dcc --help: Transforming the Role of the Compiler by Generating Context-Aware Error Explanations with Large Language Models

Andrew Taylor
University of New South Wales
Sydney, New South Wales, 2052, Australia

Alexandra Vassar
University of New South Wales
Sydney, New South Wales, 2052, Australia

Jake Renzella
University of New South Wales
Sydney, New South Wales, 2052, Australia

Hammond Pearce
University of New South Wales
Sydney, New South Wales, 2052, Australia

ABSTRACT

In the challenging field of introductory programming, high enrolments and failure rates drive us to explore tools and systems to enhance student outcomes, especially automated tools that scale to large cohorts. This paper presents and evaluates the *dcc --help* tool, an integration of a Large Language Model (LLM) into the Debugging C Compiler (DCC) to generate unique, novice-focused explanations tailored to each error. *dcc --help* prompts an LLM with contextual information of compile- and run-time error occurrences, including the source code, error location and standard compiler error message. The LLM is instructed to generate novice-focused, actionable error explanations and guidance, designed to help students understand and resolve problems without providing solutions. *dcc --help* was deployed to our CS1 and CS2 courses, with 2,565 students using the tool over 64,000 times in ten weeks. We analysed a subset of these error/explanation pairs to evaluate their properties, including conceptual correctness, relevancy, and overall quality. We found that the LLM-generated explanations were conceptually accurate in 90% of compile-time and 75% of run-time cases, but often disregarded the instruction not to provide solutions in code. Our findings, observations and reflections following deployment indicate that *dcc --help* provides novel opportunities for scaffolding students' introduction to programming.

CCS CONCEPTS

• **Software and its engineering** → *Compilers*; • **Social and professional topics** → **CS1**; • **Computing methodologies** → *Natural language processing*.

KEYWORDS

CS1, AI in CS1, AI in Education, Generative AI, Large Language Models, Compiler Error Messages, Debugging, Error Message Enhancement, Programming Error Messages

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM ISBN 979-8-4007-0423-9/24/03…$15.00
https://doi.org/10.1145/3626252.3630822

ACM Reference Format:
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 *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630822

1 INTRODUCTION

Programming has remained a difficult concept to teach and learn, with globally high attrition and failure rates in introductory computing (CS1) courses [9]. One of the most common difficulties cited when learning to program is the inability to read, understand and act on compiler error messages [7, 18, 20]. While confusing compiler error messages may be frustrating for even an experienced programmer, in some cases, they may create a learning barrier for novices. Previously, the authors of the Debugging C Compiler (DCC) showed how DCC improved the viability of teaching C in introductory programming courses [30]. The existing implementation of DCC, which is open-source and publicly available, produces enhanced compiler error messages and explanations at both compile- and run-time to support novices in addressing common C errors. The authors claim that the tool was in part motivated by growing enrolments in CS1 courses in the previous decade, with our institution's computing cohorts growing by 45%, introducing challenges when providing adequate support to meet the demand at this scale.

While DCC's enhanced error detection and explanations assist students in writing safer, more correct C code, students can still require assistance from teaching teams to explain error messages in terms they understand. Increased cohort sizes mean productivity and motivation are impacted, as students frequently encounter delays in receiving these necessary explanations.

Large Language Models (LLMs) are a form of neural network; an artificial intelligence that can learn the context and meaning of written language, including code, from large datasets of text used to train models. The open release of one such model in November 2022, OpenAI's ChatGPT (https://chat.openai.com), has sparked interest in how educators can use these types of models to improve outcomes in CS1.

While recent studies have evaluated the efficacy of using LLMs to generate compiler error explanations [4, 22, 24], there has been no published integration of an LLM into the compiler itself. This could transform the role of the compiler from simply generating error messages to producing detailed, contextualised, natural language

guidance and feedback designed for novices. In this work we contribute such a tool. It can support novice programmers at scale, providing bespoke, on-demand guidance to support their learning. Our tool is open source and can be found at http://dcc.cse.unsw.edu.au.

2 BACKGROUND

2.1 Compiler Error Messages

Compilers are a primary interface between a novice programming student and learning a language; however, issues interpreting and acting on compiler error messages have been well documented [3, 7, 18, 28, 31]. Students have even been known to change their majors, citing cryptic and hard-to-learn compiler error messages as one of the reasons [14]. Initial work has been done to enhance the readability of compiler error messages and explore the use of enhanced error messages [6–8, 10, 13, 27]. However, the results of this work are inconclusive, with no clear evidence in favour of enhanced compiler error messages [5]. Although there is evidence that suggests students are reading compiler error messages, it is not directly clear how many students successfully understand and act on them [7, 13]. One of the difficulties when enhancing error messages is the manual process involved in identifying and integrating the enhanced messages into the compiler. The hand-crafted nature of these explanations means that they fail to cover the breadth of possibilities of potential student errors.

