Compiler-Integrated, Conversational AI for Debugging CS1 Programs

Jake Renzella
jake.renzella@unsw.edu.au
University of New South Wales
Sydney, New South Wales, Australia

Alexandra Vassar
a.vassar@unsw.edu.au
University of New South Wales
Sydney, New South Wales, Australia

Lorenzo Lee Solano
l.leesolano@unsw.edu.au
University of New South Wales
Sydney, New South Wales, Australia

Andrew Taylor
andrewt@unsw.edu.au
University of New South Wales
Sydney, New South Wales, Australia

Abstract

Large Language Models (LLMs) present a transformative opportunity to address longstanding challenges in computing education. This paper presents a conversational AI extension to an LLM-enhanced C/C++ compiler which generates pedagogically sound programming error explanations. Our new tool, DCC Sidekick, retains compiler integration, allowing students to see their code, error messages, and stack frames alongside a conversational AI interface. Compiler context improves error explanations, and provides a seamless development experience. We present quantitative analyses of Sidekick's usage and engagement patterns in a large CS1 course. In the first seven weeks of use, 959 students initiated 11,222 DCC Sidekick sessions, generating 17,982 error explanations. Over half of all conversations occur outside of business hours, highlighting the value of these always-available tools. Early results indicate strong adoption of conversational AI debugging tools, demonstrating scalability in supporting large CS1 courses. We share implementation details and lessons learned, offering guidance to educators considering integrating AI tools with pedagogical guardrails.

CCS Concepts

• **Applied computing** → **Education**; • **Computing methodologies** → **Artificial intelligence**.

Keywords

Programming Error Messages, CS1, AI in Education, Generative AI

ACM Reference Format:
Jake Renzella, Alexandra Vassar, Lorenzo Lee Solano, and Andrew Taylor. 2025. Compiler-Integrated, Conversational AI for Debugging CS1 Programs. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025), February 26-March 1, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3641554.3701827

SIGCSE TS 2025, February 26-March 1, 2025, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-0531-1/25/02
https://doi.org/10.1145/3641554.3701827

1 Introduction

The ability to write, understand and debug code are essential components to becoming a computer programmer [3, 9, 20]. High-quality feedback and guidance are often necessary to help with these types of skills. Explanations of code can also help students make deeper connections and build schemas of programming which improve their overall reasoning skills [19, 20].

Traditionally, higher education models utilise, in part, teaching staff to deliver explanations to students. Drop-in help sessions, consultation hours, or online forums [6] provide important opportunities for students to debug errors, clarify misconceptions, and otherwise keep students on track. With increasing enrolments and larger class sizes, the demands on help resources grows.

Over the past two years, educators have explored utilising Large Language Models (LLMs) to generate error explanations [17, 21, 26], supporting time-limited educators and potentially re-imagining the way we teach and learn [10].

In this paper, we attempt to answer two main research questions:

RQ1 How do CS1 students engage with a compiler-integrated, conversational AI tool for programming error explanations, and to what extent do they prefer this over a non-conversational, one-shot alternative?

RQ2 How are generative AI explanation tools used across compile-time vs run-time programming errors?

2 Related Work

There has been increased interest in generative AI over the last few years, sparked by the release of OpenAI's ChatGPT. LLMs, such as ChatGPT, are based upon transformer architecture, and at their core exploit the fact that language follows a specific orderly structure. These models are trained over large quantities of text scraped from the internet. Most LLMs, such as Codex, are also trained on millions of lines of code scraped from open-source repositories, and demonstrate capabilities in code authorship using code-writing benchmarking [2].

2.1 Large Language Models and Tools in CS1

There are many ideas around how large language models can be applied in introductory computing to support student learning, and provide efficiencies. In introductory programming, LLMs have been used to solve simple CS1 programming exercises, with varying

degrees of success [4, 5, 7, 8, 29]. Models have also been used to generate explanations for CS1 code. MacNeil et al. [18] generated code explanations and integrated these into an interactive e-book. They found that students viewed these explanations and perceived them to be helpful [18]. Another example of using large language models in generating code explanations has been in Harvard's CS50 program, via a rubber duck persona [17]. This tool has been used over 50,000 times since June 2023 with an average of 15 prompts per user per day. Integrated into the programming IDE, it is able to provide explanations of code. Informally, it has been positively received by students but no formal analysis of student learning was included in this project. Others have integrated large language models into the compiler to assist with interpreting and solving programming error messages as they occur [26].

