# CS1-LLM: Integrating LLMs into CS1 Instruction

Annapurna Vadaparty
University of California, San Diego
USA
avadaparty@ucsd.edu

Daniel Zingaro
University of Toronto Mississauga
Canada
daniel.zingaro@utoronto.ca

David H. Smith IV
University of Illinois
Urbana, USA
dhsmith2@illinois.edu

Mounika Padala
University of California, San Diego
USA
mpadala@ucsd.edu

Christine Alvarado
University of California, San Diego
USA
cjalvarado@ucsd.edu

Jamie Gorson Benario
Google
USA
jamben@google.com

Leo Porter
University of California, San Diego
USA
leporter@ucsd.edu

## ABSTRACT

The recent, widespread availability of Large Language Models (LLMs) like ChatGPT and GitHub Copilot may impact introductory programming courses (CS1) both in terms of what should be taught and how to teach it. Indeed, recent research has shown that LLMs are capable of solving the majority of the assignments and exams we previously used in CS1. In addition, professional software engineers are often using these tools, raising the question of whether we should be training our students in their use as well. This experience report describes a CS1 course at a large research-intensive university that fully embraces the use of LLMs from the beginning of the course. To incorporate the LLMs, the course was intentionally altered to reduce emphasis on syntax and writing code from scratch. Instead, the course now emphasizes skills needed to successfully produce software with an LLM. This includes explaining code, testing code, and decomposing large problems into small functions that are solvable by an LLM. In addition to frequent, formative assessments of these skills, students were given three large, open-ended projects in three separate domains (data science, image processing, and game design) that allowed them to showcase their creativity in topics of their choosing. In an end-of-term survey, students reported that they appreciated learning with the assistance of the LLM and that they interacted with the LLM in a variety of ways when writing code. We provide lessons learned for instructors who may wish to incorporate LLMs into their course.

## CCS CONCEPTS

• **Social and professional topics → Computer science education**.


*ITiCSE '24, July 8–10, 2024, Milan, Italy*

ACM ISBN XX-X-XXXX-XXXX-X/XX/XX
https://doi.org/XXXXXXX.XXXXXXX

## KEYWORDS

CS1, Introductory Programming, LLM, Copilot, Generative AI

**ACM Reference Format:**
Annapurna Vadaparty, Daniel Zingaro, David H. Smith IV, Mounika Padala, Christine Alvarado, Jamie Gorson Benario, and Leo Porter. 2024. CS1-LLM: Integrating LLMs into CS1 Instruction. In *The 29th Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '24), July 8–10, 2024, Milan, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 INTRODUCTION

In 2023, the computing education research community witnessed an explosion of commentary and research on the impact of Large Language Models (LLMs) and Generative AI (GenAI) on computing education courses. In an ITiCSE Working Group report published in December 2023, the authors note "There is little doubt that LLMs and other forms of GenAI will have a profound impact on computing education over the coming years" [14]. Indeed, this profound impact is already being felt. For example, much research has demonstrated that GPT-4 can solve CS1 problems at the level of a top student in the course [14]. Capabilities of LLMs are increasing rapidly, leading instructors to worry about what their programming courses should look like now. Some instructors are attempting to ban the use of the tools, or devising types of assessments where LLMs struggle, while others are embracing the changes [9].

We suggest a redesign of introductory programming courses around LLMs, including reprioritizing learning outcomes, for two reasons. First, with the increasing capability of LLMs, the practice of software development and the required skills for programming are evolving. In a survey from GitHub, it was reported that 92% of developers in the US are using these tools and that 70% of those developers are seeing benefits such as upskilling and increased productivity [16]. We argue that students should be learning with LLMs to help them prepare for a work force that is using LLMs and thus skills we once prioritized, like writing code from scratch, may no longer be as relevant. The second reason is that incorporating LLMs can allow students to more quickly engage in larger, open-ended projects which have more personal relevance to them and

can improve engagement [6]. CS1 students typically work on small, highly constrained problems that do not resemble the software development process [1]. By working on open-ended projects, they gain better exposure to programming practices and are motivated by working on what may be more personally relevant [18].

