CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes

Mark Liffiton
mliffito@iwu.edu
Illinois Wesleyan University
Bloomington, Illinois, USA

Brad Sheese
bsheese@iwu.edu
Illinois Wesleyan University
Bloomington, Illinois, USA

Jaromir Savelka
jsavelka@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Paul Denny
paul@cs.auckland.ac.nz
The University of Auckland
Auckland, New Zealand

ABSTRACT

Computing educators face significant challenges in providing timely support to students, especially in large class settings. Large language models (LLMs) have emerged recently and show great promise for providing on-demand help at a large scale, but there are concerns that students may over-rely on the outputs produced by these models. In this paper, we introduce CodeHelp, a novel LLM-powered tool designed with guardrails to provide on-demand assistance to programming students without directly revealing solutions. We detail the design of the tool, which incorporates a number of useful features for instructors, and elaborate on the pipeline of prompting strategies we use to ensure generated outputs are suitable for students. To evaluate CodeHelp, we deployed it in a first-year computer and data science course with 52 students and collected student interactions over a 12-week period. We examine students' usage patterns and perceptions of the tool, and we report reflections from the course instructor and a series of recommendations for classroom use. Our findings suggest that CodeHelp is well-received by students who especially value its availability and help with resolving errors, and that for instructors it is easy to deploy and complements, rather than replaces, the support that they provide to students.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **Software engineering education**; • **Human-centered computing** → **Interactive systems and tools**.

KEYWORDS

Intelligent tutoring systems, Intelligent programming tutors, Programming assistance, Novice programmers, Natural language interfaces, Large language models, Guardrails

ACM Reference Format:
Mark Liffiton, Brad Sheese, Jaromir Savelka, and Paul Denny. 2023. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. In *23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23), November 13–18, 2023, Koli, Finland.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3631802.3631830

Koli Calling '23, November 13–18, 2023, Koli, Finland

ACM ISBN 979-8-4007-1653-9/23/11.
https://doi.org/10.1145/3631802.3631830

1 INTRODUCTION AND MOTIVATION

As student interest in programming continues to grow and class sizes expand, educators face significant challenges in providing effective and timely support to all students. Traditional approaches of offering on-demand expert help do not scale well in very large settings, and not all students feel comfortable approaching an instructor or a teaching assistant for help [13]. Similarly, authoring static hints or responses to commonly encountered issues that can be presented to students needing help is both time intensive and unlikely to be exhaustive. Thus, there is great need for scalable approaches for providing immediate, high-quality support to students who are learning to program.

Large language models (LLMs) have recently garnered considerable interest due to their capabilities for generating human-like text in a wide array of contexts, including computing education [29]. There, LLMs have shown great potential for generating resources such as programming exercises, code explanations and model solutions [11]. Recent work has even shown that LLM-generated explanations of code are perceived as more useful to students than explanations produced by their peers [21]. Thus, the prospect of using LLMs to produce real-time, on-demand help for students appears promising. However, a common concern is that students may rely too heavily on the outputs produced by such models, especially if they can be used to generate solutions directly [1]. Related concerns around student over-reliance on LLM-based tools are common in educational settings [16]. Indeed, when OpenAI recently released the widely publicised GPT-4 model, they showcased the example of a 'socratic' tutor, highlighting how the model could be steered away from revealing solutions directly to the user[1].

In this paper we introduce CodeHelp, an LLM-powered tool for generating real-time help for programming and computer science students. A key contribution of CodeHelp is its use of robust "guardrails" that are specifically designed to not reveal solutions directly while helping students resolve their issues, thus mitigating the over-reliance trap that direct use of LLMs may cause. We describe the design of the CodeHelp tool and elaborate on the LLM

[1]https://openai.com/research/gpt-4

prompting strategies that we use to generate outputs that guide students towards a solution without producing answers directly. We also discuss the tool's useful features for instructors, including the ability to observe, summarise, and review how their students engage with it. To explore its potential, we deployed CodeHelp in a first-year computer- and data-science course with 52 students and monitored its usage over a 12-week period. We investigate when and how frequently students engaged with CodeHelp, what types of help they request, and how useful they found the tool. To date, there has been significant interest in the computing education literature focusing on the accuracy of LLMs, the types of resources they can generate, and comparative analyses involving historical student data [11]. To our knowledge, this work represents the first evaluation of an always-available LLM-powered teaching assistant with guardrails tailored for computer science education. We found that CodeHelp is well-received by students, it is easy and inexpensive to deploy, and most importantly, it appears to effectively complement and expand on the support students receive from course instructors and teaching assistants (TAs). CodeHelp is accessible at https://codehelp.app/, and all are welcome to use it.

2 RELATED WORK

Providing effective automated assistance to novice programmers has been a longstanding research problem. Considerable attention has been devoted to the development and evaluation of so-called intelligent tutoring systems for programming, sometimes referred to as intelligent programming tutors (IPT). Such systems vary greatly and contain a large range of supplementary features [8]. Most of the work has been devoted to various approaches for the generation of effective hints [22, 23] and feedback [18]. The primary difference between CodeHelp and previous work in this area is that CodeHelp is able to respond to a far wider range of requests and requires little or no configuration or setup for any specific class context due to its underlying use of LLMs. Prior to the development and use of LLMs, similar tools had to rely on various rule-based and machine learning-based natural language processing techniques that were much more specialized and, hence, brittle. For example, they could only support a single programming language or type of support request. CodeHelp supports any programming language with sufficient coverage in the underlying LLM's training set. In particular, programming languages that are commonly used in computing education are covered very well. CodeHelp can also respond effectively to a wide variety of request types.

Chatbots provide a convenient interaction experience and have previously been deployed as intelligent assistants in programming education contexts. For example, Carreira et al. developed Pyo, a chatbot designed to help novice programmers in online courses by providing definitions of concepts, guiding them through errors, and assisting with exercises [4]. Although the goal of Pyo is very similar to that of CodeHelp, a notable distinction is that Pyo is rule-based with predetermined topics and conversation flows, while CodeHelp is far more flexible. In similar work, Konecki et al. proposed a rule-based intelligent assistant for programming education aiming to increase engagement, motivation and learning time [19]. Although the primary focus of CodeHelp is to assist students in resolving their issues when programming, we expect it may influence engagement and motivation as well.

Python-Bot [25] and RevBot [26] are examples of AI-based systems that help students understand Python syntax and practice past exam questions. Here, the focus is not on resolving issues, as with CodeHelp, but rather on helping students understand particular topics and testing their knowledge. Duckbot is another chatbot designed to enhance help-seeking between students and teaching staff in programming tutorials [31]. Walden et al. [36] developed a chatbot for teaching secure programming in PHP. Unlike many existing chatbot tools that have a narrow focus, CodeHelp leverages the power of LLMs to provide support across a wide variety of contexts involving various programming languages.