Studies have evaluated the efficacy of LLM explanations, demonstrating that they produce sufficient explanations of code in introductory programming [11, 13]. Tools such as the web-based CodeHelp tool, provide immediate support to students working on programming exercises and wanting assistance [16], which reduced the overall anxiety of students who were worried about asking educators for help. One limitation of CodeHelp and other error one-shot explanation tools is that they do not take advantage of the conversational capabilities of large language models, and as such do not allow for follow-up clarifications.

There is growing research into understanding the impact these tools have on learning outcomes. One study found that students who used Codex to learn Python could write better code than the group who did not use Codex, and demonstrated a similar understanding to the control group [12]. Prather et al. [22] found that novices struggled to understand and use these generative AI tools, and were wary about the use of such tools. Issues of over-reliance were also observed [22]. Instructors report serious concerns regarding academic integrity, cheating, and a potential lack of equity, access as well as ethical objections [15].

3 Motivation

In our large Australian University, 3,500 students enrol in the CS1 course each year. Recent and continued growth of student enrolments is managed with a comprehensive programme of student support. In addition to scheduled class time, we run frequent scheduled help sessions for the student cohort. Over a seven week period in a previous term's help sessions, there was a total of 1,484 requests for tutor help from students, with an average time to resolution of 20 minutes. However, students were waiting on average 38 minutes before they could be assisted and only 73.9% of queries were resolved. Waiting time increased during assessment deadline periods. Typically, unresolved student queries are directed to the online class forum, or told to wait until their scheduled class time. Such help sessions, whilst an important component of an overall experience, are still not enough to adequately meet the demand of student queries and required assistance.

Extending our prior work which incorporated one-off LLM-generated error message explanations into educational compilers [25, 27], this work introduces a conversational error explanation module that allows students to engage in dialogue with an LLM about their compile- or run-time programming error. The prior work identified statistically significant differences in compile-time and run-time errors, which motivated our inclusion of RQ2.

DCC Sidekick continues to utilise the full context and memory stack details provided by the compiler, while also generating a unique URL to a web interface. This interface allows students the capability to continue conversations with the LLM, providing the opportunity to guide the LLM to Socratically question the student's understanding of their program and the associated error. The tool can provide reworded, simplified, or expanded explanations depending on the student's needs.

4 DCC Sidekick

DCC Sidekick is an extension to the existing DCC Help tool, forked from the open-source repository as presented by Taylor et al. [27]. The project utilises LLMs to produce enhanced error explanations of C/C++ programming error messages. Functioning as an add-on, our new DCC Sidekick tool leverages DCC's extensive error detection and explanations system [25] to present source code, error information and a conversational interface to an LLM.

While DCC Help provides in situ (in-terminal), one-off error explanations, DCC Sidekick offers conversational guidance. Students are launched into an accessible, web-based dashboard, shown in Figure 1, which presents a comprehensive birds-eye-view of their source code, error message and program state, presented alongside a chat interface. Users can ask follow-up questions and receive Socratic guidance tailored to their current level of understanding.

Like DCC Help, DCC Sidekick's integration with the DCC compiler constructs detailed LLM prompts with no input from the student. This reduces the complexity of prompting a response, distinguishing it from contemporary tools such as CodeHelp [16], which require users to manually provide relevant information about their programming error.

4.1 Compiler Integration

To ensure a consistent learning environment, our students complete all coursework within a personal virtual environment located on the institution's privately hosted LINUX servers, which they access remotely via SSH or a VNC client. When students encounter a run-time or compile-time error using our forked version of DCC, they are presented with a programming error message that contains instructions to request additional "AI-powered" clarifications by running either `dcc-help` or `dcc-sidekick` in the command line. The latter command displays a URL in the command line, as shown in Figure 2, that can be used to launch a new, private DCC Sidekick session and access the web-based dashboard.

The DCC Sidekick command integrates with DCC Help, allowing students to seamlessly transition back and forth between the two environments.

The DCC Sidekick compiler integration is implemented using a small set of shell scripts that are installed globally, alongside a forked DCC in the institution's environment (Figure 3). These scripts utilise existing plug-in hooks within the DCC compiler, caching error data and allowing DCC Sidekick to be launched in response to errors or undefined behaviours.