In this paper, we discuss our experiences teaching a new CS1 (CS1-LLM) with LLMs at the center of learning to program. Our core question in designing the course was: What can students do now with LLMs that they could not do before? We still want students to be able to write code from scratch, and we remain committed to other fundamentals [21] such as carefully reading, tracing, and explaining code. But rather than spending several weeks teaching Python syntax as we have done in the past, we now rely on the LLM's powerful code-writing abilities to help students overcome syntax hurdles and focus on more creative aspects of programming.

## 2 DESIGN PRINCIPLES

As mentioned in the previous section, we designed our new CS1 (CS1-LLM) to help students benefit from the affordances of LLMs. With that overarching goal in mind, we began by establishing the design principles that would guide the development and implementation of our course.

Our first design principle is to enable and encourage students to use LLMs throughout their coursework. To that end, we taught students early how to install and use GitHub Copilot to help them write Python code. We chose Copilot as it integrates well with VS Code, an IDE that many of our students will use in industry or their later programming projects. As we wanted to test students in a setting mirroring the setting in which they learned, we additionally made Copilot available on some supervised quizzes and tests. There is lack of consensus on what the community believes constitute fundamentals that should be learned with and without LLMs. We therefore aimed to ensure that students would emerge from the course with fluency in coding both with and without LLM support. A key skill that is expected after completion of CS1 is to be able to independently read, trace, and write some code, so on some assessments we did not allow the use of Copilot.

Our second design principle is to prepare students to enter a workforce where the use of LLMs will be the norm. It is unfortunately the case that there is a large gap between what is taught in CS courses and the needs of industry [2, 20]. For example, researchers have found that assignments in school tend to ask students to write code from scratch rather than having students learn to read and modify code as is commonly done in industry work, and that school assignments often focus on writing small standalone programs rather than adding features to existing programs [2]. To reduce this gap, we not only teach students to use LLMs, but also use the affordances of LLMs to enable students to develop code reading and modification skills.

Our third design principle is to use what we know from research to improve outcomes for students from underrepresented groups. While not specific to LLMs, we employed several best practices known to improve student outcomes particularly for students from underrepresented groups [15]. These practices include Peer Instruction (PI) [3, 11], media computation [5], and pair programming [10]. We also wanted to ensure students had the ability to receive frequent formative feedback, both through Peer Instruction [11] and practice problems [22].

Our fourth design principle is to provide opportunities for students to be creative. There is a tendency in CS1 courses to use small, highly constrained problems [1] that can be auto-tested using predefined test cases. Unfortunately, those assignments do not provide opportunities for students to work on the types of open-ended projects that inspire creativity and better resemble industry [17]. We have changed our assignments to enable student choice in their programming through open-ended projects.

Our fifth and final design principle is to build a course that will equally serve both students who continue in CS and those who will not take any further computing courses. We want our students to be able to both build on the computing principles they learn and apply those principles to their work even outside of the computing discipline. This necessarily means we need to spend less time on low-level syntax to make room for students to work at higher levels where meaningful work can be done. For example, we introduce and encourage students to use powerful Python modules for automating tedious tasks (such as merging a huge number of pdf files or identifying duplicate images in a huge image library).

## 3 COURSE CONTEXT

The CS1 course took place in a research-intensive university in North America. The course included 10 weeks of instruction and one week for final exams. In the curriculum at the university, this "CS1" course is designed for students with no prior programming experience and is really the first half of two 10-week courses that are equivalent to a 10-week accelerated CS1 course for students with prior programming experience. The course has 3 hours of weekly in-person, instructor-led classes; 1 hour per week of in-person discussion (led by a graduate Teaching Assistant (TA)); and 1 hour per week of in-person closed labs (also led by a TA) where students complete a programming activity, often in pairs. The instructional staff consists of 5 graduate TAs and 33 undergraduate Instructional Assistants. The course redesign team included graduate students and faculty from two institutions as well as a member of the software engineering industry. Use of student data for this work is approved by our Human Subjects Board.

The course has a diverse student population with students from many different majors of study enrolled in the course. The demographics of the course can be found in Table 1. All data except number of Computing Majors came from the end-of-term survey completed by 315 out of 552 students (57% participation rate); Computing Majors came from the course pre-survey completed by 80% of the students enrolled at the start. "Computing Majors" includes Computer Science, Computer Engineering, Math with Emphasis in Computer Science, Bioinformatics, and Data Science. Note that in the United States, Pell-Grant Eligibility can be seen as a proxy for low-income students as this denotes eligibility for federal financial aid.

