Generating Programs Trivially: Student Use of Large Language Models

Siddhartha Prasad
siddhartha@brown.edu
Brown University
Providence, RI, USA

Ben Greenman
blg@cs.utah.edu
Brown University
Providence, RI, USA

Tim Nelson
tbn@brown.edu
Brown University
Providence, RI, USA

Shriram Krishnamurthi
shriram@brown.edu
Brown University
Providence, RI, USA

ABSTRACT

Educators have been concerned about the capability of large language models to automatically generate programs in response to textual prompts. However, little is known about whether and how students actually use these tools.

In the context of an upper-level formal methods course, we gave students access to large language models. They were told they could use the models freely. We built a Visual Studio Code extension to simplify access to these models. We also paid for an account so students could use the models for free without worrying about cost.

In this experience report we analyze the outcomes. We see how students actually do and do not use the models. We codify the different uses they make. Most of all, we notice that students actually do not use them very much at all, and provide insight into the many reasons why not. We believe such experiments can help rebalance some of the public narrative about such tools.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**.

KEYWORDS

large language models, formal methods, properties, testing

ACM Reference Format:
Siddhartha Prasad, Ben Greenman, Tim Nelson, and Shriram Krishnamurthi. 2023. Generating Programs Trivially: Student Use of Large Language Models. In *Proceedings of the ACM Conference on Global Computing Education Vol 1 (CompEd 2023), December 5–9, 2023, Hyderabad, India.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3576882.3617921

1 INTRODUCTION

In recent years, the easy availability of large language model-based tools, such as Copilot [6] and ChatGPT [26], has caused significant consternation. Educators, researchers, bloggers, social media influencers, opinion piece authors, and others have expressed concern about the consequences for programming and, especially, for programming education [4, 5, 35, 36, 38, 41]. Indeed, it would appear that such tools can easily generate correct solutions to conventional CS1 programming problems [12, 14].

Unfortunately, these concerns have not been matched by an understanding of what students actually do. At best we hear anecdotal evidence such as when students ask for help or when they are caught for plagiarism. However, this does not provide insight into student practices while working on an assignment. For instance, students could use an LLM to obtain a purported solution, then massage it so that it avoids detection.

From February to April 2023, we gave students *unlimited* access to an LLM. We built a plugin for Visual Studio Code so that students could access the LLM without having to leave their programming buffer. We also paid for access, so that no student would be restricted by financial circumstances, or feel forced to choose how to use the free credits given during part of this time by OpenAI. Students were not required to use the plugin, but were told they could make full and free use of it and could directly turn in solutions generated by it just so long as they provided attribution.

Our specific setting was an upper-level (primarily for third- and fourth-year) post-secondary students in an introductory formal methods course. The setting makes the task more interesting. First, the problems are not typical CS1 problems and many use a variant of the Alloy modeling language. Although prior work shows that LLMs can succeed on harder problems and in languages other than Python [7, 15], success is by no means guaranteed. Second, there is a mix of activities in the class (sometimes writing programs, sometimes writing tests, and sometimes writing specifications); each could fare differently under an LLM.

Our goal was to determine the impact of an LLM on the course. If students could make effective use of an LLM, it could potentially alter the nature and content of the course. For instance, it might enable the course to tackle more difficult, real-world problems. In particular, if LLMs made the course trivial (as they threaten to do in traditional CS1 courses), we felt it essential to rethink the course. In addition, we assumed that some students would use an LLM anyway. Instead of pretending it would not happen, we wanted to level the playing field and make the same resources available to all.

This paper examines how students *actually* used the LLM. We see a large amount of initial attention that is not matched by later use. We identify the kinds of tasks for which students tried an LLM, and examine how well it did. Most of all, we believe that our findings are a small antidote to the current panic about LLMs: they have not made this course trivial (for now), and students are not rushing to use it the way some commentators assume they would.

Terminology: In this paper, we use "LLM" to refer to tools based on large language models. Not all of these tools may be built atop

CompEd 2023, December 5–9, 2023, Hyderabad, India

ACM ISBN 979-8-4007-0048-4/23/12…$15.00
https://doi.org/10.1145/3576882.3617921

such models in the narrowest sense. However, these details are not relevant to this paper.

2 RELATED WORK

LLMs are an emerging technology and the literature on how programmers interact with them is in its early stages (see, e.g., [18, 34]). Several works study Copilot. Barke et al. [2] discover two modes of interaction with Copilot: exploring a new domain and accelerating the implementation of a plan. Prather et al. [31] study CS1 students' first interactions with Copilot and suggest ways to integrate the LLM into a curriculum. Mozannar et al. [25] propose a taxonomy for LLM interactions and use it to guide a study of Copilot use on a 20-minute coding task. Vaithilingam et al. [40] find that programmers are enthusiastic about Copilot but do not observe any productivity boosts on sample tasks. Xu et al. [43] report similar results for a different tool [42]; programmers are enthusiastic about the LLM but equally productive without it.