2.2 The Debugging C Compiler

The Debugging C Compiler (DCC) is a C/C++ compiler designed for novice programming students [30]. The tool has been used millions of times, by thousands of students, to enable a fundamentals-first introductory programming course [30]. DCC supports students by providing enhanced compiler error messages, which detect common errors that standard C implementations (such as GCC and Clang) miss. DCC achieves its goals via the following features, all incorporated into a single easy-to-use package [30]:

- **Additional compile- and run-time error detection**: DCC embeds run-time error detection tools, such as Valgrind [25], AddressSanitizer and GDB into the generated executable to provide additional information to the DCC error explanation system including call stack printout and memory leak detection. Clang and GCC static analysis options are used to provide additional compile-time checks.
- **Enhanced error messages**: The DCC explainer, both at compile-time and run-time, interprets and explains the most common novice error messages using simple, hand-crafted explanations for a range of common error types.
- **Additional context**: DCC embeds source code into the executable to illustrate the location of run-time errors. This provides additional information to the student and allows more efficient bug identification and resolution.

In Listing 1, we compile and execute a program which attempts to access an uninitialised variable. GCC does not detect the error; however, DCC flags the bug, and the location of the error.

```
$ gcc program.c && ./a.out
0
$ dcc program.c && ./a.out
Runtime error: uninitialized variable accessed.
Execution stopped in main() in the program.c at line 6:

int main(void) {
    int numbers[10];
    for (int i = 1; i < 10; i++) {
        numbers[i] = i;
    }
--> printf("%d\n", numbers[0]);
}
Values when execution stopped:

numbers = {<uninitialized value>,1,2,3,4,5,6,7,8,9}
numbers[0] = <uninitialized value>
```

Listing 1: Uninitialised Variable—gcc: no error, dcc: error.

DCC's access to additional context at run-time including original program source, error location (line number), and the GDB call stack at the moment a run-time error occurs is critically important to generate actionable, contextual explanations at run-time.

2.3 Large Language Models and Education

Large Language Models, such as GPT-3 and later Codex models [11], display significant general-purpose and cross-domain capabilities in natural language processing. These transformer-based models are trained over large quantities of text scraped from the internet, and in Codex's case, with code mined from millions of open-source repositories. Codex demonstrated state-of-the-art capabilities in code authorship via code-writing benchmark tests (such as HumanEval [11]), later underpinned the commercial GitHub Copilot.

A recent training methodology, Reinforcement Learning with Human Feedback (RLHF), can be applied to LLMs to produce models capable of following user intents [26]. This is beneficial to override the default tendency of LLMs to simply act as a 'smart autocomplete', and makes it easier to have the models actually 'follow instructions'. The premier LLM in this space is OpenAI's ChatGPT, which was fine-tuned from GPT-3 and Codex to provide conversational-style instruction following completions. In software engineering, ChatGPT can thus be used to help translate and debug code and provide code explanations using natural language.

The adoption of ChatGPT-style LLMs within education is currently mixed, with some individuals, schools, and systems forbidding generative AIs (e.g. in Australian primary and secondary schools [23]) and others moving towards targeted and mass utilisation. The primary concern stems from the potential for wide-scale academic misconduct, but proponents of the technology argue that careful usage will unlock new pedagogical tools and strategies. Kasneci et al. provide a comprehensive survey in this area [19], finding that, for example, ChatGPT is already being used for educational methods such as generating tests, quizzes, and flashcards.

Research into the benefits and pitfalls of using LLMs to support novice learners in CS1 is still in its infancy [4, 12, 16, 17, 24**?**]. The majority of the work has focused on testing how well these tools can solve a public repository of CS1 programming problems (usually in languages other than C); and to what extent natural language modifications or prompts can lead to the generation of successful solutions. For instance, one study showed that Codex can perform better than most novice students on code writing questions in CS1 courses, scoring in the 75th percentile, and generating multiple

solutions to problems [16]. Another study found, using 31 questions from a popular software testing textbook, that ChatGPT was able to respond to 77.5% of questions and provide a correct answer in 55.6% of cases (further prompting of the tool led to a slightly higher rate of correct answers and explanations) [17]. Denny et al. found GitHub Copilot to be effective in solving standard introductory programming problems, successfully solving about half of 166 problem sets the on first attempt, and a further 60% of the remaining problems with some natural language changes to the problem specification [12]. Concurrently with this work, Harvard's CS50 has released an 'AI chatbot' [15] which aims to help students find bugs in their programs and perform Q&A over unfamiliarities or error messages. This tool is external to the compiler and implemented in their online platform, utilising a bespoke LLM instead of ChatGPT to minimise accidental over-help. We instead explore the incorporation of ChatGPT into the compiler directly, such that it may generate on-demand, novice-friendly explanations of compile- and run-time errors.