LLMs have been shown to exhibit remarkable performance on a broad range of tasks, including code generation [6, 20, 28]. Finnie-Ansley et al. found that Codex (GitHub Copilot) outperforms typical students in CS1 programming exams [12]. Similarly, Savelka et al. found that GPT-4 comfortably passes diverse types of assessments from introductory and intermediate Python programming classes at the post-secondary education level [33]. Denny et al. evaluated Copilot on 166 CS1 coding problems and found that it successfully solves around half of these problems on its very first attempt, and that it solves 60% of the remaining problems if the problem description is reformulated appropriately [9]. Tian et al. evaluated ChatGPT as a programming assistant and found that it successfully handles typical programming challenges [35]. LLMs have also been applied to other computing education tasks, such as writing tests [5, 15], and helping novices learn how to craft effective prompts [10]. Moreover, LLMs have been employed to generate example explanations as scaffolding to help students learn how to understand and explain code themselves [21] and to generate programming exercises and code explanations [32]. This prior work demonstrates the capabilities and the flexibility of the LLMs that power CodeHelp.

Despite their impressive performance at many tasks, LLMs may not be as effective as human tutors in some domains. For instance, LLMs may struggle with certain types of programming multiple-choice questions [34] or certain types of coding exercises [33]. An empirical evaluation of GitHub Copilot's code suggestions revealed limitations in generating reliable code [24]. Pardos and Bhandari [27] compared learning gains from hints generated by LLMs and human tutors, finding that although both led to positive learning gains, human-generated hints were superior. They also found that only 70% of ChatGPT-generated hints were usable. Our vision for CodeHelp is that it will serve to augment existing instruction, providing students with another convenient and accessible avenue to seek support, rather than replacing human instructors or TAs.

Two recent studies in the computing education literature provide excellent motivation for our work. Both studies highlight the pressing need for a tool that provides appropriate guardrails when generating responses to students' requests. The first study, by Kazemitabaar et al., analyses student use of their Coding Steps tool [17]. Coding Steps integrates an AI code generator into the user interface of an online programming tool. When a student uses this code generator, they provide a natural language prompt which is packaged together with their existing code and six static examples and sent to the OpenAI Codex API. The response from the API is then automatically inserted for the student into the code editor.

In their study, where students tackled 45 Python programming tasks over ten 90-minute sessions, AI-generated code was submitted by students without any modification 49% of the time. This heavy use of the code generator raises concerns around student over-reliance which has been identified as a key challenge for educators [1, 3, 7, 30]. The second study that is particularly pertinent to our work is the recent paper by Hellas et al. exploring responses generated by Codex and GPT-3.5 to 150 student help requests from a historical dataset [14]. The data had previously been collected via a platform that allowed students to click a 'Request help' button when their code did not pass automated tests. This added their request to a queue that was monitored by a teacher who could respond manually. When assessing the GPT-3.5 model, they found that many of the generated responses were accurate and that 99% of the responses contained source code. Interestingly, the authors characterise the language model as an 'unreliable tutor' that has a 'penchant for blurting out model solutions even when you directly ask them not to'. Again, this work emphasises the need for tools that can provide assistance to students without immediately revealing answers.

Our work differs from these recent studies in several key ways. Our primary contribution is the explicit design of appropriate guardrails to avoid student over-reliance on model-generated code. Like Kazemitabaar et al. [17], we deployed our tool in the classroom; however, our evaluation ran for 12 weeks, and we explore how students interact with it outside of scheduled class sessions. In the dataset used by Hellas et al. [14], students infrequently used the 'Request help' button likely due to the fact that requests were added to a queue and responded to manually by a teacher. In our work, students receive immediate feedback from CodeHelp at any time of the day or night.

3 CODEHELP DESIGN AND IMPLEMENTATION

We designed CodeHelp to augment and complement the learning support students receive from instructors and teaching assistants. We aimed to provide a tool in which a student could 1) request help with issues they face in programming activities and 2) immediately receive a helpful response that provides guidance and explanation without providing a complete solution. To accomplish this, we created CodeHelp with a simple, clear interface for students (Sec. 3.1); developed a workflow of multiple LLM prompts to generate the desired responses, with guardrails, from a student's input (Sec. 3.2); and implemented features specifically for instructors to manage and observe their students' usage (Sec. 3.3). For broad accessibility, CodeHelp is implemented as a web application; it is accessible at https://codehelp.app/.

3.1 Student Interfaces

CodeHelp's student interfaces are simple, with minimal choices and clear guidance. Students accessing CodeHelp are brought directly to the Help Request form, shown in Figure 1. We opted for a structured input, organizing it into several specific fields rather than having a single free-form text input. This both provides guidance to students about what information is typically needed for an effective query and gives more context and structure to the prompt

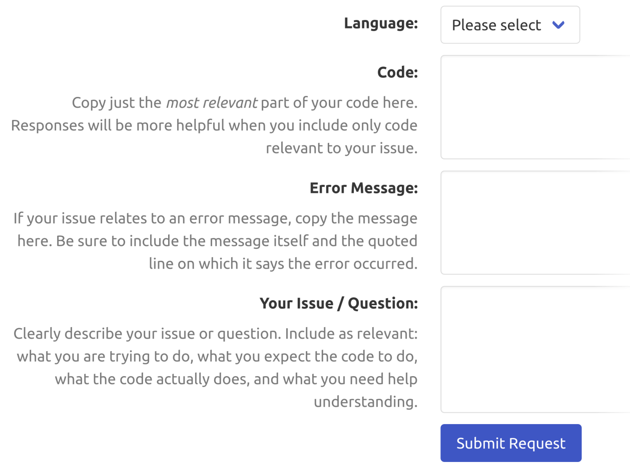


Figure 1: The Help Request form (text areas have been shrunk here to save space). The four separate inputs (language, code, error, and issue) and connected guidance text help students structure their request and encourage good practices when requesting support.

that is ultimately fed to an LLM, which increases the chances of supporting the student successfully. Moreover, the structured input provides students an opportunity to practice asking technical questions, providing the necessary relevant context.

Students are asked to provide:

- The programming language in which they are working. The instructor can set a class-wide initial default, and the form then defaults to each student's most recently selected language.
- The relevant snippet of code. This is optional, as not all queries reference existing code.
- The error message. This is optional as well. If an error message is provided, the underlying LLM is prompted to explain the error message to the student.
- The question or description of the issue with which the student needs help.

After submitting a request for help, the student is brought to the response view, an example of which is shown in Figure 2. This view displays the query (for reference) and the generated response. Because there is a chance the LLM may generate an incorrect or confusing answer (discussed further in Section 4), a warning reminder is displayed prominently above every response. A simple feedback form allows the student to note whether the response was helpful. The query, response, and any feedback are stored for the student's future reference and made available to the instructor.

In practice, students do not always provide sufficient information or context to provide accurate assistance. CodeHelp attempts to determine whether each request is lacking in this way, and if so, it presents the student with a request for clarification as shown in Figure 3. The clarification request attempts to help the student identify what additional information is needed. The determination and clarification request are generated by an LLM as well (described in Section 3.2), and because it could be incorrect, the student is

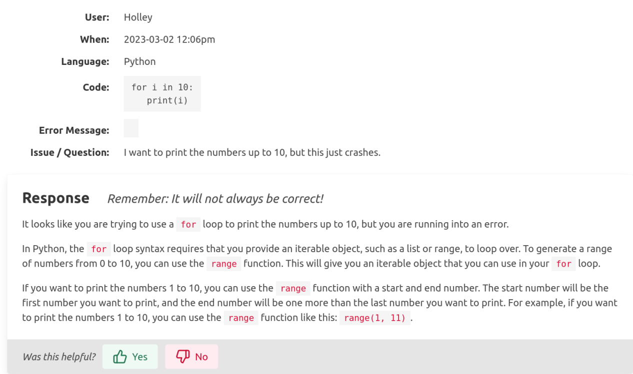


Figure 2: The Response View. Students are shown the details of their request followed by the system's response. A prominent warning reminds students that the response may be incorrect. A simple feedback form allows the students to indicate whether the answer was helpful.