Jayagopal et al. [20] contrast six LLM tools by assigning small tasks to student participants. Ross et al. [33] integrate Codex in an IDE and study its use during roughly hour-long sessions. Kazemitabaar et al. [22] build an editor that can use Codex to generate code and test how access to the LLM affects novices' scores on code-camp modules. Codex often, but not always, leads to a significant improvement. None of these works allow the long-term, free-form student use that we do, which has very high ecological validity. McNutt et al. [24] and Robe et al. [32] discuss potential interfaces for LLM tools. Participants did not use an existing tool.

Jiang et al. [21] observe programmers who solved two tasks over one week; they find several issues related to prompt engineering. While this study does allow free-form use, it takes place over a relatively short period and its participants were volunteers. By contrast, our study includes over two months of data on students' actual coursework (indeed, our conclusions would be *much different* if we had stopped after 1 week!). Vaithilingam et al. [39] redesign the user interfaces of Visual Studio IntelliCode for writing code and for editing one line. They report a 350 % increase in "regular users" of the writing feature over an unspecified period (perhaps Nov'22 to Jan'23) and a 29 % increase for the editing feature over two weeks for a preview release. Our semester-long study raises questions about whether this increase was sustained over time, as we have observed major fluctuations.

3 COURSE CONTEXT

We introduced an LLM tool in Logic for Systems (L4S), an upper-level course on applied logic and formal methods. The course took place in Spring 2023 at a highly selective, private university in the USA. Students taking the course had prior experience comparable to CS1 and CS2. Students were not required to know logic or formal methods; indeed, the purpose of the course is to give a lightweight, application-driven introduction to these topics.

L4S is built on three learning categories: modeling systems and designing abstractions; reasoning about systems using logic; and studying the algorithms that power verification tools. Assignment topics include property-based testing, solvers for river-crossing puzzles, and garbage collection. The course is primarily taught in Forge [9], a pedagogic variant of Alloy [1, 19]. Concretely, Forge

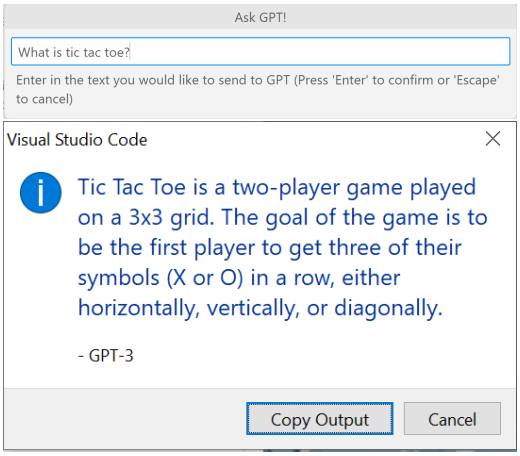

Figure 1: AskGPT textbox interface and example output.

```
1  What is tic tac toe?
2    GPT Responded:
3  Tic Tac Toe, also known as Noughts and Crosses, is
   a two-player game played on a 3x3 grid. The
   objective of the game is to get three of your marks
   in a row (horizontal, vertical, or diagonal) before
   your opponent does. Players alternate turns placing
   their marks on the grid. The first player to get
   three of their marks in a row wins the game.
```

Figure 2: AskGPT inline code selection and example output.

programs use a Java-like syntax to describe systems and their properties, and the Forge runtime uses a SAT solver to search for counterexamples. The course also uses Python for several assignments, e.g., to study solver implementation. There are 8 assignments in total (5 Forge, 3 Python), 4 major projects (all Forge), and 9 labs (6 Forge, 3 Python).

In Spring 2023, the class had 64 students. Student experience levels varied: 8 % were graduate students, 11 % were first-year students, 9 % were second-year, 33 % were third-year, and 39 % were fourth-year. Due to privacy considerations, we unfortunately cannot match this experience against LLM usage and perceptions.

4 ASKGPT: A VS CODE EXTENSION

VS Code is the standard editor in L4S. We therefore built a VS Code extension called AskGPT to give students direct access to an LLM. AskGPT supports two kinds of interaction:

- **textbox** (fig. 1) Users may insert text into a single-line box after pressing a button on the VS Code status background or entering a keyboard shortcut. The response from the LLM appears in a modal dialog with a copy-to-clipboard button.
- **inline** (fig. 2) Users may send currently-selected text to the LLM by entering a keyboard shortcut. The response appears inline, right below the prompt.