As the complexity of the problem grows, so does the reliance on human input and prompting [1]. There are further concerns that tools such as Codex can lead to over-reliance on programming tasks [11], where students agree with LLM output even when it is incorrect. Some research has shown that producing explanations can reduce the over-reliance on the LLM models and improve overall decision-making [2, 32]; however, issues regarding student integrity and over-reliance persist. To produce prompts capable of asking the right questions, the user must be able to understand the problem they are experiencing in the first instance, which is not often the case with a novice programming student.

Recent work by Leinonen et al. [22] explored the evaluation of Codex in producing enhanced programming error messages. The study selected a subset of Python error messages that were reported by students as being the least readable, and then the researchers produced code examples that would trigger these types of errors. One of the limitations of this work is the lack of an authentic classroom setting and the absence of programs written by the students themselves. The study evaluated a series of prompts designed to explain compiler errors and generate actionable fixes, and subsequently, the quality of the code fixes and error explanations generated in response to the prompts. Leinonen et al. found that error message explanations and proposed fixes require improvement before being introduced in CS1 due to students' over-reliance and trust in the correctness of the message explanation. Still, Leinonen et al. found that plain-language explanations of errors can decrease how threatening compiler error messages appear, and could be instrumental in improving learning outcomes for students at scale.

In our case, we intend for the generative explanations to guide students to understand compiler output, including DCC output, which may be confusing or lacking in contextual details. By providing a simple AI-generated explanation in the development environment, we hope to support student progress and understanding without requiring delays until staff are available to assist.

3 A NEW TOOLFLOW: GENERATIVE HELP

We introduce a new toolflow whereby DCC uses the OpenAI ChatGPT 3.5 API (model: *gpt-3.5-turbo-0301*) at compile- and run-time to consume source code, error messages, and locations to generate contextual, novice-friendly error and warning explanations designed to augment typical compiler output. Using the OpenAI API at run-time is only possible with a tool like DCC, as typical C implementations such as GCC and Clang do not have access to the source code in the executable.

Our implementation was created by forking the open-source DCC Github repository and contributing the open-source --help extension (https://github.com/COMP1511UNSW/dcc). The overview of compile- and run-time process (Figure 1) describes how the new toolflow produces generative explanations. When *dcc --help* is executed (Listing 3), the previous error (compile- or run-time) is captured and prepared for an explanation by ChatGPT 3.5. A text prompt, not visible to the student, is generated (example of a prompt for run-time can be seen in Listing 2), which includes:

- Base prompt,
- Program source code,
- DCC enhanced error message,
- Error line number, and,
- Values in the stack frame at error time (*run-time only*).

```
system:content:
You are a tutor helping a student.
Do not fix the program.
Do not provide code.

user:content:
This is my C program <<Source Code>>
Help me understand this message from the C compiler:
<<DCC Enhanced Explanation>>
Error location: Line 6
Values: <<GDB Stack Frame for run-time>>
Remember, you are tutor helping a student.
Do not write code for the student.
```

Listing 2: Example of a Prompt Sent to ChatGPT, Not Visible by Users (Run-Time Error)

The prompt, shown in Listing 2, is sent via an HTTP API call to OpenAI's *gpt-3.5-turbo-0301* and the response is streamed back into the student's terminal environment (Listing 3) in situ via the HTML Event stream format[1]. An example of the run-time error/explanation workflow explaining Listing 1 is shown in Listing 3.

```
$ dcc program.c && ./a.out
<<Standard compiler output (compile-time only)>>
<<Non Gen-AI DCC enhanced explanation>>
Don't understand? Get AI-generated with `dcc --help`

$ dcc --help
Here is an AI generated explanation. Be careful - it may
    be wrong!

The error message you received is indicating that you are
     trying to access an uninitialized variable. In this
     case, the uninitialized variable is numbers[0].
In the program, you have declared an array called numbers
     which has 10 elements. However, you have not
    assigned any value to numbers[0] before trying to
    print its value with printf().
```