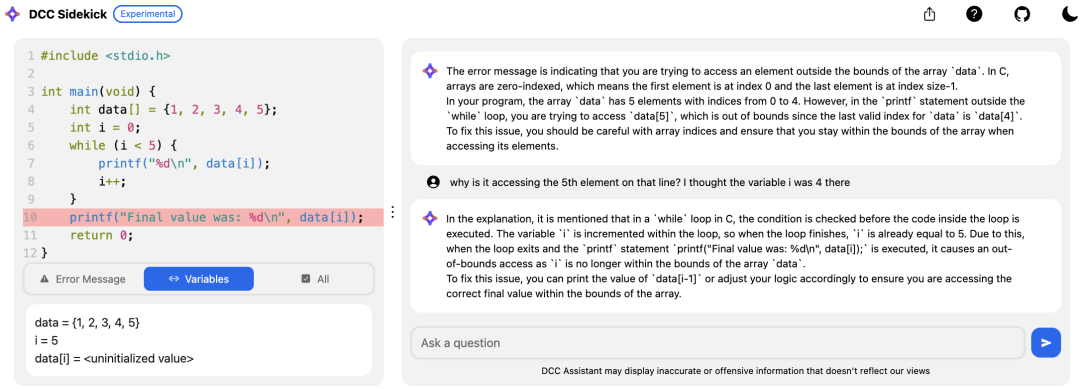

Figure 1: The DCC Sidekick Interface, Where Students can Converse with an LLM about an Error they have Encountered

4.2 Graphical Interface

The DCC Sidekick web-based dashboard is implemented in the React web framework, and prioritises accessibility for novices with a visually clear graphical interface exploring the complexities of a C error. Its separation from the locally installed integration scripts ensures student access irrespective of connection method to the institution's programming environment. Once a URL has been generated, the dashboard itself can either be accessed remotely via a VNC client, or locally on a personal device if they are instead utilising a text-only SSH connection.

When first visiting a session, DCC Sidekick automatically generates and displays an initial error explanation within the chat interface, as seen in Figure 1. This aims to provide debugging guidance on the error, after which users can continue to request follow-up questions and clarifications. The dashboard also displays error message information provided directly by DCC, below the user's program code. For run-time errors, this section of the dashboard dynamically displays the different categories of information across separated tabs, such as variable values, allowing users to explore the underlying complexity of the error at their own pace.

Users can also create read-only session links from within the DCC Sidekick dashboard, allowing course staff to view the existing debugging information and generated explanations, streamlining the process of requesting tutor intervention.

This dashboard is backed by a privately hosted server and database, which is responsible for session creation, explanation prompting, and maintaining session data.

4.3 Prompting Strategy

DCC Sidekick currently utilises OpenAI's ChatGPT3.5-turbo model via the chat completions API to generate both an initial error explanation and responses to follow-up queries. For each session, DCC Sidekick maintains a conversation history to provide to the API, facilitating a coherent, relevant and accurate conversation.

The initial error explanation is generated using a context-rich prompt, derived from the program information provided by DCC. The exact structure of the prompt is dependent on the type of error and the availability of data from DCC [27], but run-time errors in particular can include context such as the original error message,

```
vx06 % dcc atoi.c
atoi.c:7:24: error: expected ';' after expression
  printf("%d\n", x + 5)
                       ^
                       ;
Don't understand?  Get AI-generated help by running: dcc-help, or interactive
 help with dcc-sidekick
vx06 % dcc-sidekick
DCC Sidekick session URL:
https://sidekick.dcc.cse.unsw.edu.au/explanation/Je3nS3sP40RPzscJo8C77

CTRL + Click the URL to open the session in a browser.
vx06 %
```

Figure 2: Launching DCC Sidekick from the DCC Compiler

the user's program code, the function call stack and local variable assignments.

4.4 The Socratic Method

We turned to the Socratic Method to inspire the ways in which students use our AI-assisted tools. The Socratic method has a long history of use in various educational domains to develop student critical thinking skills [14]. This method of instruction proposes guiding student comprehension and learning through the use of guided questions, paying attention to key aspects of the target topic or concept, to help arrive at a solution. These guided questions are based on constructivist learning theories, which propose that learners build knowledge by doing rather than being told [1]. This is also supported by cognitive psychology literature, which states acquiring knowledge is a function of time and conscious effort [23]. The Socratic method has been used extensively in other domains, most commonly law, but there have been few applications in computing education [24]. One such study by Tamang et al. [24] used the Socratic method in teaching introductory computing skills and found that this method of guided self-explanation is more effective than free self-explanations for novice code comprehension. We seek to explore how DCC Sidekick can serve as a vehicle for the Socratic method within the introductory computing course, guiding students to arrive at a solution independently, thereby improving learning outcomes.