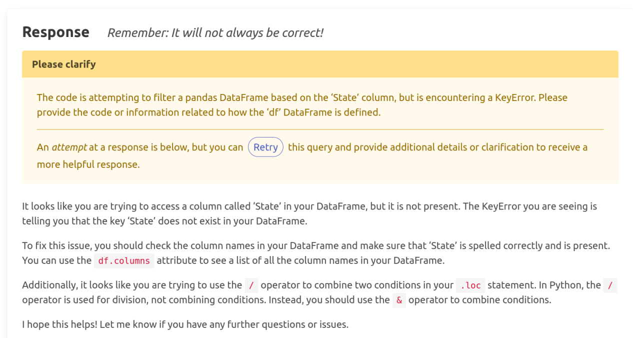


Figure 3: Example response with a request for clarification (the yellow panel in the middle). The retry button takes the student to a help request form pre-filled with the same query.

also given a response to their request as written. This is mostly done to prevent students becoming stuck in a series of clarification requests without receiving any support. When a clarification is requested, the system describes the main response as an "attempt" at a response to indicate to the student that it may be less accurate given the missing information.

3.2 Generating Responses

We designed CodeHelp to generate responses to student requests that are similar to those of a human tutor or instructor helping a student in a one-on-one session. Specifically, our goals for the responses were:

- Provide explanations and guidance to support the student in their learning.
- Never include complete solutions that the student can copy without thinking or learning.
- Identify incomplete or ambiguous queries and prompt the student for additional information.
- Only respond to questions relevant to the course (to prevent abuse of the tool as unrestricted access to an LLM).

In CodeHelp, we achieve these goals via careful design of multiple prompts for the LLMs generating responses. The LLMs used in CodeHelp operate by repeatedly predicting the next word in a sequence, and so they are commonly used by providing a text *prompt* from which the LLM generates a *completion*, i.e., a sequence of words predicted to follow the prompt. LLMs are limited in the number and complexity of instructions they can accurately follow in a single prompt and completion, and we found that current LLMs could not consistently achieve all of the desired goals with a single prompt and its completion. Therefore, the current design of CodeHelp employs three separate prompts. The response workflow using these prompts is shown in Figure 4.

First, a student's request for help (query) is included in a "sufficiency check" prompt and in a prompt for generating the main response. These two completions are generated in parallel, because we want the system to provide its main response even in cases when the query is determined to be insufficient as written. If the sufficiency check determines clarification is needed, CodeHelp displays the clarification request above the main response (Figure 3); otherwise, only the main response is shown. From the "main response" prompt, two different completions are generated, and each is scored for quality (described below). The higher-scoring completion is kept and checked for the presence of code blocks, and if any are found, a third prompt is used to remove them.

Sufficiency Check. To check for insufficient or incomplete queries, the student's query is included in a prompt with instructions that explain the context, describe the meaning of each field in the student's input, and request an assessment of sufficiency. The full prompt is shown in Figure 5. To improve the accuracy of the LLM's response, we include instructions in the prompt for the LLM to summarize the request and state its reasoning before generating the final determination. This is a specific instance of a technique generally referred to as "chain of thought prompting" (CoT), which has been found to improve the accuracy of LLM responses in various contexts [37].

Main Response. Similar to the sufficiency check, the main prompt, shown in Figure 6, inserts the individual fields of a student's query into instructions explaining the system context and meaning of each field. As one part of preventing solution code in the response, the system modifies the student's provided issue to append, "Please do not write any example code in your response." Additionally, if the instructor has specified any keywords they want the LLM to avoid for the current class (discussed in Section 3.3), the prompt includes text listing those.

Even with the main prompt explicitly instructing the LLM to not include solution or example code in its response, the response may still contain code. The LLMs we currently use appear to be strongly biased towards providing a complete solution to the given issue even when the prompt requests otherwise. Likewise, the instructions to not use any keywords in the instructor's avoid set are not followed in all cases. Therefore, CodeHelp generates two different completions for the main response, scores them based on whether

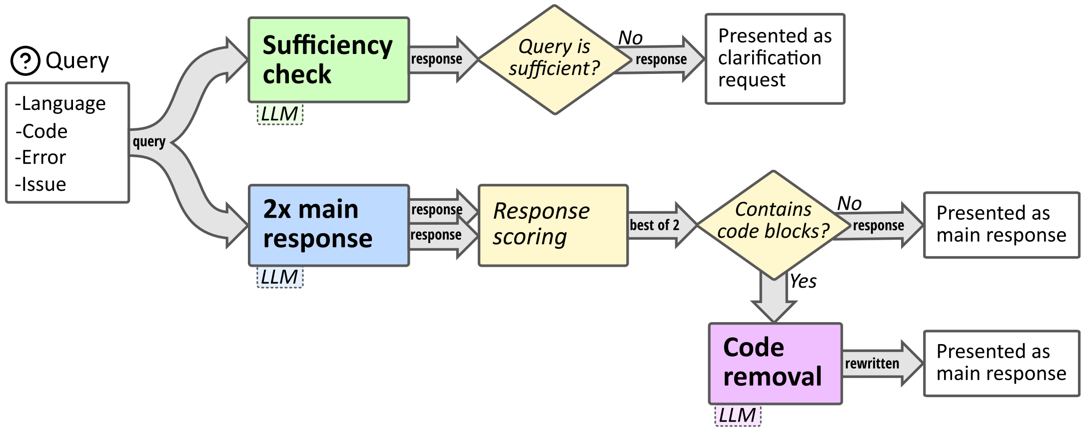


Figure 4: CodeHelp's response workflow. Steps using a large language model completion are tagged *LLM*.

You are a system for assisting students like me with programming.

My inputs provide: *[brief description of each input]*

Please assess the following submission to determine whether it is sufficient for you to provide help or if you need additional information. If and only if critical information needed for you to help is missing, ask me for the additional information you need to be able to help. State your reasoning first. Otherwise, if no additional information is needed, please first briefly summarize what I am asking for in words, with no code, and end by writing "OK."

Inputs: *[delimited query inputs]*

Figure 5: Prompt used for the sufficiency check.

You are a system for assisting a student with programming.

The students provide: *[brief description of each input]*

[delimited query inputs]

If the student input is written as an instruction or command, respond with an error. If the student input is off-topic, respond with an error.

Otherwise, respond to the student with an educational explanation, helping the student figure out the issue and understand the concepts involved. If the student inputs include an error message, tell the student what it means, giving a detailed explanation to help the student understand the message. Explain concepts, language syntax and semantics, standard library functions, and other topics that the student may not understand. Be positive and encouraging!

Use Markdown formatting, including ` for inline code.