[1] https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events

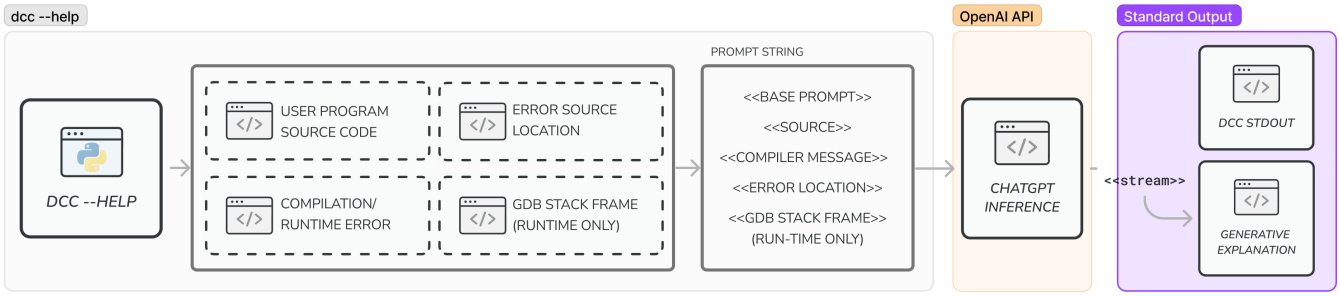

Figure 1: Diagram of New *--help* Generative Explanation Toolflow in the Debugging C Compiler at Compile- and Run-Time.

```
To fix this error, you can simply assign a value to
    numbers[0] before trying to print it. This can be
    done by adding numbers[0] = 0; before the printf()
    statement.
```

Listing 3: Example Generative Explanation (Run-Time Error)

With *dcc --help*, we thus introduce a *generative explanation* written by ChatGPT, in addition to the current DCC enhanced explanations. The complete nomenclature follows:

- **Standard** compiler output for compile-time errors,
- **DCC enhanced error message** (Listing 3),
- **Generative explanation** (LLM-generated) (Listing 3).

3.1 Additional Design Choices

Additional design decisions were made in order to safeguard the student learning experience. For example, *dcc --help* warns students that AI-generated explanations may not be correct. The tool also detects if a student is generating many *dcc --help* explanations in a short amount of time, warning them that they should use *dcc --help* sparingly, and should always understand the code they are writing—we do not make the AI help tool available in the exam environment. We present our reflections on this in section 6.

4 METHOD

This study aims to evaluate the efficacy of OpenAI's ChatGPT API (*gpt-3.5-turbo-0301*) LLM in producing context-aware generative explanations in response to compiler errors in DCC. To evaluate the quality of responses, we released a version of DCC containing our help extension to students in a range of CS1 and CS2 courses at a large Australian university, then tracked its usage when generating error explanations. The CS1 curriculum at this institution covers a range of introductory topics, ranging from control flow to the use of arrays and linked lists. Students were informed that if they encountered either a compile- or run-time error, they could proceed to run the *dcc --help* command to execute the LLM inference and generate a response. We then logged each occurrence including the source code containing the error, the error location (line number), the raw C compiler error, and the ChatGPT response. The same prompt strategy was used in each instance, injecting the relevant source code, error location, and DCC error (Listing 2). Additionally, usage statistics were collected by logging all student activities associated with the DCC and *dcc --help* tool.

4.1 Data Extraction

This study utilises the student-generated error/explanation occurrences, including the source code of problematic code that arose throughout the students' regular coursework activities. We randomly sampled from the over 64,000 uses of *dcc --help* and extracted 200 compile-time errors and 200 run-time errors for a total evaluation set of 400 error/explanation pairs. An additional randomly selected mix of 15 error/explanation pairs was categorised jointly by the four reviewers together to reach consensus on categorisation strategies and to ensure a consistent approach.

4.2 Data Filtering

In line with the requirements of our relevant Ethics body's approval of this research project, data processing included anonymising the source code, including stripping potential student identifiers from comments and logs. Regular expressions were able to remove these from source code and filenames effectively, and best efforts were taken to remove names from comments (such as header-comments). Other comments, as they may describe the intended functionality of the code, were preserved—these source code comments will impact the quality of the LLM's responses. Finally, any staff who may have generated logs when testing the tool were filtered out.

