61A Bot Report: AI Assistants in CS1 Save Students Homework Time and Reduce Demands on Staff. (Now What?)

J.D. Zamfirescu-Pereira Laryn Qi
Björn Hartmann John DeNero Narges Norouzi

{zamfi,larynqi,bjoern,denero,norouzi}@berkeley.edu

UC Berkeley EECS
Berkeley, CA, USA

Abstract

Chatbot interfaces for LLMs enable students to get immediate, interactive help on homework assignments, but even a thoughtfully-designed bot may not serve all pedagogical goals. In this paper, we report on the development and deployment of a GPT-4-based interactive homework assistant ("61A Bot") for students in a large CS1 course; over 2000 students made over 100,000 requests of our bot across two semesters. Our assistant offers one-shot, contextual feedback, primarily through a low-friction "get feedback" prompt within the command-line "autograder" our students already run to test their code. Our Bot wraps student code in a custom prompt that supports our pedagogical goals and avoids providing solutions directly. We discuss our deployment and then analyze the impacts of our Bot on students, primarily through student-reported feedback and tracking of student homework progress. We find reductions in homework-related question rates in our course forum, as well as substantial reductions in homework completion time when our Bot is available. For students in the $50^{th} - 80^{th}$ percentile, these reductions typically exceed 30 minutes per assignment, over 4 standard deviations faster than the mean in prior semesters. Finally, we conclude with a discussion of these observations, the potential impacts on student learning, as well as other potential costs and benefits of AI assistance in CS1.

CCS Concepts

• **Social and professional topics** → **CS1**; • **Applied computing** → **Computer-assisted instruction**.

Keywords

Automated Tutors, Large Language Models, AI Assistant Deployment, AI Assistant Evaluation

1 Introduction

The recent wide availability of Large Language Models (LLMs) has given students in introductory Computer Science (CS) courses a tempting alternative to asking a human TA for help on programming assignments—and potentially waiting hours to receive it. However, while naively used LLMs do help students solve assigned problems, they typically do so by providing correct answers along with explanations, allowing students to avoid the process of developing solutions themselves and the learning associated with this process. A number of recent reports [8, 10, 12, 13, 15, 18, 36] present more thoughtful approaches: new systems, also based on LLMs, geared towards offering guidance and assistance without providing direct solutions.

Both students and instructors are reported to find these systems helpful [15, 18]. Recent studies of deployed LLM-based assistants have included analyses of what affordances are most appreciated and what kinds of functionality students are most likely to use [9, 15]; how to effectively design systems with guardrails to reduce instances of solution-sharing [18]; and what kinds of error messages are most correlated with reductions in error rates [36]. There is a growing interest in understanding the landscape of these new systems and their impacts on students learning computer science [29]—but while much is known about how these systems are designed and what seems to work well, comparatively less has been reported on how these systems impact specific courses.

In this experience report, we first present our own assistant, a *low-friction* "61A Bot" with 24/7 availability that offers feedback *on every run* of an "autograder" that students use liberally to test their code-in-progress. Our Bot constructs a GPT-4 request using our custom prompt, homework question text, student code, and autograder error output (where available), returning its response to students. The prompt is itself designed to steer towards feedback that mirrors how we ourselves typically approach student questions, aligned with recent work in this area [23]: identifying whether the student understands the question, which concepts students might need reinforcement on, and whether they have a plan, and then helping students by providing conceptual, debugging, or planning support as appropriate.

Then, we examine our Bot's impact on our CS1 course—a large (~ 1000-student) Python-based course targeted at CS majors. Ultimately, we ask: **What impact does this deployment have on our course, on students and on staff?** First, we find that students assess a majority of Bot-provided hints as being helpful, in line with prior work. Second, through a retrospective observational study aimed at quantifying impacts, we find that requests for homework help to our course forum drop dramatically (75%) after our Bot is deployed, and that students spend substantially less time completing their homework—25-50% less time, often more than 30 minutes faster, a 4-8 standard deviation reduction from the mean completion time for the same assignments in prior semesters.