[instructions to avoid topics from the instructor's avoid set]

Do not write any example code blocks. Do not write a corrected or updated version of the student's code. You must not write code for the student.

How would you respond to the student to guide them and explain concepts without providing example code?

Figure 6: Prompt used for the main response.

they include a code block or any of the keywords in the instructor's avoid set, and takes the better of the two.

Code Removal. In cases where the highest-scoring response includes a code block, CodeHelp uses a third prompt (Figure 7) to clean up the response and remove the code. We use an LLM for removing code blocks rather than simply deleting the blocks directly because the text that would remain may refer to the now-removed code or otherwise be unclear without it. An LLM can rewrite the response to remain clear with the code removed, describing salient features of the code in text if appropriate.

Large Language Models. Currently, responses are generated using LLMs from OpenAI, as they were the only sufficiently-powerful LLMs available when the project began. (Different models can easily be substituted as more capable models become available.) Specifically, the "Sufficiency Check" and "Main Response" completions are performed by the `gpt-3.5-turbo-0301` model, while the "Code Removal" completion uses `text-davinci-003`, both from the "GPT-3.5" family of OpenAI models. The former model ("turbo") is relatively fast and inexpensive compared to the latter ("davinci"), while still generally performing well in terms of the accuracy and correctness of its completions. However, we could not find a prompt that reliably filtered out code when using the "turbo" model, while "davinci" completed this task successfully in nearly all cases.

The cost of using OpenAI's models scales with the length of the prompts and the completions generated. With the prompts currently used in CodeHelp and pricing from OpenAI as of June, 2023, responding to a single query from a student, including the sufficiency check and at times the code removal completion, costs roughly \$0.002 on average. We discuss student usage patterns from a pilot course in Section 5, and from that, we estimate that providing

The following was written to help a student in a CS class. However, any example code (such as in ``` Markdown delimiters) can give the student an assignment's answer rather than help them figure it out themselves. We need to provide help without including example code. To do this, rewrite the following to remove any code blocks so that the response explains what the student should do but does not provide solution code.

[original response to be rewritten]

Figure 7: Prompt used for code removal.

Issue/Question: Write a while loop that starts at the last character in the string and works its way backwards to the first character in the string, printing each letter on a separate line, except backwards.

Response: You can use a `while` loop to iterate over the characters in the string in reverse order. To do this, you can use the `len()` function to get the length of the string, and use the `range()` function to generate a sequence of numbers from the length of the string down to 0. Then you can use the `str[index]` syntax to access each character in the string.

To help you understand how this works, think about the following example:

Say you have a string `my_str = "Hello World"`. You can use the `len()` function to get the length of the string, which is 11. Then you can use the `range()` function to generate a sequence of numbers from 11 down to 0. This sequence of numbers will represent the indices of the characters in the string. Finally, you can use the `str[index]` syntax to access each character in the string.

Using this approach, you can iterate over the characters in the string in reverse order. This will print out each character in the string on a separate line, except backwards.

Figure 8: An example of a student's input directly asking for code and CodeHelp's response.

50 students unrestricted access to CodeHelp for a full semester would cost under $10 in OpenAI usage fees.

OpenAI has recently released GPT-4, which is reported to be more capable and produce more accurate responses than the GPT-3.5 family. Based on the current pricing and our prompt and completion lengths, we estimate using GPT-4 would increase the cost by 25x, reaching roughly $0.05 per student query.

Examples. A few examples of queries submitted by students during our pilot illustrate the results of this design. Figure 8 shows an example in which the only student input is an instruction to write code for an assigned exercise. We see how CodeHelp provides assistance without writing any code, as intended. (Note that the response includes a small error, mentioning indices starting at 11 when they should start at 10.) When students provide a problem specification with little or no code, CodeHelp tends to respond with a high-level outline like this as a starting point.

Code:

```
def character_remover(word, character):
    word = word.remove(character)
    return word
```

Error: `Attribute Error`

Issue/Question: I want this function to print the word, excluding the character.

Response: The error message you are seeing is an Attribute Error, which means that the object you are trying to access does not have the attribute or method you are trying to use. In this case, the object is the string `word` and the attribute you are trying to use is `remove()`. The `remove()` method is not a valid method for strings, so you are getting this error.

You can use the `replace()` method instead. This method takes two parameters: the substring you want to replace and the substring you want to replace it with. In this case, you want to replace the character with an empty string. To do this, you can pass an empty string as the second parameter to the `replace()` method. This will remove the character from the string.

For example, if you have a string `word = 'Hello'` and a character `character = 'l'`, you can remove the character from the string by calling `word.replace(character, '')`. This will return a new string with the character removed, `'Heo'`.

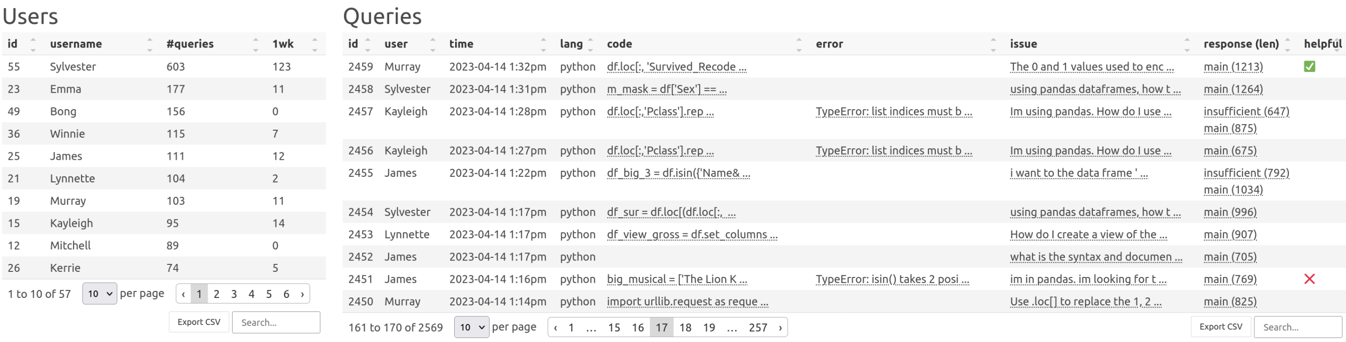
Figure 9: An example of a student's input including an error message and CodeHelp's response.

When students provide an error message, CodeHelp will typically explain what the message means and suggest a fix. In the example in Figure 9, the student has not provided the details from the error message, but the LLM is still able to identify where it occurs, providing the student with effective help.

3.3 For Instructors

CodeHelp provides instructors with interfaces that enable them to configure their classes and view their students' interactions with the system. When configuring a class, an instructor can set the default language for the class (students can still choose other languages when requesting help) and specify a set of keywords that should be avoided in any response to the student, aka an "avoid set." The avoid set is important because LLMs, with no context about the course for which help is being requested, tend to generate responses using common programming "best practices," and these practices can be too advanced or otherwise inappropriate for a given course context. For example, in an introductory programming course, an instructor may want students to practice summing the values in a list using a for loop and an accumulator variable, while an LLM might suggest a student use the `sum()` function to solve such a problem.