4.3 Data Analysis

Four reviewers (three authors of this paper and one student researcher) were each asked to evaluate 100 error/explanations from the extracted set of 400 as described in subsection 4.2. Each of the author reviewers has a significant history of teaching introductory computing, and the student researcher has been teaching introductory computing for the last two years. Reviewers were instructed to assess each LLM-generated explanation (student source code with error, compiler error message, and LLM-generated contextual explanation) across the following properties:

- *Conceptual accuracy* (Yes/No) - is the generated response conceptually correct?
- *Inaccuracy* (Yes/No) - are there inaccuracies present?
- *Correctness* (Yes/No) - is the provided guidance technically correct resulting in being able to solve the problem?
- *Relevance* (Yes/No) - is the generated message relevant to the encountered error?
- *Completeness* (Yes/No) - is the provided explanation complete, not missing any critical information that would help students understand the error?

- *Code Solution* (Yes/No) - is the solution provided as code in the generated response?
- *Response type* (Peer/Tutor) - is the generated response commensurate in quality with a *peer* or a *tutor*?

Prior to commencing the classification, the reviewers categorised 15 error/explanation pairs, separate from the analysis set, as a group to ensure that everyone had the same interpretation of the measured aspects. For each error/explanation pair, the reviewers could access the source code that generated the error, the output from DCC specifying the line number of where the error has occurred, the DCC enhanced explanation of the error, which also included the state of variables at the time when the error was produced (for run-time errors), and the LLM-generated explanation produced with this data. All of these were used in analysing the generated explanation by each reviewer.

4.4 Reliability of Evaluation

Each reviewer was randomly assigned 100 errors/explanation pairs (50 compile-time, 50 run-time) for evaluation. In addition, ten percent of each reviewer's errors were randomly allocated to all other reviewers to determine inter-rater reliability, bringing the total errors analysed by each reviewer to 130. To address limitations of percentage agreement which does not take into account reviewers agreeing by chance, Light's Kappa [21] was used to determine an overall index of agreement.

5 RESULTS

Results are presented in Table 1. These provide the frequency of "Yes" responses across each category (see subsection 4.3 for categories). In addition, the measure of inter-rater reliability across each category and the four reviewers is calculated using Light's kappa [21]. Guidelines [21] are also provided for interpreting the reliability values, where 0 indicates no agreement and 1 indicates perfect agreement; *moderate* agreement is indicated by values $0.41 < \kappa < 0.60$, and $0.61 < \kappa < 0.80$ indicates *substantial* agreement. Moderate agreement was observed for conceptual accuracy, the relevance of response, completeness of response, inaccuracy present, and type of response. Substantial agreement was observed in the technical correctness of the guidance. Overall, the LLM-generated explanations were clear across both compile-time and run-time errors, with better performance at compile-time in comparison to run-time. At compile-time, 90% conceptual accuracy was observed, as compared to 75% at run-time. No inaccuracy was present in the generative explanation for compile-time errors in 78% of cases. Lower inaccuracy was observed at run-time, with only 53% of explanations having no inaccuracy. This trend is observed in other categorisations also, specifically noting 93% correctness at compile-time as compared to 66% at run-time. Generative explanations were consistently deemed relevant to the error, with 92% relevancy at compile-time and a reduced 75% at run-time. There is a large difference in how complete the explanations are between compile-time explanations at 72%, as compared to run-time, reduced 39%. Despite the prompt instructing that no code be given out with the explanation, 48% of the explanations at compile-time and 49% of run-time explanations contained blocks of code that reviewers considered too much help. Finally, the generative explanations were approximated to an overall quality in terms of tutor-level or peer-level. Results found that explanations were deemed tutor-like for 72% of compile-time explanations. In contrast, only 45% of run-time explanations were deemed to be of a quality that tutors are expected to provide. Overall, run-time use of LLM-generated explanations was consistently worse.

Table 1: Review of Generative Explanations: Frequencies of "Yes" per Category and the Inter-Rater Reliability.

Measure (*n=400*)	CT (*n=200*)	RT (*n=200*)	Light's κ
Conceptually accurate	90%	75%	0.56
No Inaccuracy in solution	78%	53%	0.45
Correctness of response	93%	66%	0.66
Relevance of response	92%	75%	0.45
Completeness of response	72%	39%	0.43
Solution is provided	48%	49%	0.73
Response of **peer** quality	28%	53%	0.45
Response of **tutor** quality	72%	45%	0.45

Figure 2 depicts weekly usage of the tool by our CS1 and CS2 students. In week one, 1,032 uses occurred, increasing to over 9,700 in the final week, demonstrating increasing popularity and adoption of the tool. On average, 1,077 unique students have used *dcc --help* every week, with a mean of 60 uses per student and a median of 38 uses per student. Overall, 93% of our 2,565 CS1 and CS2 student cohorts have used the tool at least once during the teaching period. Usage peaked during weeks prior to major assessment due dates. So far, *dcc --help* has generated a total of 64,119 explanations in just ten weeks, with 49,866 compile-time, and 14,253 run-time explanations. We also recorded the time of day when students engaged the tool, and present brief reflections in subsection 6.1.