The extension logged every student prompt and LLM response to a database. Students were made aware of this logging (and the fact that OpenAI logs as well) in an agreement form (section 5).

AskGPT does not ask students to rate the quality of responses. Instead, we infer the usefulness of responses from logs. We were doubtful about ratings for two reasons:

(1) We might not receive enough data to make statistically sound claims about response quality.
(2) Ratings taken immediately after querying the extension reflect only an initial impression of response quality. In particular, an LLM response might appear correct but may actually have subtle bugs [8]. Collecting ratings at a later time is also problematic, but for different reasons. Users might forget the interaction, and they might find the rating process a frustrating interruption to their work.

5 EXPERIMENTAL SETUP

Student Instruction. A week into the semester (which has about 13 weeks of instruction), a graduate teaching assistant gave a half-hour lecture on LLMs, course policies, and the AskGPT extension. After class, students received a link to an agreement form about the course policies on LLMs. Students were required to complete the form. Automatically upon completion, students received an email with access tokens and information on how to use AskGPT. The course syllabus forbade all other LLM use.

Later that same week, a 2-hour lab presented a guided tour of AskGPT. Participants used AskGPT to play tic-tac-toe with the LLM and to specify the rules of tic-tac-toe.

Beyond this one lecture and lab, students received no additional instruction on AskGPT, LLMs, or prompt engineering. This was intentional: our goal was to see what prompts students write in a relative state of nature. (Of course, some students may have had independent experience with, say, ChatGPT.)

AskGPT Settings. By default, AskGPT used `text-davinci-003`, the "most capable" (and most expensive) model available in Feb. 2023. It was also the only model that produced quality Forge code during our pre-testing. The default query temperature [27] was 0.3. Students were free to modify these defaults.

In March, OpenAI released model `gpt-3.5-turbo` [29]. We posted instructions for how to access the model on the course discussion forum. Switching to this did not require any new software update; the new model was available from the AskGPT settings menu. Despite the availability of the settings menu and the announcement, however, no students updated any defaults according to our records. This is unsurprising in light of the usage logs.

6 USAGE

Between February 1 and April 14 2023 (when we ended the experiment), students sent 293 prompts to AskGPT. Figure 3 presents the number of prompts per day. In the background, fig. 3 shows a timeline of labs, course projects, and assignments (the vertical arrangement of these elements has no semantic meaning) and whether these milestones used Forge or Python.

The majority of prompts occur near the beginning; specifically, during the tic-tac-toe lab that introduced AskGPT. Afterward, usage drops to single-digit numbers except for four minor spikes of about 20 prompts each. The biggest minor spike appears in early April, when the university was on Spring break. The prompts are from one student and are unrelated to coursework.

Table 1: Counts of prompt type and response medium.

Prompt Type	Response Medium Spec.	Code	Text
Write formal specification	72	8	11
Unrelated to coursework			55
Explain coursework concept	3	1	39
Write text specification			33
Write code		18	3
Play tic-tac-toe			13
Explain specification			12
Prompt unclear		1	8
Explain programming concept		3	5
Prompt unclear (spec. only)	1		2
Explain error message			3
Explain program		1	1

Based on generous but reasonable estimates of maximum student use, we had allocated USD 2,500 for student AskGPT use. We expected modest use throughout the semester and increased use during one midterm project and the final project, as these tasked students with exploring how to model a domain of their choosing. In reality, students rarely used AskGPT. Our OpenAI bill for the length of the study totaled USD 1.16 [sic].

Codebook. Using techniques from grounded theory [17], two coders analyzed a sample of student AskGPT interactions and developed a rubric for labeling them (overall Cohen κ score: 0.86). The rubric included three axes:

Prompt Type (Cohen κ: 0.90) What sort of question did the student ask? Choices include: write specification, explain specification, and play tic-tac-toe.

Response Medium (Cohen κ: 0.91) What sort of response did the LLM give? There are four choices: formal specification, program, text, and other.

Response Relevance (Cohen κ: 0.80) Is the response relevant to the prompt? Choices are: yes, no, LLM declined to respond, and other. We did not use the other category.

Student Prompts and Responses. Once we had developed these rubrics, one coder analyzed the full dataset of student interactions. Responses from the LLM were generally relevant to student prompts (86 % of the time). Table 1 presents the relationship between prompt type and response medium. We summarize the data below:

- The majority of student interactions (81 %) with AskGPT were related to coursework.
- The most common prompt type (31 %) was to write a formal specification. AskGPT responded with a relevant formal specification 78 % of the time.
- 43 prompts (15 %) asked to explain assignment-related concepts. Nearly all (91 %) responses to these were relevant.
- 33 prompts (11 %) asked for a text specification. Nearly all (97 %) responses were relevant.
- 21 prompts (7 %) asked for a program. The responses were often relevant (76 %).
- 12 prompts (4 %) were unclear or lacked a objective.