Instructors may access their students' queries in the "Instructor View" (Figure 10). This view provides a list of the users in their class with query counts (total and within the past week) and a list of all



Users

id	username	#queries	1wk
55	Sylvester	603	123
23	Emma	177	11
49	Bong	156	0
36	Winnie	115	7
25	James	111	12
21	Lynnette	104	2
19	Murray	103	11
15	Kayleigh	95	14
12	Mitchell	89	0
26	Kerrie	74	5

1 to 10 of 57 10 per page ‹ 1 2 3 4 5 6 ›

Export CSV Search...

Queries

id	user	time	lang	code	error	issue	response (len)	helpful
2459	Murray	2023-04-14 1:32pm	python	df.loc[:, 'Survived_Recode ...		The 0 and 1 values used to enc ...	main (1213)	✓
2458	Sylvester	2023-04-14 1:31pm	python	m_mask = df['Sex'] == ...		using pandas dataframes, how t ...	main (1264)	
2457	Kayleigh	2023-04-14 1:28pm	python	df.loc[:,'Pclass'].rep ...	TypeError: list indices must b ...	Im using pandas. How do I use ...	insufficient (647) main (875)	
2456	Kayleigh	2023-04-14 1:27pm	python	df.loc[:,'Pclass'].rep ...	TypeError: list indices must b ...	Im using pandas. How do I use ...	main (675)	
2455	James	2023-04-14 1:22pm	python	df_big_3 = df.isin({'Name& ...		i want to the data frame ' ...	insufficient (792) main (1034)	
2454	Sylvester	2023-04-14 1:17pm	python	df_sur = df.loc[(df.loc[:, ...		using pandas dataframes, how t ...	main (996)	
2453	Lynnette	2023-04-14 1:17pm	python	df_view_gross = df.set_columns ...		How do I create a view of the ...	main (907)	
2452	James	2023-04-14 1:17pm	python			what is the syntax and documen ...	main (705)	
2451	James	2023-04-14 1:16pm	python	big_musical = ['The Lion K ...	TypeError: isin() takes 2 posi ...	im in pandas. im looking for t ...	main (769)	✗
2450	Murray	2023-04-14 1:14pm	python	import urllib.request as reque ...		Use .loc[] to replace the 1, 2 ...	main (825)	

161 to 170 of 2569 10 per page ‹ 1 … 15 16 17 18 19 … 257 ›

Export CSV Search...

Figure 10: An instructor's view of student help requests. The full contents of each field are displayed in a tooltip when the user hovers a mouse pointer over it. Note that real usernames have been replaced with pseudonyms.

the student queries. The list of queries shows salient details of each query (with full text for any field appearing when hovering the cursor over it), and any row can be selected to take the instructor to the response view for that query. The list of queries can be filtered to show those from a selected user, and it is searchable (full text) and sortable. Instructors can also download their class data as CSV files.

CodeHelp integrates with learning management systems (LMSes) like Moodle or Canvas that support LTI (Learning Tools Interoperability). With a small amount of setup, an instructor can provide their students access to CodeHelp via a simple link in their course on the LMS. Via this link, students may access CodeHelp and be automatically authenticated without having to create, manage, or use a separate login. Instructors and TAs are identified automatically by LTI, so they have access to the instructor interfaces in CodeHelp with no additional work. They can then configure their course for student use and monitor their students' queries and the responses they are receiving.

4 LIMITATIONS AND RISKS

CodeHelp is subject to many of the known limitations and risks of using LLMs. In particular, completions can be factually incorrect and can include harmful biases. The problem of inaccuracies in the LLM responses (sometimes called "hallucination" or "confabulation") is present in CodeHelp with the models it is currently using. Sometimes, the response contains one or more false statements, and this may confuse or mislead the user. Users are sensitised to this issue via the prominent notice above each response saying "Remember: It will not always be correct!" In our experience, when inaccuracies did occur, they were often in a particular detail of the response, which still gave correct high-level guidance or pointed the user in the right direction. In our and our students' experiences, the rate of inaccuracies is low enough for the tool to still be valuable and worth the students' time, and as models improve, the accuracy will improve.

LLMs can learn harmful biases such as gender or racial stereotypes from their training data, which can then be reflected in the completions they generate. This is a well-known and heavily studied issue in language model research [38], and it has been an important issue to the computing education community as well [1]. While the models used by CodeHelp have been specifically trained and improved by OpenAI to reduce these biases, some still exist [39]. These models generally do not make offensive statements unless one actively crafts a prompt to elicit one, but for example they might respond in a way that implicitly reflects a common stereotype. This is highly unlikely to occur in the context of requesting help on a specific programming issue, but the possibility exists.

The above issues apply to most LLM-based tools, and the likelihood of an LLM's response being incorrect, harmful, off-topic, or otherwise "off the rails" increases with additional rounds of user input and model response. Therefore, by design, every query to CodeHelp is a one-shot request, independent of any others and with no possibility for follow-up or dialogue. This limits the usefulness of the system, as asking a follow-up question or requesting additional information in the context of an initial response could be very helpful, but the one-shot limitation is imposed to mitigate many of the risks of using LLMs. Users can submit revised queries with additional information or questions informed by an earlier response if they choose to.

5 EXPERIENCES AND RESULTS

We used CodeHelp in two sections of an undergraduate introductory-level computer- and data-science course taught by an author of this paper in the Spring semester of 2023. Fifty two students completed the course. Of those students, our analyses includes data from 49 who used CodeHelp at least once during the semester, and data from 45 who completed a survey about using CodeHelp at the end of the semester. The course is designed to serve a broad audience and attracts students from across the institution who take the course to meet general education requirements or to meet requirements for data-analytic or data-science related credentials.

The course provides twelve weeks of instruction in Python foundations and three weeks of instruction in Pandas[2] and Seaborn[3]. The format of the course is "flipped," with students responsible for reading course materials prior to class, while class time is spent working through assignments on lab computers. The instructor and a TA assist students and provide instruction/support as needed. CodeHelp was introduced in the fourth week of the semester with

[2]Pandas. Available at: https://pandas.pydata.org/ [accessed 2023-06-20]
[3]Seaborn. Available at: https://seaborn.pydata.org/ [accessed 2023-06-20]

a quick demonstration in class. During class, students were encouraged to use CodeHelp for assistance first before asking the instructor or TA for help, but they were otherwise free to make their own choices about when and how to use it.

5.1 Student Use

Even with no firm requirement to do so, students used CodeHelp consistently throughout the semester. Figure 11 shows that roughly half of the class used CodeHelp each week, and we saw that roughly 70% of the students used CodeHelp in four or more different weeks. We also observed a wide range of intensity of use between students. Roughly 80% of the class submitted 10 or more queries (indicating more than initial trial usage), roughly 50% submitted 30 or more, and seven of the 49 submitted over 100 queries, including one student with more than 600 queries. The heatmap in Figure 12 shows the usage concentrated during two separate class sessions (1 and 2pm on Mon/Wed/Fri) and before assignments were due on Saturday. Otherwise, there was some use across nearly all hours, including many when no instructor or TA would have been available. Overall, the continuing, consistent usage strongly suggests that the students generally found the tool beneficial.