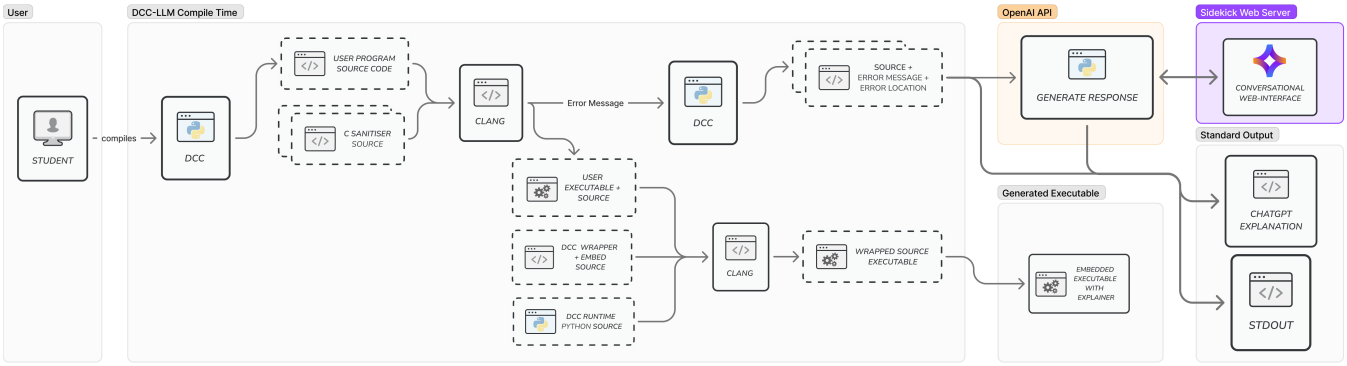

Figure 3: Diagram of Conversational DCC Sidekick Generative Explanation Toolflow as a Component of the Debugging C Compiler at Compile- and Run-Time

```
[System]
You are a part of a programming assistant AI which is
helping a student to debug code. Your job is to check if
an assistant response contains programming code.
If a response contains a code block, please rewrite the
response so that it does not contain any code blocks,
instead explaining the code in words.
If the original response does not contain a code block,
then repeat the response without any changes or additional
text.
```

Figure 4: The Guardrail System Prompt, used to Rewrite Responses that Contain Code Blocks or Solutions

4.5 Additional Guardrails

Earlier work found that large language models often ignore instructions to not provide solutions [16], with one study observing this phenomenon in 48% of cases [26]. To protect against this behaviour, additional guardrails based on prior work by Liffiton et al. [16], were incorporated into the DCC Sidekick prompting strategy. All generated responses are filtered through an additional layer of LLM guardrails, which aim to rewrite overly prescriptive responses that contain code blocks. The system prompt for this rewriting step can be seen in Figure 4.

Additionally, DCC Sidekick's standard system prompt was modified to include provisions to discourage off-topic requests and code solutions. This prompting strategy aims to encourage the generation of responses that provide valuable debugging guidance to novices, without undermining the development of their debugging skills and understanding.

Finally, the tool warns students against over-use if used frequently in a short period of time, gently reminding them that AI assistance will not be available in the final exam.

4.6 Risks

The tool is powered by large language models, which are innately vulnerable to hallucinations, biases, and attempts designed to break guardrails, a well-known and documented phenomenon [28]. This could result in unsatisfactory or inappropriate responses, although we have not observed this behaviour directly.

5 Methodology

To evaluate the user adoption, impact and early markers of efficacy, we deployed DCC Sidekick at our large Australian university in 2024, to a cohort of approximately 1,200 introductory programming students. An insignificant number of students in subsequent C courses may also have used the tools.

Students were instructed on how to use DCC Sidekick, alongside alternative tools like DCC Help, and are prompted to create a session after any compile-time or run-time error within the university's programming environment, as described in subsection 4.1.

5.1 Data Collection

We collected usage data for both DCC Sidekick and DCC Help over the first seven weeks of an introductory programming course in C, which spanned several fundamental CS1 topics such as basic IO, control flow, arrays, and linked lists.