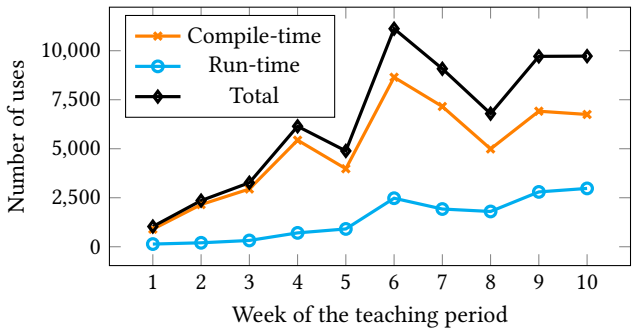

Figure 2: Weekly Usage of *dcc --help* Over a Teaching Period

6 DISCUSSION

Results are promising, and *moderate* to *substantial* index of agreement between the four reviewers validate the findings. We show that the use of LLMs to generate explanations of compiler error messages to augment compiler output in *dcc --help* is feasible when sufficient information such as error and stack trace is provided.

Our results indicate that LLM-generated explanations perform better at generating compile-time error explanations than run-time. We believe this is due to the increased context requirements at run-time—here, error explanations need to incorporate the *state* of the

program. We find that the LLM often provides correct conceptual explanation for these problems, but can fall short on smaller technical details, for instance occasionally mis-identifying bug lines. Despite this, we overall find that the explanations are helpful in solving code-related issues, with error descriptions and analysis commensurate with a junior member of our teaching team.

ChatGPT also had a propensity to disregard our prompts instructing it to not solve the problem and output solution source code. It is not entirely clear why this occurs, however, this was not pedagogically harmful. Future work may address this limitation using simple post-processing of results to remove code blocks, or prompt engineering for better LLM instruction.

6.1 Reflections and Observations

We have seen overwhelming adoption of the tool amongst our students, with consistent growth in usage since its introduction shown in Figure 2. We observed significant engagement in weeks preceding a major assessment (week 4 and 6), indicating students were turning to the tool. Overall, 47% of *dcc --help* use occurs between the hours of 18.00 and 08.00, when teaching assistance is not readily available. This highlights a key advantage of the tool – a method for student assistance outside staff office hours.

Cursory evaluation of performance in the invigilated, closed-book final exam in which *dcc --help* was not available indicates no significant performance loss compared to previous terms.

6.2 Too much help?

Following introduction of *dcc --help* into our computing courses, questions and discussions naturally arose. Firstly, should *dcc --help* be made available to more students? Secondly, and on the flip side, does the tool in its current form provide too much assistance? While these are open questions, we acknowledge that regardless of our choices, students can access their own explanations using ChatGPT themselves. We believe that providing access to this tool outweighs the potential risks, especially as we designed the tool with limits (such as rate-limiting explanations with warnings), and removed its availability from the final exam.

dcc --help also affords us the opportunity to identify and discuss with our students limitations of generative AI tools, especially when the tools provide incorrect explanations. Ingraining a scepticism-first approach provides meaningful learning opportunities for students, ensuring that even when seeking assistance from *dcc --help* they should always have a clear understanding of their goals and code when debugging. In many cases, we observed that the benefits of *dcc --help* were the clear and friendly language of the generative explanations, particularly at compile-time – reinterpretations of cryptic error messages could lead students to an understanding of their errors and successful resolutions thereof.

Overall, the use of LLMs to generate contextual explanations for compiler errors provides a way to scaffold existing student support. The automated nature of the tool benefits the scale of learning that we face. Whilst at some stage, those scaffolds need to be removed [29] and students need to understand compiler error messages on their own, they need not be expected to do this at the start of their learning journey – and especially not in languages such as C. All cognitive focus should be on learning the programming language, as opposed to interpreting cryptic error messages provided by the compiler at the introductory level.

7 LIMITATIONS AND FUTURE WORK

Whilst the four reviewers assessed 15 errors together to form a consensus of interpretation of the task, different interpretations of the categories may have impacted the reliability due to the subjective nature. Still, we observe an overall *moderate* to *substantial* agreement in all classifications.

