamlit, an application that turns scripts into web pages, to host the Llama model.[6] Our VSCode extension works as follows: as the user inputs text into an active editor, the extension collects text from the input cursor's current line. When the user finishes typing, after a small timeout (to prevent overloading the locally run model), a request is made to the StreamLit service that is hosting the Llama—Python model. After the model returns a suggestion, VSCode displays it within the active editor, allowing the user to accept or reject it, either all at once, or on a word-by-word basis. Figure 2

[6]https://streamlit.io/

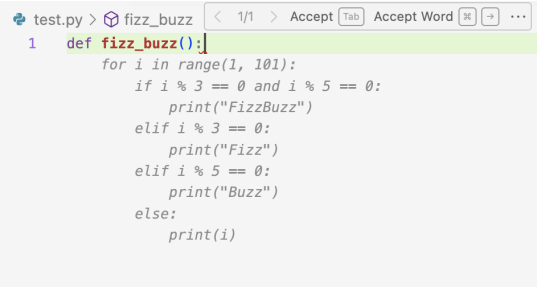

Fig. 2. A code completion suggested by the editor, given the input, "def fizz_buzz():"

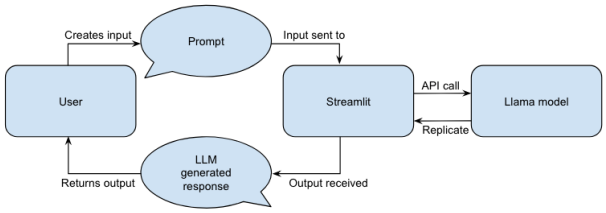

Fig. 3. The pipeline of the Chatbot. The user sends a prompt through Streamlit, which sends this input as a request to our LLama model. After performing necessary computations, Replicate returns the model's response.

shows how this process looks to the user. The overall pipeline is shown in Figure 3.

b) Procedure: We recruited users with varied programming experience levels (beginner to experienced) while still ensuring that all participants had at least some background in writing and reading code. We then sorted the participants into two subgroups: 9 in Novice (fewer than 4 years of experience) and 10 in Experienced (greater than four years of experience) programmers. We also asked users about their overall experience and knowledge of AI to better understand the pool of participants. Table IV details the range of user experiences across both groups concerning use or knowledge of AI and generative content. It is evident that a great majority of the participants have a background in either Computer or Data Science. Similarly, most participants reported to have used AI content before to some degree, which is hardly surprising given its ever-increasing prevalence. On the other hand, only around a slight majority of the total participants stated to have specific knowledge surrounding the intricacies of artificial intelligence or machine learning.

The population of users that attempted our selected tasks ranged from beginner to expert in programming experience. We attempted to target users with at least some experience in Python programming (the language of choice for our study) in order to measure how our tools could aid those with differing levels of experience in the programming language we selected to focus on. Tables V and VI detail statistics collected from both groups regarding coding background and Python experience, respectively. Regarding Python experience specifically, half of participants identified as beginner or novice, having minimal experience with Python (i.e. under one year of total experience). The Novice group in particular especially exhibited a vast majority of novice Python experienced individuals, where 78% of users within this group reported under a year of experience. The Experienced group reported a slightly more balanced distribution of Python experience, where the majority of the participants reported having at least 1 year of Python experience.

The coding tasks we asked the participants to complete varied in increasing difficulty. We designed this set of tasks to be completed without AI assistance in a timely manner—ideally no longer than 25 minutes total while still requiring critical thought from participants in order to complete them effectively. Users were allowed to skip a task if it was too difficult. The three tasks are detailed below:

- Task 1: Create a loop that counts the total number of even whole numbers divisible by 7 in the range 1-200. Increase numCounts when this criterion is met.
- Task 2: Use the init() function to assign values for name, id, and age in a Student class.
- Task 3: Create a function that squares a given input integer, but only when that integer is a perfect square less than 150. Append each returned integer that meets this criteria to the `double_squared` list.

c) Metrics: After completing the tasks, participants filled out a survey in order to gather information on their experiences. We created two forms with similar questions about AI and programming experience, completion of tasks, and the usefulness (or absence of) AI in their experience. Some sample questions/prompts include:

- Did you complete Task X?
- I found the following Tasks difficult (select all that apply)
- I wish I had AI for the following tasks:
- The AI actually helped on the following Tasks (select all that apply)
- I used chatbot AI on the following Tasks (select all that apply)
- I used code completer AI on the following Tasks (select all that apply)

We also collected data on how much time participants spent on tasks, percentage of tasks completed (and if those tasks were correct).

d) Results: Table VII depicts the results of time taken to complete the tasks across both groups. Overall, the average times for the Novices and Experienced groups were 23 minutes and 25 minutes respectively.