5.1.1 Usage Logs. DCC Sidekick usage was measured by tracking session launches, defined as instances in which a student first creates and visits a DCC Sidekick session in-browser, either in response to a programming error or to ask follow-up questions after first running DCC Help. For both compile- and run-time errors, we log the source code, compiler error messages, and all generated inferences during each session, allowing us to monitor the student engagement across both DCC Sidekick and DCC Help.

5.1.2 Heatmaps and Session Recordings. Anonymised session recordings, user events and click-based heatmaps were collected during the 7-week period using the Microsoft Clarity platform[1], providing further insight into common user interaction patterns. We reflect on the usability and design of DCC Sidekick's graphical interface in subsection 7.1.

5.2 Data Filtering

To comply with the relevant ethics requirements for this study, all collected student data was redacted before analysis. Identifiable

[1]https://clarity.microsoft.com

features were automatically parsed and removed from usage logs and recordings. This involved removing all comments from logged student source code, in an attempt to remove occurrences of user information, such as names, student IDs and emails.

To ensure that the data accurately represents novice programmers, all uses of DCC Sidekick and DCC Help by university staff members, who may have initiated sessions for testing or demonstration purposes, were removed from analyses.

6 Results

DCC Sidekick was made available to students on the first day of the term and we summarise the results of usage over the first seven weeks of the term. Overall, DCC Sidekick has been used by 959 unique users within 11,222 sessions, and generating 17,982 responses. There were an average of 11.7 sessions per student.

On average, each student has spent about 4.3 minutes actively engaging with DCC Sidekick. The maximum active time spent on any single interaction was 11.3 minutes. A total of 25.6% of all conversations resulted in multiple inferences within one session, indicating take up of follow-up dialogue resulting from the initial error. Of all sessions launched, 22.7% had multiple inferences within one session for compile-time errors, and 35.4% for run-time errors. There was an increasing trend in run-time errors as the term progressed. Across all the sessions, there was an average number of 0.6 follow-up questions to each starting message. In any given session where a student asked at least one follow-up question, there was an average of 2.6 follow-up questions asked. In 12.5% of cases, whilst a unique address was created, it was never visited.

A total of 32,090 clicks were recorded across the sessions in the last seven weeks. The main engagement was with the conversational aspect of the tool, with 61.2% of all clicks focused on the conversational side of the tool.

The tool was used extensively both in and out of business hours, with 44% of use occurring within business hours (9am-5pm), and 56% of usage occurring out of business hours (5pm-9am). Ten percent of usage occurs between the hours of midnight and 6am, when no one else is available to provide assistance to the student.

The total cost of all inferences in the eight week period is approximately USD $0.10 per student.

6.1 Usage and Adoption

Figure 5 shows DCC Sidekick launches over each teaching week broken down into compile- and run-time, and total. Adoption grows steadily throughout the teaching period, with usage peaking each Monday corresponding with the weekly assessment due dates. The results show that compile-time errors contribute to the majority of the session launches, especially at the start of term. Run-time errors start to increase steadily from week 5.

While there are more compile-time error sessions of DCC Sidekick, run-time sessions receive higher engagement. At compile-time, around 23% of DCC Sidekick sessions contain follow-on conversations, and approximately 35% at run-time.

Approximately 50% of generative AI error explanations originated from the in-terminal DCC Help tool consistently throughout the term, indicating that students found value in both tools.

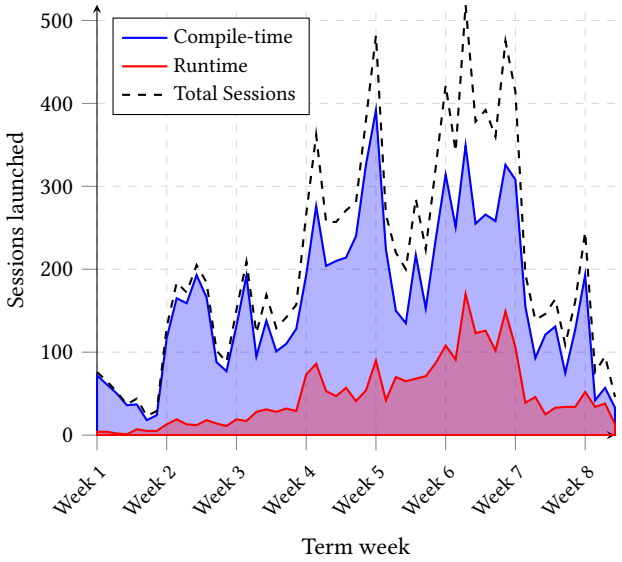

Figure 5: Launches Over Time of the DCC Sidekick Tool, Comparing both Compile- and Run-time Errors

7 Discussion

Addressing RQ1, adoption trends shown in Figure 5 indicate strong user acceptance. With 11,222 sessions initiated by 959 unique students, the tool demonstrates its capacity to support a large-scale introductory course. This level of engagement suggests that students find value not just in AI-enabled error explanations, but conversational debugging assistance provided by DCC Sidekick.