The course teaches variables, functions, conditionals, loops, strings, lists, and dictionaries in Python. The second half of the 2-part CS1 course, not otherwise discussed here, teaches classes, objects, inheritance, and polymorphism in Java. Both before and after the transition to CS1-LLM, our course has been taught using

**Table 1: Course Demographics**

| Group | Yes | No | Decline |
|---|---|---|---|
| Computing Major | 33.7% | 66.3% | – |
| **Gender** | | | |
| Male | 44.3% | 52.2% | 3.5% |
| Female | 50% | 46.5% | 3.5% |
| Nonbinary | 2.2% | 94.3% | 3.5% |
| **Race/Ethnicity** | | | |
| Hispanic or Latine | 27.0% | 70.2% | 2.9% |
| Native American | 2.5% | 77.8% | 19.7% |
| Black or African American | 3.2% | 77.1% | 19.7% |
| East or Southeast Asian | 43.8% | 36.5% | 19.7% |
| Indian or other South Asian | 9.5% | 70.8% | 19.7% |
| Pacific Islander | 0.6% | 79.7% | 19.7% |
| North African/Middle-Eastern | 3.2% | 77.1% | 19.7% |
| White or Caucasian | 22.5% | 57.8% | 19.7% |
| **Student Status** | | | |
| Transfer Student | 7.3% | 91.1% | 1.6% |
| First-Gen. College Student | 43.2% | 50.8% | 6.0% |
| Pell Grant Eligible | 47.0% | 30.8% | 22.2% |

both Peer Instruction and Live Coding in class, Pair Programming in labs, and typically at least two weeks on Media Computation. The change to incorporate LLMs occurred for the Fall 2023 offering of the course. The course was taught by an experienced instructor who has twelve years of experience and multiple teaching awards.

## 4 COURSE LEARNING GOALS

Using our design principles, we revised the learning goals from the course and divided them by Bloom's taxonomy [8]. Key to our updates is the inclusion of learning goals specific to using LLMs.

**Level 1: Knowledge**
- Define nondeterminism, Large Language Model (LLM), prompt, prompt engineering, code correctness, problem decomposition, and top-down design.

**Level 2: Comprehension**
- Illustrate the workflow that is used when programming with an AI assistant.
- Describe the purpose of common Python programming features, including variables, conditionals, loops, functions, lists, dictionaries, and modules.

**Level 3: Application**
- Apply prompt engineering to influence code generated by an AI assistant.

**Level 4: Analysis**
- Analyze and trace a Python program to determine or explain its behavior.
- Divide a programming problem into subproblems as part of top-down design.
- Debug a Python program to locate bugs.

**Level 5: Synthesis**
- Design open- and closed-box tests to determine whether code is correct.
- Identify and fix bugs in Python code.
- Modify Python code to perform a different task.
- Write complete and correct Python programs using top-down design, prompting, testing, and debugging.

**Level 6: Evaluation**
- Judge whether a program is correct using evidence from testing and debugging.

Some learning goals remain similar to a traditional CS1 course, in that students are still asked to read, trace, and explain code [21]; debug code; modify code; and recognize common Python programming constructs. Our CS1-LLM has less emphasis on writing code from scratch and fixing syntax as compared to our traditional CS1 class. New learning goals for the CS1-LLM course include problem decomposition, top-down design, prompt engineering, and a larger emphasis on testing and debugging.

## 5 OUR COURSE–CS1-LLM

In redesigning the course we used a backwards design where we first consider the course learning goals and work backwards to design assessments and instruction that support students achieving those outcomes [23]. This section describes the main components of the course that were adopted as part of reorienting the course instruction around LLMs. Ultimately, nearly every element of the course including lectures, assessments, and labs were changed for this new version of the course. We have made course materials public in the hopes that others will be able to adopt and benefit from our work [13].