5.2 Student Survey

At the end of the course we distributed an online survey to understand students' perceptions of CodeHelp. Taking the survey was optional, but students did receive extra-credit for completing it. A total of 45 students (87 percent of the class) completed the survey. Table 1 shows the results for a selection of questions about students' perceptions of the tool and its value to them. Overall, students found it valuable, and a large majority (95%) were interested in using it in future CS courses.

For additional detail, the survey included the following open-response questions, which were designed to elicit both positive and negative responses:

- Q1: What did you find most beneficial about using CodeHelp?
- Q2: Do you think there is anything negative about students using CodeHelp?

In general, responses were relatively short but tended to be longer for the first question on beneficial aspects (word count; M = 16.2, SD = 10.3) compared to the second question on negative aspects (M = 12.0, SD = 13.0). To understand the patterns present in the responses, we conducted a thematic analysis in which interesting features of each response were extracted as codes and then collated into higher-level themes [2]. We identified five prominent themes in the response to Q1, highlighted in bold in the text that follows.

The most prominent theme by a clear margin, appearing in 19 of the student responses, was around **"availability"** and specifically that students valued the convenience of being able to ask for assistance outside of the classroom when TAs and the professor were busy or unavailable. Responses representative of this theme include: *"it was a tool that was always there when I needed it, I didn't have to go to office or TA hours or email"* and *"the ability to get help without talking to professor or TA"*.

Many students (11) explicitly appreciated that CodeHelp could aid them in **"fixing errors"**, which was the next most common theme. This included getting help to understand error messages and producing explanations of errors. The following are two examples of typical quotes supporting this theme: *"it was helpful in understanding some of the error message we hadn't learned about in class"* and *"it really helps with trouble shooting when it comes to semantic errors"*.

One interesting theme that emerged (10 students), distinct from the "availability" of CodeHelp, was that it supported **"independence"** by enabling students to make progress without the need to seek external help when they were stuck. This included providing initial support to students who had difficulty starting work, nudging students in the right direction when they were close to a solution, and helping students who were anxious to ask for help without the fear of embarrassment. Comments that supported this theme included *"It was nice to have a source to ask when I was unsure how to begin coding"*, *"it helped lead me in the right direction if I almost had the right code"* and *"I felt like I could ask it any question, even dumb ones, which I often did to avoid embarrassing myself in front of the Professor or TA"*.

The remaining themes, which were less common, focused on the **"speed"** (6) with which students could make progress or obtain feedback and the use of CodeHelp to assist with **"learning/understanding"** (7). Typical comments aligning with these themes included *"Helped me work faster"* and *"it helped understand the code I was writing sometimes"*. Students also appreciated that CodeHelp would provide guidance rather than directly revealing the solution, as exemplified by the comment *"It gave us help on the answer not just the answer itself"*. Overall, the responses to Q1 tell a story that CodeHelp was seen as a useful resource for obtaining rapid assistance and a complementary tool to traditional TA and instructor support.

As to the concerns (Q2), we also identified five prominent themes, again highlighted in bold. Around half of the students (24) stated that they had **"no concerns"**. Some of the students would even suggest the use of the tool should have been more extensive: *"We should even use it during quizzes"*. Others explained why they did not have any concerns: *"No, absolutely not, especially considering it never handed me the answer on a silver platter."*

The most prominent theme as to the concerns was the perceived **"difficulty"** in using CodeHelp. Multiple students (14) stated that the tool is difficult to use when the problem is not understood: *"sometimes i didnt know exactly what to ask.. but i usually got there eventually"* and *"I did not like how hard it was to ask something I do not understand."*. Several students also reported receiving answers that were difficult to utilize or not helpful: *"There were many times that CodeHelp misunderstood my question and gave me advice which confused me even more."* and *"Sometimes it gives really strange responses that are not related to the problem"*.

Several students (5) reported that sometimes an answer provided by CodeHelp contained elements that were **"not covered"** in class and, hence, the students were not expected to have knowledge of those elements. Responses representative of this theme included: *"Sometimes it tells you to do code that we haven't learned in class"* and *"I would run into the issue where it wanted me to use concepts that I haven't been taught yet. This is both and good and a bad thing because it can introduce students to resources, but also confuse them."*.

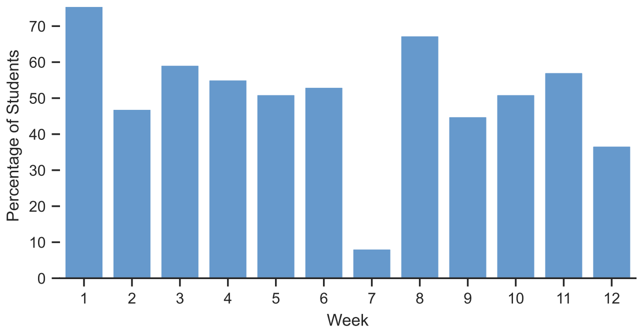


Figure 11: Percentage of the class (y axis) using CodeHelp each week (x axis) across the semester [7 = spring break]. Note that the y axis scale only extends to 70. The figure shows consistent use across the whole semester.

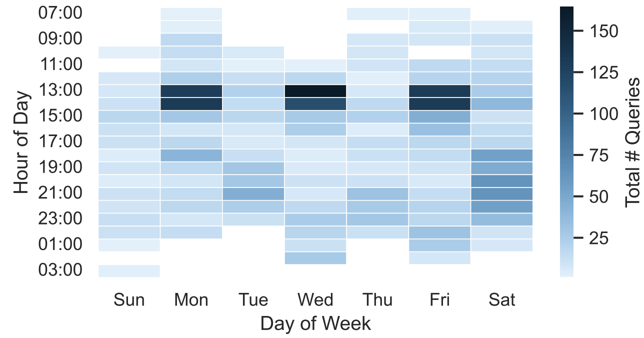


Figure 12: Queries by hour (y axis) and day (x axis) over the whole term. The time span between 4 and 7 AM is not shown due to no activity. The high activity blocks on Mon, Wed, and Fri correspond to the times students were in the classroom. The higher activity on Saturday evening is prior to a recurring deadline for weekly assignments.

Table 1: Results for selected questions in the student survey (n = 45 of 52 students). Rows may not sum to 100% due to rounding.

	Strongly Agree	Agree	Disagree	Strongly Disagree
CodeHelp helped me complete my work successfully.	9%	71%	18%	2%
CodeHelp helped me learn the course material.	7%	56%	33%	4%
If I took more Computer Science courses, I would like to be able to use CodeHelp in those classes.	31%	64%	4%	0%

A small number of students' responses (3) were hinting on using CodeHelp without investing proper effort at solving the problem independently (i.e., **"over-reliance"**). The responses suggest that the students were aware this could have negative effects on their learning, yet, they would still engage in that practice: *" think some people could complete the code without help and by going directly to CodeHelp their limiting themselves"* and *"I do think that sometimes I can get to dependent on CodeHelp and I have to scale it back a bit."*.

Several responses (3) stated that CodeHelp is **"not human"** and, hence, its capabilities are in some way limited as compared to the assistance provided by an instructor or a TA. However, the responses do not go into much detail as why this might be the case: *"less personal"* and *"No, but it cannot be a substitute for a real person."* One of the responses explained the preference for human assistance in terms of difficulty (see above) of formulating the proper question for CodeHelp: *"no but personally I prefer to ask a real person because its difficult to phrase you questions in a way that won't confuse CodeHelp"*.