The results of task completion (where tasks were done correctly) is shown in Table VIII. Task 3 was particularly challenging, as both groups reported only around half of their participants passing the task. This task was consistently rated the most difficult; both groups either used, or wished to

AI Knowledge & Background	Experienced (% of people)	Novices (% of people)	Total
Pursuing or pursued a degree in Computer or Data Science	90%	89%	89%
Knowledge of AI learning models	70%	67%	68%
Used ChatGPT or other AI generated content	80%	89%	84%
Not used AI generated content before	10%	11%	11%

TABLE IV
INFORMATION ABOUT PARTICIPANTS' BACKGROUND IN AND/OR KNOWLEDGE OF AI IN THE EVALUATION PHASE.

Years	Exp	Nov	Total
0-1	0%	11%	5%
1-3	30%	44%	37%
4-8	40%	33%	37%
8+	30%	11%	21%

TABLE V
PARTICIPANTS' CODING BACKGROUND.

Python	Exp	Nov	Total
0-1	30%	78%	53%
1-2	50%	22%	37%
2-4	10%	0%	5%
4+	10%	0%	5%

TABLE VI
PARTICIPANTS' PYTHON BACKGROUND.

Task	Experienced	Novices	Total
Task 1	90%	78%	84%
Task 2	80%	100%	89%
Task 3	60%	56%	63%

TABLE VIII
TASK COMPLETION/PASSING PERCENTAGES.

use, AI for assistance. In addition, 100% of the users in the experiment group reported using AI assistance for Task 3, but only two users claimed the AI actually helped for that task. Although AI was sought after or used for Task 3, it proved little use to actually help in this case (based on our current implementation).

Task 2 focused on Python conventions in object constructors; one must know basic Python object syntax in order to properly complete the constructor. This led to two users failing this test in the Experienced group whereas the Novice group had all participants passing (although it is worth mentioning that one of those two participants did not have time to attempt Task 2 in the Experienced group as described in written responses). This could be due to participants attempting and failing the task due to not knowing Python object syntax (data scientists often do not learn object-oriented programming), whereas the Novice group defaulted to AI for Python object syntax questions. This possibility becomes more apparent as the Novice group reported 100% use of AI when completing Task 2, and all but one participant reported that the AI was helpful in doing so.

Task 1 proved a nuanced mix; this task featured basic algorithm strategies with more users passing in the Experienced over the Novice group. There were 0% of users across both groups that found this task challenging or tedious to complete, but from user observations, there was a note that those using the chatbot to complete this task would receive code that prints the value, instead of incrementing it, as stated in the task's instructions. This meant that users had to modify the code to pass our tests.

Time spent	Experienced	Novices	Total
10-15	30%	0%	16%
15-20	20%	22%	21%
20-25	20%	56%	37%
25-30	20%	22%	21%
30	10%	0%	5%

TABLE VII
TIME SPENT ON TASKS FOR EACH GROUP.

As detailed in Table IX, the survey results revealed that most individuals wished for AI use on specific tasks. As expected, Novice programmers want AI assistance more overall, even for easy tasks, whereas more experienced programmers do not want to rely on AI assistance. We assumed Tasks 1 and 2 would have similar rates of desire for AI aid, due to their apparent similar difficulties, but Task 2 ultimately had a lower completion rate, potentially resulting in a greater desire for AI aid among participants. Overall, the rates of desire for AI aid were not as high as originally anticipated, but when examining group differences, the average participant in this group had one more year of Python experience than the average participant from the Novice group. Therefore, the average control group participant would in theory need less assistance and aid when completing simple Python coding tasks.