### 5.1 Course Schedule

The topics by week in the course appear in Table 2. For the first four weeks, students focused on learning how to read, trace, and explain code; they also learned how to ask Copilot to explain code and to use the VSCode debugger to gain insight into the state of memory during program execution. Students learned the basics of variables, conditionals, loops, functions, strings, and lists. The remaining 6 weeks taught the software engineering process when working with Copilot (see Figure 1), fleshing out ideas like testing, debugging, and problem decomposition in three separate domains. For two weeks each, we taught students these concepts in the context of data science, image manipulation [5], and games. To ensure students had access to Copilot by the start of the project weeks, students had a mandatory assignment to set up VSCode with Copilot due in the second week of the class. Students were also provided with videos about how to set up their system for both Windows and Mac.

**Table 2: Course Schedule**

| Week | Topic(s) |
|---|---|
| 1 | Functions and Working with Copilot |
| 2 | Variables, Conditionals, Memory Models |
| 3 | Loops, Strings, Testing, VSCode Debugger |
| 4 | Loops, Lists, Files, Problem Decomposition |
| 5 | Intro to Data Science, Dictionaries |
| 6 | Revisit Problem Decomposition and Testing |
| 7 | Intro to Images, PIL, Image Filters |
| 8 | Copying Images, Intro to Games and Randomness |
| 9 | Large Game Example |
| 10 | Python Modules and Automating Tedious Tasks |

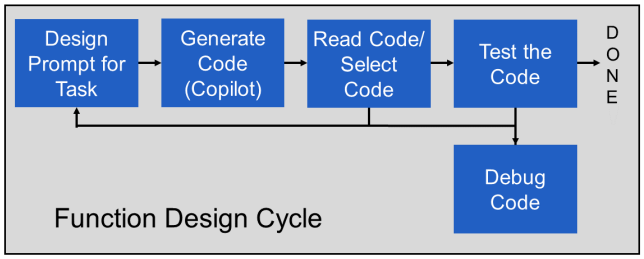


**Figure 1: Workflow recommended to students for working with Copilot. Based on workflow in course textbook [12].**

## 5.2 Textbook

We used the recently published book "Learn AI-Assisted Python Programming with GitHub Copilot and ChatGPT" as the primary course textbook [12]. The book was selected because it teaches readers how to code with the aid of an LLM, had a games project chapter that overlapped well with our final project, and still teaches students how to understand Python code.

## 5.3 Lectures

Lectures consisted of a combination of mini-lectures, Live Coding, and Peer Instruction. In addition to standard materials one might regularly see in a CS1 class, most lectures contained discussion of Copilot interactions. Students were given responses from Copilot that were incorrect for the task and asking students to either identify the error with the response or identify test cases that would uncover the mistake. Lecture also showed conversations with Copilot Chat. Copilot Chat is similar to ChatGPT but it is integrated into VS Code and has access to the current code being written. These Copilot Chat conversations were used to help students dive deeper into understanding their code, or to offer examples of how students can use Chat to explore Python libraries that may help them complete a given task. When drawing memory diagrams for students, the instructor alternated between drawing on their tablet in class and using the VSCode Debugger to show the state of execution in the debugger.

## 5.4 Assessments

We designed a variety of new assessments that better align with the new learning goals and structure for the course.

**Formative Assessments.** Our third design principle sought to offer students frequent formative feedback. In addition to Peer Instruction questions in class and course labs, we adopted PrairieLearn [22] to offer students many opportunities to practice code tracing, code explaining, code testing, and code writing both on homework and as practice quizzes. Formative Assessments were worth 35% and prepared students for the larger summative assessments. Peer Instruction participation was worth 5%, reading quizzes 5%, homework 15%, and labs 10%. Summative assessments were worth 65%, with projects worth 10%, quizzes 30%, and final exam 25%.

**Homework.** Students were given a homework on PrairieLearn [22] each week. The homework consisted of a variety of problem types (multiple choice, short answer, Explain in Plain English [19], Parson's Problems [4], debugging, code writing) and students were allowed multiple attempts to solve each problem correctly. The homework was designed to be completed without using an LLM but students were told they could use an LLM if stuck.

**Quizzes.** The four 50-minute quizzes for the course increased in complexity as the term progressed. All four quizzes included code tracing, code explaining [21], Parsons Problems [4], and small code writing questions, mostly without access to Copilot. In addition, on later quizzes, students were asked to answer questions on testing, debugging, and problem decomposition. As the data science, image manipulation, and game domains were introduced in class, they were also introduced on quizzes.