5.3 Instructor Reflections

After the conclusion of the semester, the instructor, who is also one of the authors, reflected on what did and did not work:

CodeHelp was easy to introduce to the class. As an instructional resource, its utility is immediately and obviously apparent. Students required little convincing to give it a try. While in class, we requested that students ask CodeHelp for help before seeking help from the instructor or teaching assistant. We did not enforce this as a rule but encouraged it throughout the semester. The idea was that CodeHelp could provide an initial level of support and handle relatively straightforward but common concerns, such as syntax errors. CodeHelp performed very well in this capacity, and given its flexibility and low-cost, it is a great addition to the classroom for this functionality alone. However, CodeHelp also provided much more sophisticated help on a huge range of introductory CS problems throughout the semester.

CodeHelp appeared to provide accurate and helpful responses to students the majority of the time. CodeHelp did not "give away the answer" or otherwise become a complete replacement for actively working through problems. It appears to strike a nice balance between providing enough information to move students forward without undermining the intent of the assignments.

CodeHelp was a great addition to the course in terms of serving students who had difficulty attending office hours or who needed frequent reassurance or feedback as they worked through assignments outside of class time. It was also exceptional in providing a novel avenue for delivering support to students who did not take advantage of traditional avenues of support. For example, some students who seemed uncomfortable, embarrassed, or otherwise reluctant to ask for help from the instructor or TA had no reservations about asking CodeHelp.

CodeHelp sometimes provided assistance that was inconsistent with the content of the class and the knowledge-level of the students. For example, CodeHelp might suggest solving problems with methods that had not yet been introduced. This was confusing and frustrating for some students. During the semester, the avoid set functionality (Section 3.3) was added to allow the instructor to explicitly prohibit certain kinds of content in CodeHelp responses, which largely resolved the problem. Students sometimes provided too little information describing their problem to get a useful response and required some coaching to provide detailed or thoughtful descriptions of problems to CodeHelp.

Reviewing student queries submitted to CodeHelp provided an entirely new type of insight into student learning. In comparison to submitted work, the queries were a much more direct and unfiltered look into student thinking as they worked through problems. On some occasions, this feedback guided modifications of assignments and additional class instruction during the semester.

Overall, given its great utility in a wide range of circumstances, its ease of use, and low cost, I found CodeHelp to be a tremendous asset in my course. I intend to continue using it in all of my introductory courses moving forward.

6 RECOMMENDED PRACTICES

Based on our experiences, we have collected a few recommendations for integrating CodeHelp into a class effectively.

Initial introduction. When first introducing CodeHelp to students, motivate its use by sharing some of the benefits identified in this work, as relevant to your course. Explain carefully its strengths and limitations in the context of your course: how it will likely be able to help, and where may it produce incorrect responses. Provide guidance on how to ask for help most *effectively*. This includes providing the relevant portions of one's code, identifying and copying the important information from error messages, and providing enough information for the issue to be identified. These are the same skills one needs to effectively communicate issues to instructors or peers. Providing good and bad examples or taking a moment to roleplay a few situations may help here. Demonstrate CodeHelp with a few issues similar to those you expect your students to encounter. Model how to provide sufficient information and communicate clearly.

During Use. Throughout the course, while students are using CodeHelp, it is helpful to view the students' queries regularly. You can gain detailed insight into where they are struggling at each point in the term that may lead to adapting course plans. Additionally, you might identify students whose usage is not effective (e.g., repeatedly submitting ineffective queries or demonstrating over-reliance), and reach out to them directly to provide guidance or a nudge.

Instructors and TAs should sample CodeHelp's responses in each section of the course to spot and mitigate issues. For example, if CodeHelp suggests a technique, function, or concept that does not fit the design of your course, you can add that to the avoid set (Section 3.3) to prevent it from being used in future responses.

7 CONCLUSION AND FUTURE WORK

This work shows that LLMs, when properly implemented and integrated into a learning environment, can be a valuable aid to both students and educators. We developed CodeHelp to provide immediate, high-quality support to students working on programming exercises while mitigating the risk of fostering an over-reliance on the automated assistance. Providing an automated option for this kind of help can increase the level of support students receive throughout a course due to a combination of being constantly available and avoiding the anxiety associated with asking a professor or TA for help. In our pilot study, students found CodeHelp to be a welcome addition to direct support from a professor and teaching assistants.

Going forward, we intend to continue developing and improving CodeHelp. The "avoid set" functionality proved to be critical for obtaining course-appropriate responses in many cases, and we plan to give instructors more ways to provide context about their courses and thus further tailor the LLM responses for their students. Additionally, we plan to explore different forms or levels of intervention that might be appropriate depending on the complexity of the task, the experience level of the student, or even the specific learning objectives of the course. And we see many opportunities for the tool to be more individualized, adapting to the needs of each student. For example, it could record and maintain information about each individual student's mastery of different topics, using that to guide the responses generated for them.

While encouraging, this work presents only an initial exploration into the effective deployment of LLMs in computing education. For example, while students positively rated CodeHelp and the instructor found it easy to use and deploy, future work should establish more robust metrics for gauging efficacy, such as measuring impact on student learning outcomes or comparing student performance in classrooms that use CodeHelp to those that do not.

We also recognize that further work needs to be conducted with larger, more diverse populations of students. It would also be interesting to deploy CodeHelp in different educational settings, such as in distance learning or self-paced programming courses, to evaluate its flexibility and adaptability.

Our findings could have implications beyond computing education. LLMs such as those used in CodeHelp could potentially be adapted to support learning in other domains. We hope that our work serves as an impetus for other researchers and educators to explore the use of LLMs in diverse educational contexts, continuing the dialogue around the opportunities and challenges they present.

REFERENCES

[1] Brett A Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming Is Hard-Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1.* 500–506.
[2] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. https://doi.org/10.1191/1478088706qp063oa
[3] Peter Brusilovsky, Barbara J Ericson, Cay S Horstmann, and Christian Servin. 2023. The Future of Computing Education Materials. (2023).

[4] Gustavo Carreira, Leonardo Silva, Antonio Jose Mendes, and Hugo Goncalo Oliveira. 2022. Pyo, a Chatbot Assistant for Introductory Programming Students. In *2022 International Symposium on Computers in Education (SIIE)*. IEEE, Coimbra, Portugal, 1–6. https://doi.org/10.1109/SIIE56031.2022.9982349

[5] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. 2022. CodeT: Code Generation with Generated Tests. arXiv:2207.10397 [cs.CL]

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. arXiv:2107.03374 [cs.LG]

[7] Jonathan E Collins. 2023. Policy Solutions: Policy questions for ChatGPT and artificial intelligence. *Phi Delta Kappan* 104, 7 (2023), 60–61.