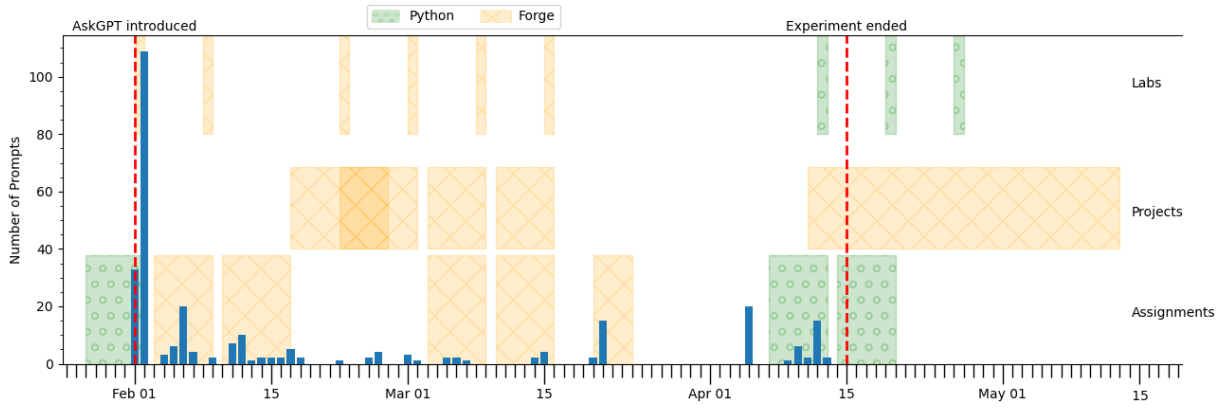

Figure 3: Number of student prompts to AskGPT per day, overlaid on an course schedule.

7 WHY DIDN'T STUDENTS USE ASKGPT?

The extremely low volume of usage logs shows that students rarely used the AskGPT extension. To learn more, we created a retrospective survey asking students about AskGPT and LLMs more generally. The survey did not disclose that usage was far lower than we had expected; it simply asked for details about where AskGPT was and was not useful.

During lecture on April 21 (after ending the experiment), the instructor explained the survey and gave students ample time to complete it. Students could also submit for a few days afterward (this was important for those who happened to be absent). The survey was described as mandatory. In total, 52 students (81 %) completed the survey. Because students received course credit for submitting it, some students may have felt the credit was insufficient to do the work, or may have simply missed it. Students received credit irrespective of the quality of their response, and students who did not submit were not penalized beyond the loss of credit.

There were 25 questions on the survey, half of which asked yes/no questions about whether AskGPT was useful on specific course milestones. Only one question was required. For us, the most interesting question is the following optional question:

> **Q.** If you did not use the extension often, please tell us why? *Optional, select all choices that apply.*

Table 2 summarizes the responses. We use these as the starting point for further discussion in the paragraphs below. The discussions incorporate responses from the entire survey, not just the question above. In particular, only two students checked the box for "Concerns about AI" but a few others demonstrated concerns in their responses to other questions.

May Interfere with Learning. Over half the students who took the survey felt that using AskGPT would interfere with their learning. Students were given the option of providing a more detailed answer. One student wrote:

> P52: "*A large part of what I enjoy in CS is figuring out solutions and bugs on my own*"

Another responded:

Table 2: Reasons students did not use AskGPT.

Reason	Count	% Responses
May interfere with learning	25	57 %
Fear of breaking course rules	12	27 %
Awkward UI	9	20 %
Concerns about logging	7	16 %
Worse than alternatives	6	14 %
Responses were not useful	6	14 %
Did not want to use	4	9 %
Fear of overuse	4	9 %
Installation issues	2	5 %
Concerns about AI	2	5 %
Did not use VS Code	1	2 %

> P2: "*I learn better through struggling to find an answer. If it comes too easy, it's hard for me to remember any information I've learned.*"

Fear of Breaking Course Rules. The course had explicitly allowed students to use AskGPT. Nevertheless, students seemed to worry about the consequences of doing so. However, students did not provide narrative responses that could help us understand their concern. The closest we got was this comment:

> P51: "*I didn't know in what capacity it was meant to be used.*"

Awkward UI. Many students said that they did not use AskGPT because the interface is awkward. The usage logs suggest three reasons for the awkwardness:

(1) *Unfamiliarity with LLMs*: Responses from LLMs have a certain style that takes getting used to. They usually present a lot of text with a few subtle errors mixed in. Spotting the errors is a bit of an art that takes practice.
(2) *Lack of conversation context*: AskGPT has no memory of past interactions. It accepts one prompt, responds, and forgets the interaction. This is in contrast to ChatGPT, which uses prior interactions to inform its responses (P45: "*a huge feature*"). The closest work-around in AskGPT is to include

the text of past interactions in the current prompt. Though the intro lab (section 5) gave a demo, the work-around is unwieldy and students may not have remembered it.