We note that while these impacts do not necessarily imply improvements in student learning outcomes, speeding up homework completion times without increasing learning could still be a positive outcome: the time saved on completing traditional assignments

could be harnessed by instructors to increase learning in other ways. In our final section, we discuss some of the costs and potential benefits of deploying LLM-based assistance in CS1, exploring possible learning impacts, reductions in the visibility of student challenges to staff, and opportunities for future research.

Thus, our main contributions are:

(1) A rich description of our low-friction CS1 assistant and its deployment in our course;

(2) Findings of students' subjective experiences based on student surveys over two semesters; and

(3) A retrospective, observational analysis of student homework completion patterns, identifying quantitative impacts our bot had on staff demand and students' homework experiences.

2 Background & Related Work

Generative models such as ChatGPT,[1] OpenAI Codex [4], Amazon CodeWhisperer[2], and GitHub Copilot[3] offer promising opportunities for enriching the learning experience of students. These models have already been leveraged by educators in different areas of Computing education [8, 10, 11, 13], where they accelerate content generation and seem to be impacting the relevant skills students gain in introductory CS courses. Researchers have studied LLMs in areas such as generating code explanations [3, 16, 19, 22], providing personalized immediate feedback [2], enhancing programming error messages [17, 36], generating discussion forum responses [20, 25], and automatic creation of personalized programming exercises and tutorials [31, 32, 38] to enhance the comprehensiveness of course materials.

However, the integration of LLMs in CS1 instruction comes with challenges. Students could become overly reliant on automation (a concern at least as old as calculators [7]), potentially hindering their development of critical problem-solving skills—though recent work suggests these negative effects can be avoided, at least for programming assistance [14]. Taken to an extreme, the resulting absence of human interaction could have negative effects, alongside other ethical concerns related to plagiarism and the responsible use of LLM-generated code. To maximize the benefits of LLMs while mitigating these challenges, a thoughtful and balanced approach to their incorporation into CS1 courses is essential [9, 21, 24].

Through deployments of LLMs as intelligent tutors, students can receive immediate, personalized support and guidance, which would ideally foster a deeper understanding of coding concepts and promote self-paced learning—just as with pre-LLM Intelligent Tutoring Systems (see [6] for a review, and [35] for an example). The ability of LLMs to generate *tailored* resources, such as new, personalized tutorials and newly-generated code examples, not only expands the available learning materials but also accommodates students' varying learning preferences—though these generated materials are not always better [27]. Educators should integrate LLMs as *complementary* tools, striking a balance between automation and human interaction while emphasizing the development of critical problem-solving skills and responsible coding practices, ultimately serving students better in their CS education.

Researchers are also increasingly integrating LLM-based *chatbots* in courses [19, 37] and online educational websites [26] to provide immediate personalized feedback, and in tools in supporting students' development of programming skills [5, 12, 30]. These include CodeHelp [18, 33] and CodeAid [15], two systems (and deployments) that bear a number of similarities to our own—though those systems enable students to ask questions, while ours (we believe uniquely) integrates feedback directly into the tool students already use to execute that code, and then builds on the student's history of prior assistant hints and code changes in response.

3 Design & Deployment

Our deployment focused on providing students help with homework problems in part to address frequent student feedback from prior terms about long wait times for TA support for homework problems. In particular, we chose to focus primarily on the kinds of debugging assistance our staff are often asked for.

Three concerns—hallucinations, students sharing personal information with a third party, and the harms from an unmonitored chat interaction—led us to deploy a one-shot "Get Help" interaction mode without an opportunity for "chat" follow-up. This meant that one valuable pedagogical tool—having students explain their understanding of the problem—would remain out of reach in this initial deployment.

Following a common tutoring pattern [23], we designed a prompt that would try to assess student conceptual knowledge, based on the provided code, and offer syntactical, logical, or even template-code suggestions—but not solutions. This prompt explicitly includes a sequence of questions to consider in response to the student code:

(1) Is the student missing conceptual knowledge?
(2) Is their current code on the right track?
(3) How close are they to a solution?
(4) Were they able to follow previous advice?
(5) Do they have a reasonable plan?