Of the total sessions with follow-up dialogue, 1,969 were related to compile-time errors, while 899 addressed run-time issues. The predominance of follow-on dialogue in compile-time error sessions, particularly in the early weeks, aligns with the typical progression of novice programmers who often grapple with syntax and basic structural errors in the initial stages of learning.

Addressing RQ2, we observed a marked increase in run-time error assistance requests starting from week 4. This shift coincides with the introduction of more complex programming tasks in the CS1 course, particularly those involving memory management and pointers. As students tackle these more challenging concepts, they appear to be turning to DCC Sidekick rather than external resources like ChatGPT, and the non-conversational, but in-terminal DCC Help tool. This uptake is noteworthy, as it suggests that the benefits of the DCC Sidekick conversations outweigh any perceived inconvenience of switching environments.

Usage patterns reveal a concentration of activity around assessment deadlines, peaking on Mondays when weekly assessments are typically due, and especially in week 6, which correlates with the major assignment deadline. This trend underscores the role of DCC Sidekick in supporting students during high-stress periods when the teaching team may become overwhelmed by requests for assistance. Similarly, more than half the usage of the tool occurs outside of business hours, when students would not otherwise be able to get help as needed. This underscores the capability of AI-enabled tools to support teaching teams when they need it most.

The high return rate and number of follow-up questions in DCC Sidekick sessions demonstrates students are meaningfully engaging to deepen their understanding and debugging techniques.

While DCC Sidekick adoption is strong, it's important to note that the existing DCC Help tool remains the primary source of AI-generated assistance. This inclination for in situ help suggests that immediacy and seamless integration is preferred, until a more thorough debugging session is warranted. This is a relevant finding to the wider community, as many general-purpose AI tools such as ChatGPT, CodeHelp [16], or Harvard's CS50 forum bot [17] are external to the development environment.

7.1 Lessons Learned

Our experience developing and deploying DCC Sidekick presents valuable insights for tool makers, researchers, and educators:

Integration is key: Despite allowing external LLMs like ChatGPT, DCC Help and DCC Sidekick usage indicates that our approach of a) integrating with the compiler to produce more accurate responses, and b) seamlessly integrating within existing workflows, is crucial for adoption. This is important, as we can retain our pedagogical guardrails such as rate limiting and encouraging language models to provide guidance rather than solutions.

Balancing immediacy and depth: While DCC Sidekick offers a conversational interface, the continued high usage of the simpler, terminal-based DCC Help tool indicates that students value immediate, in situ assistance for certain types of errors such as simpler syntax errors in earlier weeks. Since both tools were used heavily, we believe this indicates that future tools should consider how to benefit from both in-situ tools like DCC Help, and more in-depth tools like Sidekick.

Investment in conversations: When students do choose to engage with conversational debugging in DCC Sidekick, they invest more time and effort. This is higher for run-time error cases, evidencing the role that conversational AI has in supporting more complex debugging tasks.

8 Future Work

We identify three avenues for future research:

Student surveys and pedagogical impact: We plan to interview and survey CS1 students to gain deeper insights into DCC Sidekick's efficacy and impact on long-term learning.

Time-series analyses: By analysing student progress immediately following a DCC Help or DCC Sidekick invocation, we can quantitatively explore the impact these tools have on metrics like successful compilation and successfully passing autotests.

Model evaluation and refinement: We plan to assess alternative commercial and open-source LLMs, exploring on-premises hosting options and fine-tuning techniques to improve CS1 error debugging performance, and to address privacy concerns.

9 Limitations

We identify three limitations that impact the validity and generalisability of our findings.

Despite integrating an open-ended feedback form into the DCC Sidekick tool, low response rates mean that this work lacks understanding of the qualitative aspects of DCC Sidekick's impact on the learning experience. For example, it is not clear why students choose to transition from the in-terminal DCC Help responses to the conversational DCC Sidekick tool. Future work such as surveys will help ascertain students' perceptions of the tool's efficacy or its influence on their problem-solving strategies. Surveys would also allow us to evidence our claims that DCC Sidekick prevented or dissuaded students away from general-purpose LLMs like ChatGPT.