**Projects.** Students completed one project per domain of data science, image manipulation, and games. Each project was intentionally open-ended, allowing students to showcase their creativity. The first project on data science asked students to find a dataset on Kaggle [7], ask a question that can be answered by that data, and then write a program to answer that question. The second project on images asked students to write a program to create an image collage by filtering their images and pasting images on top (or adjacent) to one another. The third project on games asked students to design a text-based game and implement either a playable game with user interaction or a game simulation to determine the likelihood of winning a particular game. Students were allowed to use Copilot for the projects to increase the scale of what they could accomplish in a CS1 course. To help students properly scope their projects, they were required to meet with an instructional assistant a week before the due date to obtain guidance on proper scoping and get hints on how they might go about solving the problem.

We had students submit their code and any supporting files, a diagram of the functions they created when they decomposed the problem into multiple subproblems, and a 5-minute video of them explaining their project with at least 3 minutes explaining how one of their functions worked. Each project was graded by an instructional assistant and grading took 10–15 minutes per project.

**Labs.** Each week students were given a lab to complete. These labs were completed either during the synchronous mandatory 50-minute lab session or, on quiz days when the lab was used for a quiz, at home. Labs were designed to help students get started with the programming concepts or programming domain (e.g., images) that were discussed in lecture that week. Labs included working with Copilot to solve a problem, writing code without Copilot, and debugging buggy code using the VSCode Debugger.

**Final Exam.** The final exam consisted of three parts: 1) (90 minutes, worth 70% of the exam grade) a primarily multi-choice component consisting of tracing code, explaining code, testing code, and debugging, along with short answer and Parsons Problems, 2) (45 minutes, worth 15% of the grade) four code writing tasks of increasing difficulty to be completed without Copilot, 3) (45 minutes, worth 15% of the grade) one large new problem to be completed with Copilot. For the first two parts of the exam, students worked on a lab machine with access to a web browser for PrairieLearn. For the third part of the exam, students had access to a workspace in PrairieLearn with Copilot enabled but could also use their personal computers if they preferred. Proctors ensured students were only using allowed content.

For the third part of the exam, students were asked to analyze a new dataset or implement a restricted version of a spell check. For example, for spell check, students needed to take a given word and

return all correctly spelled words that can be reached by adding, removing, or changing one character. Students were given example test cases and were provided partial credit for partial progress.

# 6 STUDENT PERCEPTIONS

Given the scope of changes we made to CS1, we wanted to understand the impacts on the student experience. To that end, we asked students for their opinions about the course in an end-of-course survey. We report here on questions from that survey that directly relate to the student perceptions of working with an LLM.

For the quantitative questions, we created graphs to visualize the responses. For the open-ended questions, we identified quotes in the data that provided context to these initial visualizations and findings. We did not conduct a formal quantitative or qualitative analysis of the data at this point.

## 6.1 Student Comfort with Copilot

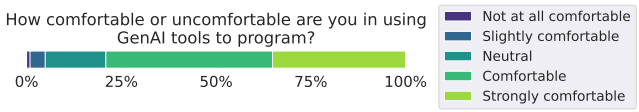


**Figure 2: Student Comfort Programming with GenAI**

We asked students: "How comfortable or uncomfortable are you in using GenAI tools to program?" and their responses appear in Figure 2. Encouragingly, the vast majority (79%) reported being comfortable using GenAI tools to program.

## 6.2 Impact of Copilot on Student Learning

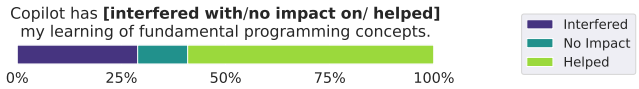


**Figure 3: Student Perceptions of how Copilot impacted their learning.**

When asked about their overall opinion of working with Copilot, a slight majority of students report that it helped their learning of programming concepts (Figure 3). Student responses to an open-ended survey question ("How did you feel about working with Copilot as you learned to program this quarter?") offered insights into why students reported that Copilot either helped or interfered with their learning. Among students who reported that it was helpful for their learning, some students reported an appreciation for having immediate personalized help. For example, one student reported that "It was really nice having an assistant that could always help me the moment I needed it and made programming a lot less daunting." Among students who reported that it interfered with their learning, many reported that they found Copilot useful but felt they had become over-reliant on it. For example, "Copilot allowed me to develop a sufficient understanding of a lot of concepts, but I wasn't necessarily able to master most of those concepts. I feel like this is because Copilot enhances the speed I'm able to learn at, but doesn't encourage me to master the concept to the level where I'm able to write the code entirely on my own..." Another student reported "If I were asked to code without Copilot, I wouldn't feel very confident in myself despite doing well in the course."