Task	Experienced	Novices
Task 1	30%	67%
Task 2	60%	89%
Task 3	80%	100%

TABLE IX
EXPERIENCED GROUP SURVEY RESULTS.

Novice programmers further expressed that the AI assitance really helped in the completion of tasks, particularly Task 2 (78% of respondents). This is an interesting result; it points to AI code assitance chatbots more effective in some specific areas than others (e.g., Task 1 vs. Task 2), and that interacts with the kinds of things that Novice programmers may be comfortable doing (simple scripts as in Task 1) or find more complicated (writing classes as in Task 2), therefore necessitating help from the AI tool. Overall, the AI usage among the Novice group proved quite insightful, as those with little to no Python experience could still complete tasks at similar success rates compared to those more experienced in the language.

Finally, we include some of the open-ended feedback feedback from the participants in both groups. Some Experienced group testimonials were as follows:

- "What took the longest was understanding what a perfect integer is"
- "what is a perfect square"

- "Time constraints, had to skip task 2"

Two of the above comments are related to Task 3 and the specific terminology associated with that task. Some Novice group testimonials were as follows:

- "LLM's are way to[o] good, I am truly shocked by how good they are"
- "the chatbot showed me how to do a def__init__ for question 2 and how to check the type of a variable for question 3"
- "the chatbot doesn't always give you exactly what you want compared to others i have used e.g chatgpt. I had to re ask specifically what i wanted before it gave them to me"
- "I am not very familiar with Python, but the AI made it easier to understand conventions."

F. Discussion

Given the findings of our evaluation surrounding the usefulness of AI in aiding individuals as they program, a key takeaway is that there is great potential in generative AI tools and software for this use case. Our goal was to provide access to such tools as local applications that anyone could utilize without excessive or exhaustive overhead on one's own personal machine. The results of our evaluation were interesting, as both the Experienced and Novice groups were surprisingly similar. Neither considerably outperformed the other, both in task completion rates and total time. When taking a closer look, the group with access to AI tools had a massive difference in technical and target programming language experience. Specifically, the great majority of participants using the AI tools had little to no target programming language experience (or general coding experience), while the control group had a great majority of at least one or more years of target programming language experience (and much more general coding experience). Given this key information, it's telling that those with much lower technical experience, with access to AI, can perform somewhat on par with those having greater technical experience gained from years of practice.

However, a critical item missing from our evaluation is retention of knowledge. Given our intent to illustrate AI's usefulness in the classroom, the question remains: did the users in the experiment group *learn* anything valuable about Python programming as a result of their interaction with Code Llama? We leave this important pedagogical question for future work.

V. Limitations & Ethical Considerations

It should be noted that while we evaluated these chatbots using a standard process, this process did not emulate or take place in an educational setting. We performed these evaluations to test the ability of code-based chatbots on various problems from CS assignments. These evaluations were not done with groups of randomly selected students in a particular cirriculum. This means that results of this study, while informative, are not from a classroom setting and are not representative of all possible use cases.

While we did not encounter any incidents of offensive or harmful output occurred, it is important to note that chatbots are based on models trained on data gathered from the Internet, and as such may contain harmful, offensive, or toxic content. The chatbots used in this study may have learned these harmful biases, and may repeat them in response to certain prompts. As noted in the Experiments section, we found that the code-based chatbots also generate incorrect output, information, and code snippets, which may confuse or mislead unknowing or unsuspecting students. We should also note that code-based chatbots could also be abused by students to create harmful output, write malicious code, or use them to cheat on programming assignments and exams. We strongly encourage that any faculty who wants to use this technology take these issues into account before applying it to their classrooms.

VI. Conclusion

We systematically evaluated 4 open-source coding language models on existing CS assignments. The results showed generally that the models were useful, though with nuanced results depending on the model and assignment. The models could handle easier assignments, which has implications for new students who might use code assistants as a tool as they learn to program but incorrectly assume that the code assistant can solve all coding problems as they become more complex. We then evaluated the best model—Llama—with human participants grouped into Experienced and Novice programmers, and found that the AI tools aided the Novices to be comparable in time efficiency and task completion.