Though we avoided students explicitly writing natural language "chat" messages to the bot, we did want some degree of continuity—which we achieved by also including up to three prior *(student code, bot advice)* exchanges, if available (enabling question 4).

In addition to the steps above, the prompt also includes a per-problem instruction block, which we used for about 10% of problems, and more general instructions such as `Do not give the student the answer or any code.` and `Limit your response to a sentence or two at most.`

3.1 Course Details & Deployment

Our course covers most of the typical CS1 content, plus a few additional topics, and we report here on the CS1 portion of our course. These modules are taught using Python, to a student population made up predominantly of CS majors and intended majors (50 ± 10% majors/intended), with substantial prior preparation (80 ± 5% any prior CS course, including high school). These demographics are broadly consistent across the semesters we examine historically in this report, and variations do not correlate with the results described here. (Fall and Spring semester students do differ

[1] https://chat.openai.com/
[2] https://aws.amazon.com/codewhisperer/
[3] https://copilot.github.com/

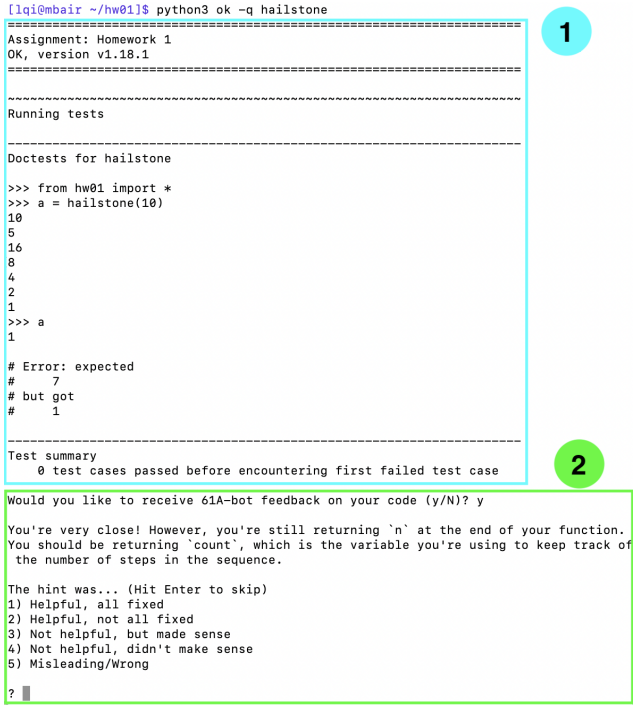


Figure 1: Screenshot of the "autograder" command-line interface. Students run the autograder to test their code against a test suite (1); if any test cases fail, they are asked if they would like to receive Bot feedback, and then asked to assess the hint (2).

demographically due to how course requirements are structured at our institution, and so we treat those as separate populations for this work.) In our course, content is typically introduced in lecture, and then reinforced in three assignment types: first, "labs," which include a mixture of notional machine reinforcement (e.g., tracing, "What would Python do?") and traditional homework problems, completed in small groups without access to 61A Bot. These are followed by "homework," which students complete individually *with* access to 61A Bot. Finally, students complete "projects" that bring together multiple concepts towards a single goal, over a few weeks, again without access to 61A Bot. In addition, students were prohibited by course policy from using ChatGPT or other similar AI-based systems for help across all assignment types—but we do not have the means to enforce this.

3.1.1 Using 61A Bot. Students primarily (93%) access our Bot through an autograder[4] in the command line which provides the result of running the student's code on a set of test cases; if any fail, the student is then asked whether they would like to receive feedback on their code from 61A Bot (see Figure 1). If they do, the autograder collects the student's code, any errors from executing test cases, and constructs a request from these.

Students using VS Code[5] (our course does not prescribe a specific editor) can install an extension to enable a "Get Help" button in the toolbar. When activated, the extension collects students' code, makes a best guess of which homework problem the student is working on (as several problems often appear in a single file), and constructs a request.