Secondly, our study lacks analysis of learning outcomes. While engagement metrics indicate user acceptance, they do not necessarily correlate with improved programming skills or conceptual understanding. A comparative study of academic performance between DCC Sidekick users and non-users would provide more conclusive evidence of its educational value.

Finally, this study explores a single institution's CS1 course, limiting the generalisability and long-term adoption trends.

10 Conclusion

This paper presents DCC Sidekick, a novel, compiler-integrated conversational AI tool designed to support debugging activities in a large-scale CS1 course. Our approach combines in-terminal responses via the forked DCC Help, with DCC Sidekick: a web-based conversational interface, providing students with flexible, context-aware debugging sessions. The high adoption rate — 959 students initiating over 11,222 sessions in seven weeks, and significant engagement over 4.3 minutes on average, demonstrates the tool's impact. We explore behaviours across compile- and run-time errors, indicating that the nature of a programming error influences the type of AI help students engage with.

DCC Sidekick's compiler integration offers students a compelling alternative to general-purpose AI tools like ChatGPT, despite DCC Sidekick's pedagogical guardrails. By keeping students within our guided learning environment, we aim to foster genuine understanding and skill development - dissuading the use of tools that are not pedagogically aligned.

DCC Sidekick demonstrates significant potential for scaling support in large programming courses, particularly in its ability to allow ongoing exploration and Socratic guidance of programming errors in a novel, compiler-integrated environment. The approach is shown to handle high volumes of student queries, which is especially valuable outside business hours and near assignment deadlines. As we continue to refine our approach to AI-generated support in introductory computer science courses, we believe that AI-assisted tools will play a crucial role in computing education. The advancement of these tools, whilst promising, needs to be executed responsibly, ensuring safe pedagogical environments that encourage learning.

Acknowledgments

This work was partially supported via the Google Award for Inclusion Research program.

References

[1] Mordechai Ben-Ari. 1998. Constructivism in computer science education. *SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education)* 30, 1 (1998), 257–261. https://doi.org/10.1145/274790.274308

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish

Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374* (7 2021). http://arxiv.org/abs/2107.03374

[3] Kathryn Cunningham, Yike Qiao, Alex Feng, and Eleanor O'Rourke. 2022. Bringing "high-level" Down to Earth: Gaining Clarity in Conversational Programmer Learning Goals. In *SIGCSE 2022 - Proceedings of the 53rd ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 551–557. https://doi.org/10.1145/3478431.3499370

[4] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 1136–1142. https://doi.org/10.1145/3545945.3569823

[5] Paul Denny, Stephen MacNeil, Jaromir Savelka, Leo Porter, and Andrew Luxton-Reilly. 2024. Desirable Characteristics for AI Teaching Assistants in Programming Education. In *2024 on Innovation and Technology in Computer Science Education*. 408–414. https://doi.org/10.1145/3649217.3653574

[6] Augie Doebling and Ayaan M. Kazerouni. 2021. Patterns of Academic Help-Seeking in Undergraduate Computing Students. In *21st Koli Calling International Conference on Computing Education Research*. Association for Computing Machinery, 1–10. https://doi.org/10.1145/3488042.3488052

[7] 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 *ACM International Conference Proceeding Series*. Association for Computing Machinery, 10–19. https://doi.org/10.1145/3511861.3511863

[8] James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A. Becker. 2023. My AI Wants to Know if This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises. In *ACM International Conference Proceeding Series*. Association for Computing Machinery, 97–104. https://doi.org/10.1145/3576123.3576134

[9] Max Fowler, David H. Smith IV, Mohammed Hassan, Seth Poulsen, Matthew West, and Craig Zilles. 2022. Reevaluating the relationship between explaining, tracing, and writing skills in CS1 in a replication study. *Computer Science Education* 32, 3 (2022), 355–383. https://doi.org/10.1080/08993408.2022.2079866

[10] Simone Grassini. 2023. Shaping the Future of Education: Exploring the Potential and Consequences of AI and ChatGPT in Educational Settings. https://doi.org/10.3390/educsci13070692