(3) *Lack of code context*: In inline mode (fig. 2), AskGPT bases its responses on currently-selected text. It does not see the rest of the file that contains the text, nor the rest of the codebase. Furthermore, it does not know that the text is being written for a L4s assignment. Relevant details must appear in the prompt; AskGPT does not insert a preamble [10, 33].

The lack of code context is the most significant roadblock. For example, the prompt "What does one mean?" was asking about a Forge keyword called one, but received a long-winded definition of the word "one" in English. Other students summarized the issue:

> P35: "*It seemed like assignments were specific enough that GPT wouldn't be able to provide something useful.*"

> P18: "*I have to give too much context for it to be helpful.*"

A few students commented that Copilot is easier to use than AskGPT because it works well in-flow. Prior studies confirm the importance of in-flow AI suggestions [24, 39]. However, we do not know the context in which our students used Copilot. Course policy forbade the use of any LLM tools other than AskGPT. (We chose not to investigate whether these survey comments were indeed because of policy violations.)

Despite the awkwardness of AskGPT, few students (10 %) reported any familiarity with other LLM tools. Of course, this number could be heavily underreported because course policy forbade the use of such tools on coursework.

It is also worth noting a sense in which AskGPT is *less* awkward than tools like ChatGPT: its direct integration into the IDE. Students do not have to switch to a browser, deal with distractions (such as notification badges) from other tabs, lose their sense of spatial familiarity with their code, etc. However, these effects do not seem have been very important.

Concerns about Logging. Students were aware that we were logging their work in AskGPT, and a few were concerned about this:

> P51: "*I have concerns about having interactions logged.*"

Other Concerns. Students expressed a variety of other concerns, which are summarized in table 2. These included:

Alternatives As noted above, some students found AskGPT unattractive to use. Presumably these students had gotten used to the interaction styles of tools like ChatGPT and Copilot. AskGPT is less sophisticated than either of these.

Utility Usage logs substantiate these concerns; see table 1.

Undesired This is related to the interference with learning.

Overuse Some students were concerned about the cost incurred by using AskGPT. Our agreement form had warned students about runaway costs from overuse, which may have caused anxiety for some students. In retrospect, there was nothing to worry about (section 6). Students may have also been concerned about the overall environmental impact of LLMs, though none said so directly in the survey.

AI Students were simply concerned or unhappy about AI:

> P11: "*I have a vitriolic hatred toward AI.*"
> P17: "*I didn't want AI doing my homework. I also didn't feel like it was trustworthy anyway, and I didn't want to waste time debugging its answers.*"
> P39: "*Really never felt like I would benefit from using it. Usually never even crossed my mind to use it, and if it did, I would rather go to office hours and understand from a person than from ChatGPT.*"
> P38: "*I'm not generating lots of "boiler plate" code in this course. Every line is carefully crafted and AI models just don't get the details right enough to make it easier to use AskGPT. Some sort of Forge syntax autocomplete would be much more helpful. . . . it isn't worth my time debugging GPT output when I could have just written the line*"

Installation Some students used older versions of VS Code.

No VS Code One student used Emacs instead.

8 OTHER STUDENT OPINIONS ON LLM USE

The final two survey questions asked for general comments on AskGPT and its use in the course. Both were optional:

Q. Do you have any other thoughts on the use of AskGPT?

Q. If you were in charge of [this course], how would you use tools like AskGPT?

Several students did identify positive uses and a few suggested ways to refine its use. Three wanted to *prohibit* its use.

Keep It. In the retrospective survey, students listed several situations in which they found the tool useful:

- generating code (N=9)
- generating creative ideas or inspiration (N=6)
- improving code quality (N=3)
- understanding existing code (N=3)

Though these numbers are small, they do reflect the ways in which programmers have been employing LLMs. These quotes shed additional light:

> P52: "*Normally they were questions I would just Google ("how do I add a key value pair to a dict in Python") but instead I could get the answer without leaving VS Code and I wouldn't have to sort through (possibly too verbose) results myself.*"

> P16: "*It greatly helped with getting familiar with Forge's syntax. It helped not having to write simple things.*"

> P14: "*GPT really improved code quality. If I used it after I wrote my implementation of the assignment, it could provide me with more intuitive or simpler ways to solve the algorithm.*"

Some students also played with the LLM, asking it to generate jokes or social security numbers (it refused the latter ask, despite several attempts). These activities are not only frivolous, technically they also violate the course's usage policy, which said that AskGPT should only be used for course-relevant queries. (We did not, however, penalize the students for this in any way.) However, we should consider the value of being able to have a little fun within the IDE

while doing work. Perhaps *all* IDEs should have a joke generator that a student can turn to for a little stress relief!