In both cases, the collected code includes the student's full assignment code file for that assignment, which includes code for multiple questions. Requests thus include: our prompt, along with any problem-specific notes; the text of the specific homework problem the student is requesting help for (typically one of 4-6 such problems within the assignment); up to 3 pairs of prior-code/Bot-response text (if the student has made prior requests); the student's current code (for all problems within the assignment); and, finally, any error text from failed test cases. These requests are sent to a server run by our instructional staff, which passes it to GPT-4 and logs the request and GPT-4's response for further analysis.

Students are informed when installing the software that, in using our assistant, all code they write will be sent to OpenAI via Microsoft Azure[6], and that they should not include any content in their code files (e.g., comments) that they do not want to share.

3.1.2 Deployment Timeline. We piloted and continued development on 61A Bot throughout the academic year 2023-2024. In Fall 2023, we deployed an initial pilot of 61A Bot to a section of 400 students. This was followed by a full-scale deployment in week 10 (after the CS1 portion of our course) for the approximately 1400 students across both sections of the course. In conjunction with the wider deployment, we also enabled access through the autograder tool students could already run from the command line to validate their code against a set of test cases. In Spring 2024, all 900 enrolled students had access to both modalities of the bot from the start of the semester. Our qualitative analysis thus draws on both semesters, while our quantitative analysis is focused on the full deployment in Spring 2024 where we can more confidently compare like-for-like with a full-term deployment and historically comparable student populations.

4 Outcomes

Students' adoption of 61A Bot was immediate, and usage exploded once we integrated access into our autograder. In this section, we detail usage patterns and changes in reliance on course staff, report on student assessments of their experiences, and finally offer an investigation of the possible impacts of 61A Bot on homework completion times. Unless otherwise noted, statistical accounts in this section come exclusively from Spring 2024, the semester in which our fully-developed Bot was deployed for all students (see §3.1.2 for deployment details).

4.1 Usage & Reliance on Staff

Usage patterns show that students are returning to 61A Bot multiple times as they engage in homework: across our pilot and full deployment semesters, over 2000 students made a total of 105,689 requests of our bot. The median student in our full deployment

[4]OK Client, https://github.com/okpy/ok-client

[5]Visual Studio Code, https://code.visualstudio.com

[6]https://ai.azure.com/

	Fa22	Sp23	Fa23	Sp24
# Students	1656	1169	1407	872
# Questions	1853	2035	1823	1033
per 1000 students	1119	1741	1296	1185
# HW Questions	234	402	177	77
per 1000 students	141	344	126	88

Table 1: Average number of forum questions, Fall 2022–Spring 2024, for the CS1 portion of the class (Fall 2023 results thus do not include the period of widespread deployment).

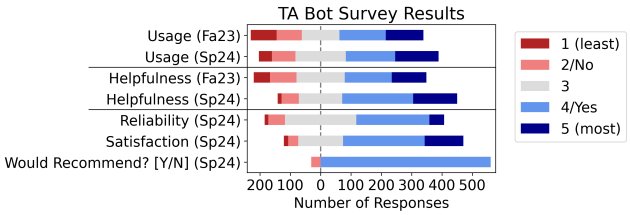


Figure 2: 61A Bot Survey Results. Students were asked to report on their usage and perceptions of Bot helpfulness and reliability, as well as their overall satisfaction on a scale of 1-5, from least to most.

semester made 25 requests to our bot, rising to 80 requests for the student at the 95th percentile. As expected, usage increases as the assignment deadline nears and is concentrated in the late afternoons and evenings—a pattern similar to the usage reported in [15, 18]—with a peak request rate of 291 requests/hour.

This engagement correlates with a reduction in help requests on our online discussion forum for students to receive asynchronous help. There is a substantial (30%) decrease in the number of questions asked (scaled to total enrollment) throughout the semester from Spring 2023 to Spring 2024, from 1741 to 1185 questions per thousand students. The impact on homework-specific questions is even larger, showing a 75% decrease from 344 homework questions per thousand students in Spring '23 to 88 in '24—see Table 1. (We include scaled question counts for Fall 2022 and Fall 2023 as data points illustrating the level of consistency across semesters.)