[11] 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. In *ICER 2023 - Proceedings of the 2023 ACM Conference on International Computing Education Research V.1*. Association for Computing Machinery, Inc, 93–105. https://doi.org/10.1145/3568813.3600139

[12] 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 *Conference on Human Factors in Computing Systems - Proceedings*. Association for Computing Machinery, 1–23. https://doi.org/10.1145/3544548.3580919

[13] Natalie Kiesler and Daniel Schiffner. 2023. Large Language Models in Introductory Programming Education: ChatGPT's Performance and Implications for Assessments. *arXiv preprint arXiv:2308.08572* (8 2023). http://arxiv.org/abs/2308.08572

[14] Dubravka Knezic, Theo Wubbels, Ed Elbers, and Maaike Hajer. 2010. The Socratic Dialogue and teacher education. *Teaching and Teacher Education* 26, 4 (5 2010), 1104–1111. https://doi.org/10.1016/j.tate.2009.11.006

[15] Sam Lau and Philip Guo. 2023. From "Ban It Till We Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. In *ICER 2023 - Proceedings of the 2023 ACM Conference on International Computing Education Research V.1*. Association for Computing Machinery, Inc, 106–121. https://doi.org/10.1145/3568813.3600138

[16] 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*. Association for Computing Machinery, 1–11. https://doi.org/10.1145/3631802.3631830

[17] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J Malan. 2024. Teaching CS50 with AI: Leveraging Generative Artificial Intelligence in Computer Science Education. In *55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*, Vol. 1. ACM, Portland, USA, 750–756. https://doi.org/10.1145/3626252.3630938

[18] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2022. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. In *54th ACM Technical Symposium on Computer Science Education*. ACM, 931–937. http://arxiv.org/abs/2211.02265

[19] Samiha Marwan, Nicholas Lytle, Joseph Jay Williams, and Thomas Price. 2019. The Impact of Adding Textual Explanations to Next-step Hints in a Novice Programming Environment. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*. Association for Computing Machinery, 520–526. https://doi.org/10.1145/3304221.3319759

[20] Laurie Murphy, Sue Fitzgerald, Raymond Lister, and Renée McCauley. 2012. Ability to 'explain in plain english' linked to proficiency in computer-based programming. In *Ninth annual international conference on International computing education research (ICER '12)*. ACM, 111–118.

[21] 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 Petersen, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. The Robots are Here: Navigating the Generative AI Revolution in Computing Education. In *ITiCSE-WGR 2023 - Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education*. Association for Computing Machinery, Inc, 108–159. https://doi.org/10.1145/3623762.3633499

[22] 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. *ACM Transactions on Computer-Human Interaction* 31, 1 (4 2023), 1–31. https://doi.org/10.1145/3617367

[23] John Sweller. 2023. Cognitive load theory: What we learn and how we learn. In *Learning, design, and technology: An international compendium of theory, research, practice, and policy*. Cham: Springer International Publishing., 137–152.

[24] Lasang Jimba Tamang, Zeyad Alshaikh, Nisrine Ait Khayi, Priti Oli, and Vasile Rus. 2021. A Comparative Study of Free Self-Explanations and Socratic Tutoring Explanations for Source Code Comprehension. In *SIGCSE 2021 - Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. Association for Computing Machinery, Inc, 219–225. https://doi.org/10.1145/3408877.3432423

[25] Andrew Taylor, Jake Renzella, and Alexandra Vassar. 2023. Foundations First: Improving C's Viability in Introductory Programming Courses with the Debugging C Compiler. In *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 346–352. https://doi.org/10.1145/3545945.3569768

[26] Andrew Taylor, Alexandra Vassar, Jake Renzella, and Hammond Pearce. 2023. Dcc –help: Generating Context-Aware Compiler Error Explanations with Large Language Models. (8 2023). http://arxiv.org/abs/2308.11873

[27] Andrew Taylor, Alexandra Vassar, Jake Renzella, and Hammond Pearce. 2024. dcc - Help: Transforming the Role of the Compiler by Generating Context-Aware Error Explanations with Large Language Models. In *SIGCSE 2024 - Proceedings of the 55th ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 1314–1320. https://doi.org/10.1145/3626252.3630822

[28] 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 *ACM International Conference Proceeding Series*. Association for Computing Machinery, 214–229. https://doi.org/10.1145/3531146.3533088

[29] Michel Wermelinger. 2023. Using GitHub Copilot to Solve Simple Programming Problems. In *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 172–178. https://doi.org/10.1145/3545945.3569830