The results from this study could be used as initial information for future work involving studies between chatbots and students, potential ways to improve code-based chatbots to better suit a CS department's needs, and give helpful insights that could help CS departments adapt their education.

Future work can expand on this study by conducting a larger study with students in which they can use code-based chatbots to solve programming problems and give feedback about their experiences. This work would allow students to give feedback on the effectiveness and quality of using this technology from their perspective. We are cautiously optimistic that these future directions of work could give CS education institutions the information and tools they need to apply code-based chatbots in their curricula while mitigating potential harms.

Acknowledgements

We wish to thank the anonymous reviewers for their helpful feedback and comments. We also wish to thank the faculty who provided the assignments and evaluations. This work was approved by Boise State University IRB protocol IRB24-109.

REFERENCES

[1] J. Pereira, "Leveraging chatbots to improve self-guided learning through conversational quizzes," in *Proceedings of the fourth international conference on technological ecosystems for enhancing multiculturality*, 2016, pp. 911–918.

[2] M. W. Ashfaque, S. Tharewal, S. Iqhbal, and C. N. Kayte, "A review on techniques, characteristics and approaches of an intelligent tutoring chatbot system," in *2020 International Conference on Smart Innovations in Design, Environment, Management, Planning and Computing (ICSIDEMPC)*. IEEE, 2020, pp. 258–262.

[3] J. Qadir, "Engineering education in the era of chatgpt: Promise and pitfalls of generative ai for education," in *2023 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 2023, pp. 1–9.

[4] R. Yilmaz and F. G. Karaoglan Yilmaz, "Augmented intelligence in programming learning: Examining student views on the use of chatgpt for programming learning," *Computers in Human Behavior: Artificial Humans*, vol. 1, no. 2, p. 100005, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2949882123000051

[5] T. Susnjak, "Chatgpt: The end of online exam integrity?" 2022.

[6] D. R. Cotton, P. A. Cotton, and J. R. Shipway, "Chatting and cheating: Ensuring academic integrity in the era of chatgpt," *Innovations in Education and Teaching International*, pp. 1–12, 2023.

[7] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," *CoRR*, vol. abs/2107.03374, 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

[8] M. A. Kuhail, N. Alturki, S. Alramlawi, and K. Alhejori, "Interacting with educational chatbots: A systematic review," *Education and Information Technologies*, vol. 28, no. 1, pp. 973–1018, 2023.

[9] A. T. Neumann, T. Arndt, L. Köbis, R. Meissner, A. Martin, P. de Lange, N. Pengel, R. Klamma, and H.-W. Wollersheim, "Chatbots as a tool to scale mentoring processes: Individually supporting self-study in higher education," *Frontiers in artificial intelligence*, vol. 4, p. 668220, 2021.

[10] S. Hobert and F. Berens, "Small talk conversations and the long-term use of chatbots in educational settings–experiences from a field study," in *Chatbot Research and Design: Third International Workshop, CONVERSATIONS 2019, Amsterdam, The Netherlands, November 19–20, 2019, Revised Selected Papers 3*. Springer, 2020, pp. 260–272.

[11] F. Frangoudes, M. Hadjiaros, E. C. Schiza, M. Matsangidou, O. Tsivitanidou, and K. Neokleous, "An overview of the use of chatbots in medical and healthcare education," in *International Conference on Human-Computer Interaction*. Springer, 2021, pp. 170–184.

[12] A. T. Lolinco and T. A. Holme, "Developing a curated chatbot as an exploratory communication tool for chemistry learning," 2023.

[13] J. R. Aguilar-Mejía, S. Tejeda, C. V. Ramirez-Lopez, and C. L. Garay-Rondero, "Design and use of a chatbot for learning selected topics of physics," in *Technology-Enabled Innovations in Education: Select Proceedings of CIIE 2020*. Springer, 2022, pp. 175–188.

[14] D. Pijetlovic and G. Mueller-Christ, "Humanrobolab: Experiments with chatbots in management education at universities," in *Diginomics research perspectives: The role of digitalization in business and society*. Springer, 2022, pp. 1–12.

[15] R. Liu, C. Zenke, C. Liu, A. Holmes, P. Thornton, and D. J. Malan, "Teaching cs50 with ai," 2024.