4.2 Student Reception

Student feedback suggests that students also found the Bot helpful. We solicited this feedback in two ways: First, we queried students for their assessment of each individual hint, which we received for approximately 27% (7459/27419) of queries.[7] Of these, 70% (5210/7459) were rated as "helpful," with 45% of those, 2368/5210, reporting that the problem was now resolved. A further 10% (743/7459) were rated as "not helpful, but made sense," while the final 20% (1496/7459) were rated as insensible, misleading, or wrong.

Second, we formally surveyed students on their usage and perceptions of the Bot. In Fall 2023, we conducted a non-anonymous survey at the end of the semester to which 49% (698/1407) of students responded. Students were asked to rate how much they used the bot and how helpful they found it on a scale from 1 to 5. As expected, those who reported more usage also found it more helpful.

In Spring 2024, we conducted a non-anonymous survey at the end of the semester, to which 89% (774/872) of students responded. On a scale from 1 to 5, we asked students to rate their bot usage, bot helpfulness, bot reliability, and overall satisfaction with the bot. Finally, we asked them whether or not they recommend that the bot be available to students in future semesters. The results from these surveys can be found in Figure 2.

Note that these results include both our partial- (Fall 2023) and full-deployment (Spring 2024) semesters; the Fall 2023 results thus reflect usage only post-deployment. Additionally, these surveys were non-anonymous to track individual participation (for pedagogical goals independent of this project); because of this non-anonymity, however, we did not ask whether students relied on ChatGPT or other prohibited (by course policy) methods of support, reasoning that we were unlikely to be able to rely on such results.

In the Spring 2024 survey, we also asked two optional free-response questions: What did students like the most and least about learning with the bot? We include a representative response for each here:

Most liked:
"What I loved about the bot is that it allowed me to get feedback when I didn't have access to a tutor. Accordingly, instead of banging my head against the wall for hours, I was able to get feedback about what I was doing wrong and correct the mistakes. For me personally, I would have had a lot more success in this class if I would have had access to the bot for labs. (Labs on average took me about 2 and a half hours to complete, and sometimes longer if I didn't have access to a tutor)."

Least liked:
"Sometimes the answers were slightly vague. Of course, the bot can't simply spit out the answer, but sometimes it was frustrating how it would say 'you're on the right track, but there seems to be a conceptual misunderstanding with __' — the explanations for the blank could be a bit jargon-filled and didn't always directly help me resolve the misunderstanding due to imprecise language."

Students generally appreciated 61A Bot's accessibility, debugging capabilities, and time savings. However, the hints were sometimes too vague for the students to make changes, while other times, the bot was too specific and gave away too much. Despite our attempts to steer GPT-4 towards rephrasing and a broader diversity of responses to repeat inputs, these incidents still occur.

4.3 Effects on Homework

To understand the effects of 61A Bot on students in our course, we compare student performance from our semester of full deployment, Spring 2024 (SP24), with performance from prior Spring semesters (see §3.1), going back through Spring 2021 (SP21)—comprising a total of 1,643,613 data points from 6,034 students. We chose this method of analysis in large part because our IRB does not allow differential access to tools in courses, and thus we could not run a randomized control trial.

[7] These counts reflect queries from only those students who gave consent for their data to be used for research, and only during the CS1 portion of our course.