Refine It. It is now understood (e.g., [7, 10, 13, 21, 44]) that to make effective use of LLM tools, we need to develop a new kind of computational literacy: prompt engineering. This was not taught in this course. Multiple students asked for some basic instruction:

> P45: "*One main reason I did not use it for this class is because I did not have a structured way of learning what the tool might be capable of.*"
>
> P12: "*I think it's a good idea to provide students with access and encourage them to learn to use the tool if they're interested, but maybe more structured guidance on how it can be helpful would be useful.*"
>
> P10: "*I would make an assignment just for AskGPT, and require that all other assignments do not use it. I am scared of breaking course rules, so the expectations will be clearly set in this format.*"

Prohibit It. A few students wanted the tool eliminated entirely:

> P11: "*I think it provides a crutch in these bottled examples that will get people too comfortable an unable to think on their feet in real, active coding situations.*"
>
> P19: "*Students should be submitting their own work, not that of a LLM.*"

9 VALIDITY & GENERALIZABILITY THREATS

As the paper discusses (section 7), there are many issues such as the quality of the AskGPT interface or desire to not be logged that could have resulted in students using other LLM tools in ways that we could not observe.

As we note in section 8, LLMs engender a new skill: prompt-engineering. Informally, we see that our students have some difficulty in designing good prompts. As students improve at writing prompts and obtaining quality responses, they may be much more likely to use an LLM.

Any results in the LLM space are very dependent on the exact models in use. Though revolutionary for their time, GPT-3 and GPT-3.5 are already significantly weaker than the claimed powers of GPT-4 [28]. There are also models more specialized to individual tasks such as programming, and a growing interest in LLMs Modulo Theories [16, 37], by analogy to Satisfiability Modulo Theories [3, 11]. The performance of those models could well change how and how much students use LLMs.

Nevertheless, this work has extremely high ecological validity. We studied students in an actual course over a good portion of a semester. Students were given unfettered use in multiple ways: integration into their IDE, free-of-cost access, and the freedom to use the results directly in their homework. Furthermore, the class had many kinds of activities (section 3), including tasks for which LLMs are generally considered quite good, such as writing Python and explaining errors [7, 23].

Generalizability is naturally weak because we studied only one course offering. Besides the specific factors of that student body, we also used an upper-level course in formal methods. Naturally, there is no reason to believe the findings of this work will generalize to, say, a CS1 course. We have not taken any steps to be able to generalize our knowledge; that is why we submit this work as an *experience report*. Rather, we believe this is useful preliminary information, and encourage other educators to try the same experiment in their classes. Our software will be available for them to use.

However, the above caveats must be put in context. Computing education is much broader than just CS1. Furthermore, our findings still have use as a view into *student attitudes*: e.g., students saying that they want to get an education or want to talk to humans for help are relatively independent of the course level or topic.

10 DISCUSSION

Much ink has been spilt on the effect of LLMs. We have little novel to add to that discussion. We simply note that student use of LLMs may be governed by two opposing forces. On the one hand, competition for jobs may cause students to feel they must have "perfect" transcripts, which can be aided by leaning on an LLM. On the other, students may realize that getting an attractive job is hard, and decide they need to learn more in order to pass interviews and perform well to retain their positions.

A seemingly minor issue that we think has consequences from an equity perspective is the cost of using LLMs. What might seem like a modest or even trivial amount to a well-paid professional programmer may be a prohibitive amount to a poor student. Of course, these inequities are already rampant in the system: one student can afford to pay someone else to do their work for them, but another can't. It is important to consider the ways in which they will get further entrenched with paid subscriptions for LLMs, and how educational institutions might try to mitigate them.

11 DATA COLLECTION AND AVAILABILITY

Our data came from two sources: AskGPT logs (section 6) and a retrospective survey (section 7).

Our Institutional Review Board (IRB) did not consider this work human subjects research because the logs gathered information about the use of the AskGPT tool, not about students themselves. Nevertheless, to use the extension, students had to sign an agreement form that informed them that we were tracking identity and use. At the end of the semester, student logs were deidentified with random, persistent IDs. All analysis was carried out on these anonymized logs.

Similarly, our IRB did not consider the retrospective survey as human subjects research because the students were considered "key informants" about the AskGPT tool. Nevertheless, we ensured the privacy and confidentiality of the data. While the survey software collected student email addresses, this information was anonymized before conducting the paper's analysis.

