Raspberry Pi Foundation

CodeAid: An LLM coding assistant for the classroom

Majeed Kazemitabaar (University of Toronto)

Kazemitabaar, M. (2024). CodeAid: An LLM coding assistant for the classroom. In Learning programming with AI. Understanding computing education (Vol. 5). Proceedings of the Raspberry Pi Foundation Research Seminars.

Available at: rpf.io/seminar-proceedings-vol-5-kazemitabaar

CodeAid: An LLM coding assistant for the classroom

Majeed Kazemitabaar (University of Toronto)

Introduction

Large language model (LLM)-powered tools like ChatGPT can assist students requiring help in programming classes by explaining code snippets and coding concepts, fixing incorrect code, and even writing entire code solutions. However, their productivity-driven and direct responses are concerning in educational settings. Currently, many educators are prohibiting their usage in introductory programming classes to avoid academic integrity issues and students' over-reliance on artificial intelligence (AI). In our research, we have explored the design and evaluation of "pedagogical" LLM-powered coding assistants to scale up instructional support in educational settings. CodeAid is one of the tools that was developed and presented recently at the CHI 2024 conference.

CodeAid: A new tool for teachers

CodeAid was developed with five main features (see Figure 1) that were iteratively updated during the deployment based on student and educator feedback:

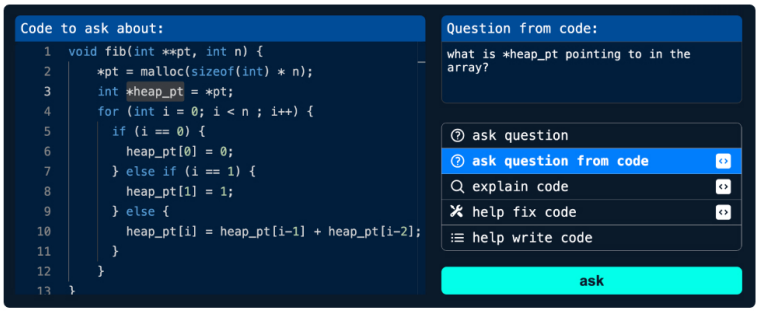
- General question
- Question from code
- Help write code
- Explain code
- Help fix code

In its responses, CodeAid uses LLMs to generate pedagogical answers that do not contain direct code solutions. When asked general questions or to generate code, it provides a natural language response with an interactive pseudocode that allows students to hover over each line and understand what each line does.

Some of the unique properties of CodeAid are:

- **Interactive pseudocode**: Instead of generating code, CodeAid generates an interactive pseudocode. The pseudocode allows students to hover over each line to see a detailed explanation about that line.
- **Relevant function documentations**: Not everything needs to be AI-generated. CodeAid uses retrieval augmented generation (RAG) to display official and instructor-verified documentation of functions relevant to students' queries. These were designed to save time, as well as to allow students to see code examples to learn about using those functions.
- **Suggested follow-up questions**: CodeAid also generates several suggested follow-up questions for students to ask after each response.
- **Annotated incorrect code**: When using the 'Help fix code' function, CodeAid does not display the fixed code. Instead, it highlights incorrect parts of the students' code with suggested fixes.
- **Interactive explained code**: Instead of just displaying a high-level explanation of the entire code in a paragraph, CodeAid renders an interactive component in which students can hover over each line to understand the purpose and implementation of each line of the provided code.
- **Stream rendering of interactive components**: To minimise delay, CodeAid adds the interactive components as each part of the response is received from the LLM, meaning students don't have to wait for the whole response before they can start exploring the code and explanations.

CodeAid's Input:



1. Enter Question
2. Insert Code as Context (optional)
3. Select from the five features

CodeAid's Pedagogical Responses:

Question from Code

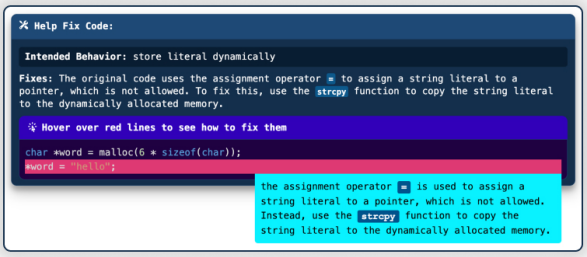
General Question

Help Write Code



1. Read natural language response
2. Use AI-generated pseudo-code
3. Hover over lines to read explanation
4. Use relevant documentations (retrieved)
5. Use suggested follow-up questions

Help Fix Code



1. Does not display fixed code
2. Highlights incorrect parts
3. Provides suggestions to fix code
4. Provides high-level explanation of fixes

Explain Code



1. Hover over each line to read explanation
2. Read high-level summary of entire code

Figure 1. CodeAid's features.

Studying the effectiveness of CodeAid

We developed CodeAid as a wrapper around ChatGPT-3.5 with many interactive features. Our team then deployed CodeAid in a C programming class of 700 students as an optional resource similar to office hours and Q/A forums, for the entire 12-week semester.

We collected data from the following sources:

- Weekly surveys about using CodeAid vs. other, non-AI resources
- CodeAid's log data
- Interviews with 22 students about why and how they used CodeAid
- Final survey asking students to compare CodeAid with ChatGPT
- Interviews with 8 computing educators about CodeAid, their ethical and pedagogical considerations, and comparison with other resources

During the deployment, 372 students used CodeAid and asked 8000 questions. We thematically analysed 1750 of the queries and CodeAid's responses to understand students' usage patterns and types of queries, and CodeAid's response quality in terms of correctness, helpfulness, and directness. Furthermore, we qualitatively analysed data collected from the interviews and surveys to understand the perspectives of students and educators about CodeAid.