We gain a deeper insight into this range of opinions through the questions asked in Figure 4. Students overall felt confident that they could recognize and understand the code generated by Copilot, that they are learning how to write programs on their own when using GenAI tools, and that they have gained a fundamental understanding of programming concepts. Fewer students reported confidence in their ability to perform the tasks from the course without Copilot. This may be related to their lack of confidence in their ability to "identify the types of coding problems they should be able to complete without Copilot"; 31.1% of the students were slightly to strongly unconfident that they could do so (Figure 4). Some students felt frustrated by their inability to use Copilot for some portions of the course and not others. A representative quote from a student is that "Although it was helpful, it was really difficult for me on quizzes, the whole course is based on using AI tools such as copilot for help, yet taking it away on quizzes when we had it in every other type of work seemed a little unfair." As we discuss in Section 7, we could have offered more guidance about what we expect students should be able to code with and without Copilot.

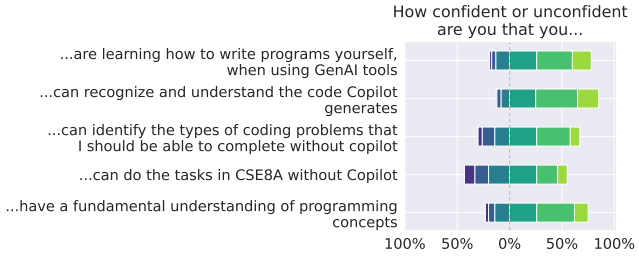


**Figure 4: Student confidence in their ability and understanding at the end of the course.**

## 6.3 How Students Interacted with Copilot

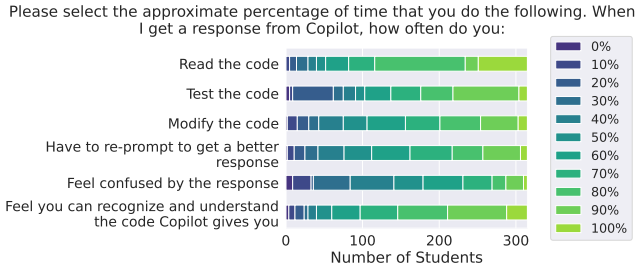


**Figure 5: Student perceptions on how they interact with Copilot.**

Figure 5 summarizes how students worked with Copilot during the term. We're encouraged that the majority of students said that they read the code returned by Copilot at least 80% of the time, and that they tested and modified the code at least 60% of the time. It is not surprising that students occasionally or even frequently did not have to modify the code returned by Copilot; indeed, we coached them on how to give prompts that would lead to code that was as close to what they were looking for as possible. Finally, while around a third of students reported only rarely (30% of the time or less) being confused by Copilot's responses, 44% of students

reported being confused 50% of the time or more, with 10% of students reporting being confused the vast majority of the time (80% of the time or more). Perhaps a reason for this trend is that Copilot can sometimes give sophisticated solutions that are beyond the scope of an introductory course, but we have not yet analyzed the complexity of Copilot responses.

## 6.4 Student Perceptions of the Projects

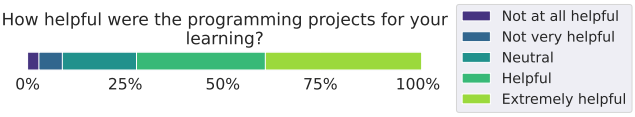


**Figure 6: Student Learning Programming with Projects**

We also asked students about the projects, as they played a large role in the new class. We assigned projects that were open-ended and more challenging than projects in a typical CS1 so students could showcase their ingenuity. Students largely reported that the programming projects were valuable for their learning, as shown in Figure 6.

Students overall seemed to enjoy projects, reflected in one representative comment: "Though they were a little frustrating at times, I was really impressed by what I was able to do for each project and they are probably what I will remember most from the class. I [...] enjoyed being able to be creative and apply what we learned in class. I also thought they were challenging enough, while still being doable. I definitely felt more comfortable with [...] coding in general after each project."