[16] B. Jury, A. Lorusso, J. Leinonen, P. Denny, and A. Luxton-Reilly, "Evaluating llm-generated worked examples in an introductory programming course," in *Proceedings of the 26th Australasian Computing Education Conference*, ser. ACE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 77–86. [Online]. Available: https://doi.org/10.1145/3636243.3636252

[17] T. Gerald, S. Ettayeb, H. Quang Le, A. Vilnat, G. Illouz, and P. Paroubek, "Un corpus annoté pour la génération de questions et l'extraction de réponses pour l'enseignement (an annotated corpus for abstractive question generation and extractive answer for education)," in *Actes de la 29e Conférence sur le Traitement Automatique des Langues Naturelles. Volume 3 : Démonstrations*. Avignon, France: ATALA, 6 2022, pp. 15–17. [Online]. Available: https://aclanthology.org/2022.jeptalnrecital-demo.5

[18] F. Petersen-Frey, M. Soll, L. Kobras, M. Johannsen, P. Kling, and C. Biemann, "Dataset of student solutions to algorithm and data structure programming assignments," in *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, Jun. 2022, pp. 956–962. [Online]. Available: https://aclanthology.org/2022.lrec-1.101

[19] R. Abdulrahman and H. Hassani, "A language model for spell checking of educational texts in Kurdish (Sorani)," in *Proceedings of the 1st Annual Meeting of the ELRA/ISCA Special Interest Group on Under-Resourced Languages*. Marseille, France: European Language Resources Association, Jun. 2022, pp. 189–198. [Online]. Available: https://aclanthology.org/2022.sigul-1.25

[20] F. Clarizia, F. Colace, M. Lombardi, F. Pascale, and D. Santaniello, "Chatbot: An education support system for student," in *Cyberspace Safety and Security: 10th International Symposium, CSS 2018, Amalfi, Italy, October 29–31, 2018, Proceedings 10*. Springer, 2018, pp. 291–302.

[21] X. Zhai, "Chatgpt user experience: Implications for education," *Available at SSRN 4312418*, 2022.

[22] C. K. Lo, "What is the impact of chatgpt on education? a rapid review of the literature," *Education Sciences*, vol. 13, no. 4, p. 410, 2023.

[23] M. M. Rahman and Y. Watanobe, "Chatgpt for education and research: Opportunities, threats, and strategies," *Applied Sciences*, vol. 13, no. 9, p. 5783, 2023.

[24] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve, "Code llama: Open foundation models for code," 2023.

[25] N. Muennighoff, Q. Liu, A. Zebaze, Q. Zheng, B. Hui, T. Y. Zhuo, S. Singh, X. Tang, L. von Werra, and S. Longpre, "Octopack: Instruction tuning code large language models," 2023.

[26] Z. Luo, C. Xu, P. Zhao, Q. Sun, X. Geng, W. Hu, C. Tao, J. Ma, Q. Lin, and D. Jiang, "Wizardcoder: Empowering code large language models with evol-instruct," 2023.

[27] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," 2023.

[28] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, M. Davaadorj, J. Lamy-Poirier, J. Monteiro, O. Shliazhko, N. Gontier, N. Meade, A. Zebaze, M.-H. Yee, L. K. Umapathi, J. Zhu, B. Lipkin, M. Oblokulov, Z. Wang, R. Murthy, J. Stillerman, S. S. Patel, D. Abulkhanov, M. Zocca, M. Dey, Z. Zhang, N. Fahmy, U. Bhattacharyya, W. Yu, S. Singh, S. Luccioni, P. Villegas, M. Kunakov, F. Zhdanov, M. Romero, T. Lee, N. Timor, J. Ding, C. Schlesinger, H. Schoelkopf, J. Ebert, T. Dao, M. Mishra, A. Gu, J. Robinson, C. J. Anderson, B. Dolan-Gavitt, D. Contractor, S. Reddy, D. Fried, D. Bahdanau, Y. Jernite, C. M. Ferrandis, S. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries, "Starcoder: may the source be with you!" 2023.

[29] Z. Xu, S. Jain, and M. Kankanhalli, "Hallucination is inevitable: An innate limitation of large language models," Jan. 2024.