The AskGPT extension, initial agreement form, relevant syllabus excerpts, and full codebooks (section 6) are available online [30].

ACKNOWLEDGMENTS

Thanks to Conrad Zimmerman, David Fryd, and Megan Frisella for suggesting AskGPT keybindings. Thanks to Anna Fariha for comments on an early draft. This work was partially supported by the US National Science Foundation grant SHF-2227863 and grant 2030859 to the CRA for the CIFellows project.

REFERENCES

[1] Alloytools. 2023. *Alloy Analyzer Website.* https://alloytools.org

[2] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *PACMPL* 7, OOPSLA (2023), 85–111. https://doi.org/10.1145/3586030

[3] Clark W. Barrett and Cesare Tinelli. 2018. Satisfiability Modulo Theories. In *Handbook of Model Checking*. Springer, 305–343. https://doi.org/10.1007/978-3-319-10575-8_11

[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*. ACM, 500–506. https://doi.org/10.1145/3545945.3569759

[5] Emery Berger. 2022. Coping with Copilot. https://blog.sigplan.org/2022/08/18/coping-with-copilot/

[6] Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. 2023. Taking Flight with Copilot. https://queue.acm.org/detail.cfm?id=3582083

[7] Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q. Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. 2023. MultiPL-E: A Scalable and Polyglot Approach to Benchmarking Neural Code Generation. *Transactions on Software Engineering* 49, 7 (2023), 3675–3691. https://doi.org/10.1109/TSE.2023.3267446

[8] 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:2107.03374 [cs.LG]

[9] Forge Contributors. 2023. *Forge: A Tool and Language for Teaching Formal Methods.* https://forge-fm.org

[10] Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. 2023. nl2spec: Interactively Translating Unstructured Natural Language to Temporal Logics with Large Language Models. In *CAV*. Springer, 383–396. https://doi.org/10.1007/978-3-031-37703-7_18

[11] Leonardo De Moura and Nikolaj Bjørner. 2011. Satisfiability modulo theories: introduction and applications. *Commun. ACM* 54, 9 (2011), 69–77. https://doi.org/10.1145/1995376.1995394

[12] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *SIGCSE*. ACM, 1136–1142. https://doi.org/10.1145/3545945.3569823

[13] Jean-Baptiste Döderlein, Mathieu Acher, Djamel Eddine Khelladi, and Benoît Combemale. 2022. Piloting Copilot and Codex: Hot Temperature, Cold Prompts, or Black Magic? *CoRR* abs/2210.14699 (2022), 14 pages. https://doi.org/10.48550/arXiv.2210.14699

[14] 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 *ACE*. ACM, 10–19. https://doi.org/10.1145/3511861.3511863

[15] 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 *Australasian Computing Education Conference*. ACM, 97–104. https://doi.org/10.1145/3576123.3576134

[16] Matt Fredrikson, Kaiji Lu, Saranya Vijayakumar, Somesh Jha, Vijay Ganesh, and Zifan Wang. 2023. Learning Modulo Theories. *CoRR* abs/2301.11435 (2023). https://doi.org/10.48550/arXiv.2301.11435 arXiv:2301.11435

[17] B. Glaser and A. Strauss. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Sociology Press.

[18] Austin Z. Henley. 2023. *Papers on the UX of AI programming assistants.* https://austinhenley.com/blog/uxaicoding.html

[19] Daniel Jackson. 2012. *Software Abstractions: Logic, Language, and Analysis* (2 ed.). MIT Press. https://doi.org/10.5555/2141100

[20] Dhanya Jayagopal, Justin Lubin, and Sarah E. Chasins. 2022. Exploring the Learnability of Program Synthesizers by Novice Programmers. In *UIST*. ACM, Article 64, 15 pages. https://doi.org/10.1145/3526113.3545659

[21] Ellen Jiang, Edwin Toh, Alejandra Molina, Kristen Olson, Claire Kayacik, Aaron Donsbach, Carrie J. Cai, and Michael Terry. 2022. Discovering the Syntax and Strategies of Natural Language Programming with Generative Language Models. In *CHI*. ACM, 386:1–386:19. https://doi.org/10.1145/3491102.3501870

[22] 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 *CHI*. ACM, 455:1–455:23. https://doi.org/10.1145/3544548.3580919

[23] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent N. Reeves, Paul Denny, James Prather, and Brett A. Becker. 2023. Using Large Language Models to Enhance Programming Error Messages. In *SIGCSE*. ACM, 563–569. https://doi.org/10.1145/3545945.3569770

[24] Andrew M. McNutt, Chenglong Wang, Robert A. DeLine, and Steven Mark Drucker. 2023. On the Design of AI-powered Code Assistants for Notebooks. In *CHI*. ACM, 434:1–434:16. https://doi.org/10.1145/3544548.3580940