# 7 DISCUSSION

As shown in the last section, a vast majority (79%) of students felt they could program with the LLM and a slight majority (59%) felt it was helpful in learning programming. In contrast, some students reported concerns about an over-reliance on the tool or felt they may not have learned the fundamentals as well as they'd hoped. While these concerns should be addressed, students did create projects far beyond what we would have previously expected in a CS1 course. Thus, it is not surprising that some students felt like they are not capable of writing the code without an LLM, as a CS1 student would not be expected to write such advanced code.

The changes we made to the course led to both design and administrative challenges. That said, we are encouraged by the student enthusiasm for the new course and open-ended projects. We plan to iterate on the course and are optimistic about future offerings.

For instructors looking to incorporate LLMs into their teaching of CS1, we offer these observations and lessons learned based on the experiences of the instructional staff and student feedback.

**Student Performance.** Student performance on exams mostly mirrored performance in previous CS1 classes. Anecdotally, the CS1-LLM students' performance on code writing (from scratch) questions was slightly lower than past offerings, but their performance on code tracing and code reading questions was roughly the same. As already discussed, the scope of their projects was well beyond what we would regularly see in a CS1 course. We were also pleased to see many students were able to solve the large programming task in the 3rd part of the final exam. Future studies will explore student performance in more detail.

**Essential Components.** If one is hesitant to fully adopt a new course, the changes we feel are most essential include the projects (on which students used LLMs), teaching of problem decomposition and testing, and using LLM code responses in class examples.

**Extent of Changes.** Although nearly all elements of the course were changed, ultimately the depth of the changes felt less than we initially expected. Students still learned how to read, trace, explain, and write basic code. The larger changes occurred when interacting with the LLM in lecture, teaching problem decomposition and testing, using larger projects, and assessing students with Copilot.

**When to Introduce LLMs.** We introduced LLMs in the first week. This meant that students were grappling with using an IDE, using an LLM, and working with Python all at the same time. In future offerings, we recommend delaying introducing the LLM briefly so students can write small programs in the IDE on their own first.

**Explicit Expectations.** A common concern voiced in surveys was that students were confused about what they should be able to do with and without the aid of an LLM. In hindsight, the advice that students could use LLMs if stuck on homework may not have been helpful as some students reported using Copilot heavily for the homework and becoming over-reliant. In future offerings, we will tag every homework question clearly as something they should be able to do with/without the aid of an LLM.

**Learning Goals and Assessments.** A challenge in re-imagining a course is in ensuring the assessments are aligned to the learning goals of the course. Given the many decades of teaching CS1 without the aid of an LLM, our team often defaulted back to the kinds of questions we asked in prior versions of the course. If past questions are used, we recommend checking that the questions align with the (updated) course learning goals.

**Beware of Non-Determinism in Class.** A challenge in live-coding with an LLM is that it learns from your behavior. In teaching two sections of the same class, the LLM made discussion-worthy mistakes in the first section but then simply parroted back our fixed code from that section for the second section. We learned to have Copilot responses pasted into our slides rather than Live Coding.

**Signing into GitHub is Difficult in Exams.** For part of Quiz 2, we attempted to have students solve a small task in 15 minutes with the aid of Copilot in a PrairieLearn workspace with Copilot enabled. Students were unable to finish this task as much time was lost due to forgotten credentials or need of their phones for 2-factor verification. In general, if students are allowed Copilot during an exam, we recommend allotting sufficient time to log in or allowing them to use a personal device with stored credentials.

# 8 CONCLUSION

We offered a new kind of introductory programming course (CS1-LLM) that integrated LLMs into the course instruction. We described the revised learning goals, the course structure, and lessons learned for potential course adopters. From student surveys and student projects, we learned how students interact with the LLM, that students in the course valued the open-ended course projects, and that the majority felt that the LLM helped their learning. We see this course offering as a first step toward using LLMs to improve student outcomes and experience in CS1.

# REFERENCES


[1] Joe Michael Allen, Frank Vahid, Alex Edgcomb, Kelly Downey, and Kris Miller. 2019. An analysis of using many small programs in CS1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 585–591.