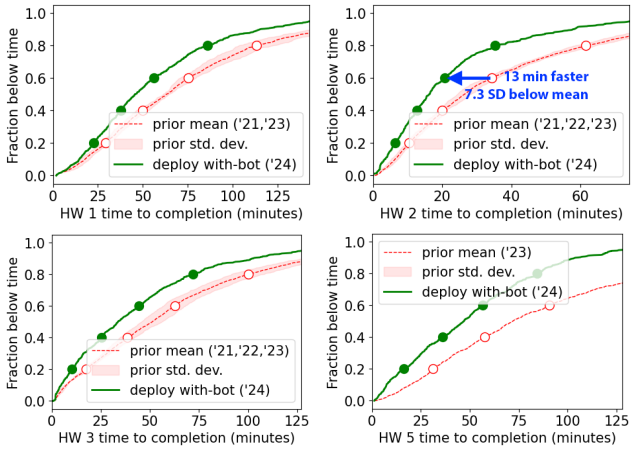
Though 61A Bot is available to students for **homework** assignments, it is not available for **lab** assignments nor for **projects** (see §3.1). To the extent that performance differences on homework assignments between pre- and post-deployment semesters are inconsistent with performance differences on lab assignments and projects, some of this difference may be attributable to the use of the Bot. Though course content is substantially similar across the semesters we report on here, there is some variation in course staff, individual lectures, and specific problems within labs and homework assignments—where homework and lab problems have changed more than trivially, we have omitted them from our comparisons, examining only those that are identical in nature and sequencing within our course. (We confirm, too, that the specific assignments we report on here are representative of the full set of assignments, and we are aware of no other major changes in course content, delivery, or staffing occurred during this time.)

4.3.1 Data & Analytical Method. Our course "autograder" records every attempt a student makes to test their code, storing a "snapshot" of student code on our instructional servers, along with a student identifier and a timestamp; we use these to reconstruct a history of student progress. Autograder use is unlimited, and students typically revise their programs repeatedly until they are ready to submit their final code. Nearly all students submit code that successfully passes all test cases, so examining the final submitted code artifact alone does not necessarily provide useful insight into impacts. Thus the metric we consider here is *time to completion*, rather than passing test cases or other measures of code quality.

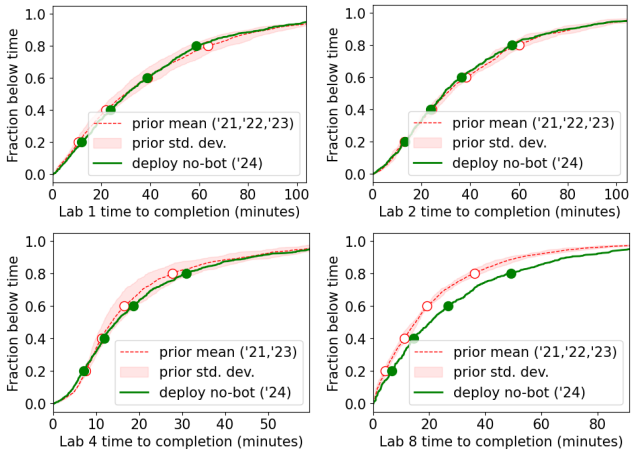
We calculate an approximate total completion time by summing the timestamp deltas between snapshots. To account for students completing the homework across multiple sessions, we ignore time deltas above a 60-minute threshold—a value chosen somewhat arbitrarily, but we confirm that the results we report here are robust to values in the range of 20 to 120 minutes.[8] For clarity, we aggregate individual repeated problems into "assignments," e.g., "HW 1." Occasionally, one assignment in a particular semester differs sufficiently from other semesters that we omit it entirely.[9]

We then report the distribution of student *assignment completion times* using Cumulative Distribution Function (CDF) plots for specific homework assignments, labs, and projects. This CDF can be read as "What fraction of students (y-axis) complete the homework in less than some number of minutes (x-axis)?", capturing how long students take to complete these assignments across the full distribution of completion times.

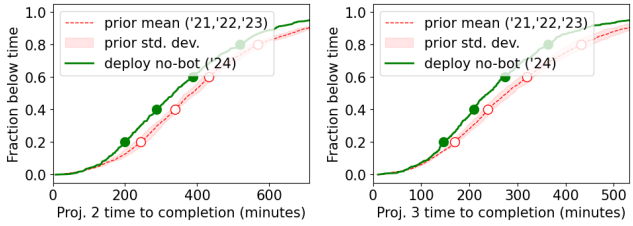
4.3.2 Results. Our primary finding is that student completion times on identical *homework* assignments are substantially faster in our post-deployment semester, Spring 2024, compared to prior semesters (Figure 3a). This effect is not seen in the similar *lab* assignments where the bot is not available (Figure 3b); a smaller, but still substantial, speedup is seen in *projects* (Figure 3c).