[25] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2023. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. *CoRR* abs/2210.14306 (2023), 38 pages. https://doi.org/10.48550/arXiv.2210.14306

[26] Open AI. 2023. *Introducing ChatGPT.* https://openai.com/blog/chatgpt

[27] OpenAI. 2023. *API Reference: temperature.* https://platform.openai.com/docs/api-reference/completions/create#completions/create-temperature

[28] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023). https://doi.org/10.48550/arXiv.2303.08774

[29] OpenAI. 2023. *Introducing ChatGPT and Whisper APIs.* https://openai.com/blog/introducing-chatgpt-and-whisper-apis

[30] Siddhartha Prasad, Ben Greenman, Tim Nelson, and Shriram Krishnamurthi. 2023. Supplementary Material: Generating Programs Trivially. https://doi.org/10.5281/zenodo.8326445

[31] 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. *CoRR* abs/2304.02491 (2023), 26 pages. https://doi.org/10.48550/arXiv.2304.02491

[32] Peter Robe, Sandeep K. Kuttal, Jake AuBuchon, and Jacob Hart. 2022. Pair Programming Conversations with Agents vs. Developers: Challenges and Opportunities for SE Community. In *ESEC/FSE*. ACM, 319–331. https://doi.org/10.1145/3540250.3549127

[33] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael J. Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *IUI*. ACM, 491–514. https://doi.org/10.1145/3581641.3584037

[34] Advait Sarkar, Carina Negreanu, Ben Zorn, Sruti Srinivasa Ragavan, Christian Pölitz, and Andrew D. Gordon. 2022. What is it like to program with artificial intelligence?. In *PPIG*. 127–153. https://ppig.org/papers/2022-ppig-33rd-sarkar/

[35] Abdulhadi Shoufan. 2023. Exploring Students' Perceptions of ChatGPT: Thematic Analysis and Follow-Up Survey. *IEEE Access* 11 (2023), 38805–38818. https://doi.org/10.1109/ACCESS.2023.3268224

[36] Grace H. Sun and Stephanie H. Hoelscher. 2023. The ChatGPT Storm and What Faculty Can Do. *Nurse Educator* 48, 3 (2023), 119–124. https://doi.org/10.1097/NNE.0000000000001390

[37] Stefano Teso, Roberto Sebastiani, and Andrea Passerini. 2017. Structured learning modulo theories. *Artificial Intelligence* 244 (2017), 166–187. https://doi.org/10.1016/j.artint.2015.04.002

[38] Ahmed Tlili, Boulus Shehata, Michael Agyemang Adarkwah, Aras Bozkurt, Daniel T. Hickey, Ronghuai Huang, and Brighter Agyemang. 2023. What if the devil is my guardian angel: ChatGPT as a case study of using chatbots in education. *Smart Learning Environments* 10, 1 (2023), 24 pages. https://doi.org/10.1186/s40561-023-00237-x

[39] Priyan Vaithilingam, Elena L. Glassman, Peter Groenwegen, Sumit Gulwani, Austin Z. Henley, Rohan Malpani, David Pugh, Arjun Radhakrishna, Gustavo Soares, Joey Wang, and Aaron Yim. 2023. Towards More Effective AI-Assisted Programming: A Systematic Design Exploration to Improve Visual Studio IntelliCode's User Experience. In *SEIP@ICSE*. IEEE, 185–195. https://doi.org/10.1109/ICSE-SEIP58684.2023.00022

[40] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *CHI*. ACM, 332:1–332:7. https://doi.org/10.1145/3491101.3519665

[41] Matt Welsh. 2022. The End of Programming. *Commun. ACM* 66, 1 (2022), 34–35. https://doi.org/10.1145/3570220

[42] Frank F. Xu, Zhengbao Jiang, Pengcheng Yin, Bogdan Vasilescu, and Graham Neubig. 2020. Incorporating External Knowledge through Pre-training for Natural Language to Code Generation. In *ACL*. ACM, 6045–6052. https://doi.org/10.18653/v1/2020.acl-main.538

[43] Frank F. Xu, Bogdan Vasilescu, and Graham Neubig. 2022. In-IDE Code Generation from Natural Language: Promise and Challenges. *Transactions on Software Engineering and Methodology* 31, 2 (2022), 29:1–29:47. https://doi.org/10.1145/3487569

[44] Ming-Ho Yee and Arjun Guha. 2023. Do Machine Learning Models Produce TypeScript Types That Type Check?. In *ECOOP*. Schloss Dagstuhl, 37:1–37:28. https://doi.org/10.4230/LIPIcs.ECOOP.2023.37