[2] Michelle Craig, Phill Conrad, Dylan Lynch, Natasha Lee, and Laura Anthony. 2018. Listening to early career software developers. *Journal of Computing Sciences in Colleges* 33, 4 (2018), 138–149.

[3] Catherine H Crouch and Eric Mazur. 2001. Peer instruction: Ten years of experience and results. *American Journal of Physics* 69, 9 (2001), 970–977.

[4] Paul Denny, Andrew Luxton-Reilly, and Beth Simon. 2008. Evaluating a new exam question: Parsons problems. In *Proceedings of the 4th International Workshop on Computing Education Research*. 113–124.

[5] Mark Guzdial. 2003. A media computation course for non-majors. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*. 104–108.

[6] Carol Hulls, Chris Rennick, Sanjeev Bedi, Mary Robinson, and William Melek. 2015. The use of an open-ended project to improve the student experience in first year programming. *Proceedings of the Canadian Engineering Education Association (CEEA)* (2015).

[7] Kaggle. [n. d.]. Kaggle: Your Machine Learning and Data Science Community. https://www.kaggle.com/. Accessed: 2024-01-01.

[8] David R Krathwohl. 2002. A revision of Bloom's taxonomy: An overview. *Theory into Practice* 41, 4 (2002), 212–218.

[9] 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 *Proceedings of the 19th ACM Conference on International Computing Education Research*. 106–121.

[10] Charlie McDowell, Linda Werner, Heather E Bullock, and Julian Fernald. 2006. Pair programming improves student retention, confidence, and program quality. *Commun. ACM* 49, 8 (2006), 90–95.

[11] Leo Porter, Dennis Bouvier, Quintin Cutts, Scott Grissom, Cynthia Lee, Robert McCartney, Daniel Zingaro, and Beth Simon. 2016. A multi-institutional study of peer instruction in introductory computing. In *Proceedings of the 47th ACM Technical Symposium on Computer Science Education*. 358–363.

[12] Leo Porter and Daniel Zingaro. 2024. *Learn AI-Assisted Python Programming: With GitHub Copilot and ChatGPT*. Manning Publishing.

[13] Leo Porter, Daniel Zingaro, Beth Simon, and Christine Alvarado. [n. d.]. CS1-LLM Course Materials. https://github.com/copilotbook/CS1-LLM. Accessed: 2024-03-15.

[14] 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 *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education*. 108–159.

[15] Adrian Salguero, Julian McAuley, Beth Simon, and Leo Porter. 2020. A longitudinal evaluation of a best practices CS1. In *Proceedings of the 16th ACM Conference on International Computing Education Research*. 182–193.

[16] Inbal Shani. 2023. Survey reveals AI's impact on the developer experience — The GitHub Blog. https://github.blog/2023-06-13-survey-reveals-ais-impact-on-the-developer-experience

[17] Sadia Sharmin. 2021. Creativity in CS1: A Literature Review. *ACM Trans. Comput. Educ.* 22, 2 (2021).

[18] Sadia Sharmin, Daniel Zingaro, and Clare Brett. 2020. Weekly Open-Ended Exercises and Student Motivation in CS1. In *Proceedings of the 20th Koli Calling International Conference on Computing Education Research*.

[19] David H Smith IV and Craig Zilles. 2023. Code Generation Based Grading: Evaluating an Auto-grading Mechanism for "Explain-in-Plain-English" Questions. *arXiv preprint arXiv:2311.14903* (2023).

[20] Sander Valstar, Caroline Sih, Sophia Krause-Levy, Leo Porter, and William G. Griswold. 2020. A Quantitative Study of Faculty Views on the Goals of an Undergraduate CS Program and Preparing Students for Industry. In *Proceedings of the 17th ACM Conference on International Computing Education Research*. 113–123.

[21] Anne Venables, Grace Tan, and Raymond Lister. 2009. A closer look at tracing, explaining and code writing skills in the novice programmer. In *Proceedings of the 5th International Workshop on Computing Education Research*. 117–128.

[22] Matthew West, Geoffrey L Herman, and Craig Zilles. 2015. PrairieLearn: Mastery-based online problem solving with adaptive scoring and recommendations driven by machine learning. In *2015 ASEE Annual Conference & Exposition*. 26–1238.

[23] Grant Wiggins and Jay McTighe. 2005. *Understanding by design*. Association for Supervision and Curriculum Development.