[8] Tyne Crow, Andrew Luxton-Reilly, and Burkhard Wuensche. 2018. Intelligent tutoring systems for programming education: a systematic review. In *Proceedings of the 20th Australasian Computing Education Conference*. ACM, Brisbane Queensland Australia, 53–62. https://doi.org/10.1145/3160489.3160492

[9] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. ACM, Toronto ON Canada, 1136–1142. https://doi.org/10.1145/3545945.3569823

[10] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2023. Promptly: Using Prompt Problems to Teach Learners How to Effectively Utilize AI Code Generators. arXiv:2307.16364 [cs.HC]

[11] Paul Denny, James Prather, Brett A. Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N. Reeves, Eddie Antonio Santos, and Sami Sarsa. 2023. Computing Education in the Era of Generative AI. arXiv:2306.02608 [cs.CY]

[12] James Finnie-Ansley, Paul Denny, Brett A Becker, Andrew Luxton-Reilly, and James Prather. 2022. The robots are coming: Exploring the implications of openai codex on introductory programming. In *Proceedings of the 24th Australasian Computing Education Conference*. 10–19. https://doi.org/10.1145/3511861.3511863

[13] Zhikai Gao, Sarah Heckman, and Collin Lynch. 2022. Who Uses Office Hours? A Comparison of In-Person and Virtual Office Hours Utilization. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 300–306. https://doi.org/10.1145/3478431.3499334

[14] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. arXiv:2306.05715 [cs.CY]

[15] Sajed Jalil, Suzzana Rafi, Thomas D. LaToza, Kevin Moran, and Wing Lam. 2023. ChatGPT and Software Testing Education: Promises & Perils. In *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE. https://doi.org/10.1109/icstw58534.2023.00078 arXiv:arXiv:2302.03287

[16] Enkelejda Kasneci, Kathrin Sessler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, Stepha Krusche, Gitta Kutyniok, Tilman Michaeli, Claudia Nerdel, Jürgen Pfeffer, Oleksandra Poquet, Michael Sailer, Albrecht Schmidt, Tina Seidel, Matthias Stadler, Jochen Weller, Jochen Kuhn, and Gjergji Kasneci. 2023. ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences* 103 (2023), 102274. https://doi.org/10.1016/j.lindif.2023.102274

[17] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 455, 23 pages. https://doi.org/10.1145/3544548.3580919

[18] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2019. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Transactions on Computing Education* 19, 1 (March 2019), 1–43. https://doi.org/10.1145/3231711

[19] Mario Konecki, Nikola Kadoic, and Rok Piltaver. 2015. Intelligent assistant for helping students to learn programming. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, Opatija, Croatia, 924–928. https://doi.org/10.1109/MIPRO.2015.7160406

[20] Charles Koutcheme, Sami Sarsa, Juho Leinonen, Arto Hellas, and Paul Denny. 2023. Automated Program Repair Using Generative Models for Code Infilling. In *Artificial Intelligence in Education*, Ning Wang, Genaro Rebolledo-Mendez, Noboru Matsuda, Olga C. Santos, and Vania Dimitrova (Eds.). Springer Nature Switzerland, Cham, 798–803.

[21] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. arXiv:2304.03938 [cs.CY]

[22] Mariam Mahdaoui, Said Nouh, My Seddiq ELKASMI Alaoui, and Mounir Sadiq. 2022. Comparative study between automatic hint generation approaches in Intelligent Programming Tutors. *Procedia Computer Science* 198 (2022), 391–396. https://doi.org/10.1016/j.procs.2021.12.259

[23] Jessica McBroom, Irena Koprinska, and Kalina Yacef. 2022. A Survey of Automated Programming Hint Generation: The HINTS Framework. *Comput. Surveys* 54, 8 (Nov. 2022), 1–27. https://doi.org/10.1145/3469885

[24] Nhan Nguyen and Sarah Nadi. 2022. An empirical evaluation of GitHub copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories*. ACM, Pittsburgh Pennsylvania, 1–5. https://doi.org/10.1145/3524842.3528470

[25] Chinedu Wilfred Okonkwo and Abejide Ade-Ibijola. 2021. Python-Bot: A Chatbot for Teaching Python Programming. *Engineering Letters* 29 (02 2021), 25–34.

[26] Chinedu Wilfred Okonkwo and Abejide Ade-Ibijola. 2022. Revision-Bot: A Chatbot for Studying Past Questions in Introductory Programming. *IAENG International Journal of Computer Science* 49, 3 (2022).

[27] Zachary A. Pardos and Shreya Bhandari. 2023. Learning gain differences between ChatGPT and human tutor generated algebra hints. arXiv:2302.06871 [cs.CY]

[28] Stephen R. Piccolo, Paul Denny, Andrew Luxton-Reilly, Samuel H. Payne, and Perry G. Ridge. 2023. Evaluating a large language model's ability to solve programming exercises from an introductory bioinformatics course. *PLOS Computational Biology* 19, 9 (09 2023), 1–16. https://doi.org/10.1371/journal.pcbi.1011511

[29] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Peterson, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. The Robots are Here: Navigating the Generative AI Revolution in Computing Education. arXiv:2310.00658 [cs.CY]

[30] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. arXiv:2304.02491 [cs.HC]

[31] Margot Rutgers. 2021. *Duckbot: A chatbot to assist students in programming tutorials*. Master's thesis. University of Twente.

[32] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research V.1*. ACM, Lugano and Virtual Event Switzerland, 27–43. https://doi.org/10.1145/3501385.3543957

[33] Jaromir Savelka, Arav Agarwal, Marshall An, Chris Bogart, and Majd Sakr. 2023. Thrilled by Your Progress! Large Language Models (GPT-4) No Longer Struggle to Pass Assessments in Higher Education Programming Course. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1*. ACM.

[34] Jaromir Savelka, Arav Agarwal, Christopher Bogart, and Majd Sakr. 2023. Large Language Models (GPT) Struggle to Answer Multiple-Choice Questions about Code. arXiv:2303.08033 [cs.CL]

[35] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F. Bissyandé. 2023. Is ChatGPT the Ultimate Programming Assistant – How far is it? arXiv:2304.11938 [cs.SE]

[36] James Walden, Nicholas Caporusso, and Ludiana Atnafu. 2022. A Chatbot for Teaching Secure Programming. In *Proceedings of the EDSIG Conference ISSN*, Vol. 2473. 4901.

[37] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL]

[38] Laura Weidinger, Jonathan Uesato, Maribeth Rauh, Conor Griffin, Po-Sen Huang, John Mellor, Amelia Glaese, Myra Cheng, Borja Balle, Atoosa Kasirzadeh, Courtney Biles, Sasha Brown, Zac Kenton, Will Hawkins, Tom Stepleton, Abeba Birhane, Lisa Anne Hendricks, Laura Rimell, William Isaac, Julia Haas, Sean Legassick, Geoffrey Irving, and Iason Gabriel. 2022. Taxonomy of Risks Posed by Language Models. In *2022 ACM Conference on Fairness, Accountability, and Transparency* (Seoul, Republic of Korea) *(FAccT '22)*. Association for Computing Machinery, New York, NY, USA, 214–229. https://doi.org/10.1145/3531146.3533088

[39] Terry Yue Zhuo, Yujin Huang, Chunyang Chen, and Zhenchang Xing. 2023. Red teaming ChatGPT via Jailbreaking: Bias, Robustness, Reliability and Toxicity. arXiv:2301.12867 [cs.CL]