Of particular interest in this study was the implementation of CodeAid in a large class setting for the duration of an entire semester. Below are some of the high-level results we found in this deployment.

Students' usage patterns and type of queries

From the 300 students who consented to the use of their data, 160 students used CodeAid less than 10 times, whereas 62 students used it more than 30 times. On average, women used CodeAid significantly more frequently than men (33.8 queries vs. 18.4 queries) while representing only 30% of the entire class.

The thematic analysis revealed four types of queries for CodeAid:

1. Asking programming questions (36%)

- **Code and conceptual clarification queries** (70%) about the programming language, its syntax, its memory management, and operations.
- **Function-specific queries** (15%) about the behaviour, arguments, and return types of specific functions.
- **Code execution probe queries** in which students used CodeAid in a similar way to a compiler to verify execution or evaluate the output of their code on particular inputs.

2. Debugging code (32%)

- **Buggy code resolution queries** (68%) that focused on fixing their incorrect code based on a provided behaviour.
- **Problem source identification queries (**23%) in which students asked CodeAid to identify the source of the errors in their code.
- **Error message interpretation queries** (9%) to better explain the error that they received.

3. Writing code (24%)

- **High-level coding guidance queries** (56%) in which students asked "how-to" questions about a specific coding task.
- **Direct code solution queries** (44%) where students asked CodeAid to generate the direct solution (by copying the task description of their assignment).

4. Explaining code (6%)

- Such as explaining the starter code provided in their assignments.

CodeAid's response quality

The thematic analysis of the responses produced by CodeAid showed that about 80% of the responses were technically correct. The 'General question', 'Explain code', and 'Help write code' features all responded correctly 90% of times, while the 'Help fix code' and 'Question from code' functions were correct 60% of times.

Furthermore, in terms of not revealing direct solutions, CodeAid almost never revealed direct code. Instead, it generated:

- Natural language responses (43%)
- High-level response with pseudocode of generic example code (16%)
- Pseudocode of a specifically requested behaviour (6%)
- Suggested fixes for minor syntax errors (16%)
- Suggested fixes for semantic issues (8%)

Students' perspectives and concerns

Based on the student interviews and surveys:

- Students appreciated CodeAid's **24/7 availability** and being "**a private space to ask questions without being judged**".
- Students also liked CodeAid's **contextual assistance**, which provided a faster way to access relevant knowledge, allowed students to phrase questions however they wanted, and produced responses that were relevant to their class.
- In terms of the **directness of responses**, some students indicated that they wanted CodeAid to produce less direct responses, like hints. Interestingly, some students regulated themselves to not use features that produced more direct responses.
- In terms of **trust**, some students trusted CodeAid while others found that "it can lie to you and still sound confident". Some students trusted CodeAid just because it was part of the course and the instructor endorsed it.
- When asked about their **reasons for not using CodeAid**, students mentioned a lack of need, a preference to use existing tools, wanting to solve problems by themselves, or a lack of trust in AI.
- Even though using ChatGPT was prohibited, students reported using it slightly more than CodeAid. They preferred its easier interface and larger context window to ask about longer code snippets. However, some students did not like ChatGPT since it did a lot of the work for them.

Educators' perspectives and concerns

Overall, most educators liked the **integration of pseudocode** with line-by-line explanations as it provides structure and reduces cognitive load, and is better than finding exact code solutions that are available on the internet. However, some educators were concerned that when the algorithm itself is what students need to learn, revealing its pseudocode will be harmful to their learning.

Most educators wanted to **keep students away from ChatGPT**. They would rather encourage students to use CodeAid instead, and even suggested integrating code editors with code execution right inside CodeAid.

Educators wanted the ability to **customise** CodeAid with course topics, or to be able to activate/deactivate displaying pseudocode.

Lastly, most educators wanted a **monitoring dashboard** to see a summary of asked questions and generated responses so as to reflect on their own instruction. However, other educators mentioned that students should not feel like they are being watched.

Design considerations for future educational AI assistants

Finally, we synthesised our findings into four major design considerations for future educational AI assistants, positioned within four main stages of a student's help-seeking process:

1. **Exploiting AI's unique benefits**: Future AI assistants should clearly distinguish their unique advantages compared to traditional tools to make it clear for students why they should use this specific AI tool versus all the non-AI tools that they already have at their disposal (such as debuggers or a search engine, etc.).
2. **Designing the AI querying interface**: Once the user decides to use AI assistance, how should students formulate questions and provide context? Particular consideration should be given to the aspects of problem identification, query formulation, and context provision.
3. **Balancing the directness of AI responses**: Once the user asks a question, how directly should the AI application respond? And who should have control agency over the directness level?
4. **Supporting trust, transparency, and control**: Upon receiving a response, how can students evaluate it and if necessary steer the AI assistant towards a better response?

Conclusion

CodeAid is one of the first steps in the ethical and pedagogical usage of LLMs to scale up instructional support in educational settings. We can envisage, for example, future AI assistants that are designed based on a particular course taught by a specific instructor with that exact curriculum. Students would be able to access information about the specific course that it is trained on (or has access to), or maybe it would be trained to respond to questions about midterms or final exams for that class. This is just one example of how LLMs can be employed and we hope that these results with CodeAid will guide the design and development of future LLM-powered educational assistants in all learning domains.



Majeed Kazemitabaar
(University of Toronto)

Majeed Kazemitabaar is a PhD student in Computer Science at the University of Toronto researching at the intersection of Human-Computer Interaction and Education, specifically focusing on AI and Programming in learning environments.



Raspberry Pi
Foundation

www.raspberrypi.org



Raspberry Pi Foundation, UK registered charity 1129409