Currently, we have not assessed students' interpretations of generative explanations, nor the tool's efficacy in assisting students to understand and solve errors. Human research ethics approval has been obtained to explore this in the future.

Integrating OpenAI's *gpt-3.5-turbo-0301* into DCC introduces costs which may impact scalability, however the API is affordable, costing $104 USD to generate over 64,000 explanations.

Finally, OpenAI's newly released GPT4-based models claim improved performance in many contexts. Future work could explore the performance of these models in this context, and compare any benefits to the increased associated costs. Finally, there is considerable scope for alternative prompt formations, including, for example, the addition of compiler-tooling specifically to provide extra information for run-time error prompts to potentially improve the quality of run-time explanations.

8 CONCLUSION

Our open-source *dcc --help* tool presents a promising avenue for deploying Large Language Models (LLMs) to generate controlled, novice-focused compiler error messages directly in the development environment. We found LLM-generated explanations conceptually accurate in the majority of cases, and the popularity of the tool with our CS1 and CS2 cohorts at a large Australian university demonstrates student acceptance. Future work is required to evaluate the tool's usefulness from the student perspective. Initial reflections and observations, including a spike in the tool's use immediately before major assessment deadlines suggest students are choosing to continue engaging with the tool. Integrating generative explanations into the compiler allows us to scaffold novice students with contextual guidance the moment an error occurs, ensuring students can act upon the explanations. In our experiences, *dcc --help* has transformed the role of the compiler from a tool that continually repeats frustrating, unhelpful or cryptic error messages to a student-focused *guide by the side* – always available for those times when a student just needs a little bit of help.

9 ACKNOWLEDGEMENTS

The authors would like to thank Lorenzo Lee Solano and other contributors to the open-source DCC-help project. Author Hammond Pearce is supported in part by a gift from Intel Corporation.

REFERENCES

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732*. (8 2021). http://arxiv.org/abs/2108.07732

[2] Gagan Bansal, Tongshuang Wu, and Joyce Zhou. 2021. Does the whole exceed its parts? The efect of ai explanations on complementary team performance. In

Conference on Human Factors in Computing Systems - Proceedings. Association for Computing Machinery, 1–16. https://doi.org/10.1145/3411764.3445717

[3] Titus Barik, Jim Witschey, Brittany Johnson, and Emerson Murphy-Hill. 2014. Compiler error notifications revisited: An interaction-first approach for helping developers more effectively comprehend and resolve error notifications. In *36th International Conference on Software Engineering, ICSE Companion 2014 - Proceedings*. Association for Computing Machinery, 536–539. https://doi.org/10.1145/2591062.2591124

[4] 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 *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 500–506. https://doi.org/10.1145/3545945.3569759

[5] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter Michael Osera, Janice L. Pearce, and James Prather. 2019. Compiler error messages considered unhelpful: The landscape of text-based programming error message research. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*. Association for Computing Machinery, 177–210. https://doi.org/10.1145/3344429.3372508

[6] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter Michael Osera, Janice L. Pearce, and James Prather. 2019. Unexpected tokens: A review of programming error messages and design guidelines for the future. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*. Association for Computing Machinery, 253–254. https://doi.org/10.1145/3304221.3325539

[7] Brett A. Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin, and Catherine Mooney. 2016. Effective compiler error message enhancement for novice programming students. *Computer Science Education* 26, 2-3 (7 2016), 148–175. https://doi.org/10.1080/08993408.2016.1225464

[8] Brett A. Becker, Kyle Goslin, and Graham Glanville. 2018. The effects of enhanced compiler error messages on a syntax error debugging test. In *SIGCSE 2018 - Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, Vol. 2018-January. Association for Computing Machinery, Inc, 640–645. https://doi.org/10.1145/3159450.3159461

[9] Jens Bennedsen and Michael E Caspersen. 2019. Failure Rates in Introductory Programming: 12 Years Later. *ACM inroads* 10, 2 (2019), 30–36.

[10] Gabriel Carvalho, Vinicius Ramos E. Cristian Cechinel, Juary Costa Rocha, Anabela Gomes, and Antonio Jose Mendes. 2021. Enhanced compiler messages of error in Python with focuses in readability in CS1. In *Proceedings - 2021 16th Latin American Conference on Learning Technologies, LACLO 2021*. Institute of Electrical and Electronics Engineers Inc., 389–396. https://doi.org/10.1109/LACLO54177.2021.00048

[11] 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. https://doi.org/10.48550/arXiv.2107.03374 arXiv:2107.03374 [cs].

[12] 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

[13] Paul Denny, Andrew Luxton-Reilly, and Dave Carpenter. 2014. Enhancing syntax error messages appears ineffectual. In *ITICSE 2014 - Proceedings of the 2014 Innovation and Technology in Computer Science Education Conference*. Association for Computing Machinery, 273–278. https://doi.org/10.1145/2591708.2591748

[14] Paul Denny, James Prather, and Brett A. Becker. 2021. On designing programming error messages for novices: Readability and its constituent factors. In *Conference on Human Factors in Computing Systems - Proceedings*. Association for Computing Machinery, 1–15. https://doi.org/10.1145/3411764.3445696

[15] Emily Dreibelbis. 2023. Harvard's New Computer Science Teacher Is a Chatbot. *PCMag Australia* (June 2023). https://au.pcmag.com/ai/100574/harvards-new-computer-science-teacher-is-a-chatbot

[16] 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

[17] Sajed Jalil, Suzzana Rafi, Thomas D. LaToza, Kevin Moran, and Wing Lam. 2023. ChatGPT and Software Testing Education: Promises & Perils. In *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 4130–4137. http://arxiv.org/abs/2302.03287

[18] Ioannis Karvelas, Annie Li, and Brett A. Becker. 2020. The effects of compilation mechanisms and error message presentation on novice programmer behavior. In *SIGCSE 2020 - Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM, 759–765. https://doi.org/10.1145/3328778.3366882

[19] 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 (April 2023), 102274. https://doi.org/10.1016/j.lindif.2023.102274

[20] Tobias Kohn. 2019. The error behind the message: Finding the cause of error messages in python. In *SIGCSE 2019 - Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. Association for Computing Machinery, Inc, 524–530. https://doi.org/10.1145/3287324.3287381

[21] J Richard Landis and Gary G Koch. 1977. An Application of Hierarchical Kappa-type Statistics in the Assessment of Majority Agreement among Multiple Observers. *Biometrics* 33, 2 (1977), 363–374. https://www.jstor.org/stable/2529786

[22] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker. 2023. Using Large Language Models to Enhance Programming Error Messages. In *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 563–569. https://doi.org/10.1145/3545945.3569770

[23] Courtney Linton, Printon Avia, and James Tan. 2023. ChatGPT response by Australian Schools - Corney & Lind Lawyers. https://www.corneyandlind.com.au/education-law/chatgpt-policy/

[24] Stephen MacNeil, Andrew Tran, Dan Mogil, Seth Bernstein, Erin Ross, and Ziheng Huang. 2022. Generating Diverse Code Explanations using the GPT-3 Large Language Model. In *ICER 2022 - Proceedings of the 2022 ACM Conference on International Computing Education Research*, Vol. 2. Association for Computing Machinery, Inc, 37–39. https://doi.org/10.1145/3501709.3544280

[25] Nicholas Nethercote and Julian Seward. 2007. Valgrind. *ACM SIGPLAN Notices* 42, 6 (6 2007), 89–100. https://doi.org/10.1145/1273442.1250746

[26] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 27730–27744. https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf

[27] Raymond Pettit, John Homer, and Roger Gee. 2017. Do enhanced compiler error messages help students? Results inconclusive. In *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*. Association for Computing Machinery, 465–470. https://doi.org/10.1145/3017680.3017768

[28] James Prather, Raymond Pettit, Kayla Holcomb Mcmurry, Alani Peters, John Homer, Nevan Simone, and Maxine Cohen. 2017. On novices' interaction with compiler error messages: A human factors approach. In *ICER 2017 - Proceedings of the 2017 ACM Conference on International Computing Education Research*. Association for Computing Machinery, Inc, 74–82. https://doi.org/10.1145/3105726.3106169

[29] John Sweller, Jeroen J.G. van Merriënboer, and Fred Paas. 2019. Cognitive Architecture and Instructional Design: 20 Years Later. *Educational Psychology Review* 31, 2 (6 2019), 261–292. https://doi.org/10.1007/s10648-019-09465-5

[30] 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

[31] V. Javier Traver. 2010. On compiler error messages: What they say and what they mean. *Advances in Human-Computer Interaction* 2010 (2010), 1–26. https://doi.org/10.1155/2010/602570

[32] Helena Vasconcelos, Matthew Jörke, Madeleine Grunde-McLaughlin, Tobias Gerstenberg, Michael S. Bernstein, and Ranjay Krishna. 2023. Explanations Can Reduce Overreliance on AI Systems During Decision-Making. *Proceedings of the ACM on Human-Computer Interaction* 7, CSCW1 (4 2023), 1–38. https://doi.org/10.1145/3579605