(a) HW assignments 1, 2, 3, and 5.



(b) Lab assignments 1, 2, 4, and 8.



(c) Projects 2 and 3.

Figure 3: CDFs of assignment completion times. Circles identify the 20, 40, 60, and 80th percentiles; differences can be read by matching circle markers horizontally. E.g., for HW2, the 60th percentile time across SP21-SP23 is 34 minutes (SD=1.9 min); in SP24 this time is 21 minutes, 7.3 SDs below the mean.

[8] We also conducted an independent analysis looking only at snapshot *counts* rather than summing time deltas. Those results were consistent with the findings we report here, as expected from [28]; we report times here because we find the time metric to be more straightforward to reason about.

[9] Like our choice of completion time *vs.* submission counts, we opted for this method of aggregation because we found it to be the clearest presentation of the effects we observed. We performed the analysis described in this section with individual problems on their own, as well as by comparing individual semesters with their prior (Spring) semesters and comparing across different subsets of problems and prior years; the results we report here are robust to all of the variations we tried, including many others that we do not report on here for space.

The data indicate that the (Bot-available) homework completion times in Spring 2024 are between 25%−50% faster among students in the middle 20 − 80% percentile of completion times. This represents a reduction in time of 3 − 9 Standard Deviations (SD) compared with completion times from prior Spring semesters, a large effect.

In contrast, no-Bot *lab* completion times fall within 1 SD of the prior semesters' mean for the first few labs. Most later labs lack enough consistency across semesters for a robust comparison, but lab 8 does offer a glimpse into one possible outcome: here, students in Spring 2024 overall took several standard deviations *more* time than the mean for students in prior semesters.

Meanwhile, the no-Bot *project* completion times fall somewhere in between: Spring 2024 students completed these projects between 10% − 20% faster than the pre-deployment mean—a speedup about half as large as the Bot-available homeworks.

5 Discussion

Overall, our findings are consistent with a causal link between the availability of our bot and faster homework completion times. In the context of our course (see §4.3.1), reduced completion times are likely to be a result of students reaching a correct solution more quickly, rather than stopping before reaching a correct solution, or making other mistakes, as might be the case in domains where assignments have a greater variety of possible outcomes.

Though we did not carefully examine (and don't make claims about) student learning outcomes here, we have reason to believe that students are not performing dramatically worse after deployment. In particular, though student exam performance is sufficiently inconsistent (due to dramatic exam coverage differences from term to term) that we did not report on it here, it does not appear that Spring 2024 students performed substantially worse than in prior terms. Additionally, the differences in performance on assignments where the Bot was not available (labs and projects) were much smaller. We can't say for certain, but we have not found much evidence for a major decrease in outcomes.

If indeed this reduction in homework time results in little or no learning loss, we can then ask: What implications does this have for CS1 courses? Are there other costs unrelated to learning loss? And what other benefits might accrue?

On the costs side, with most homework help requests going through an LLM, our human TAs may struggle more to stay on top of common challenges among students, leading to a looser feedback loop. However, this challenge could be addressed with new tools that help TAs aggregate over automated help requests—but now with LLM-generated hints and student feedback on whether those hints are helpful. That is, of course, assuming CS1 courses do not elect to reduce staff in response.

Given the mode of access of 61A Bot, one likely change in student learning is a reduction in the ability to read and comprehend error traces. Now that our students have an easy way to get natural language feedback on failed test cases, they are much less incentivized to try to understand why a test case failed by reading the trace.

On the benefits side, instructors could use this extra student time to cover more material, such as effective debugging techniques or reading stack traces. Or, students could continue to simply spend less time on the course. Similarly, TAs could spend less time debugging straightforward homework errors and more time focused on other forms of support.

These factors all point to a need for more research to better understand the actual costs and to inform decisions made in the hope of realizing actual benefits beyond saved student and TA time.

5.1 Limitations & Threats to Validity

Differences in prior preparation could explain why students complete homework more quickly in Spring 2024—but any such differences would also have to explain the *lack* of a decrease in lab completion times. In fact, the consistency in early lab completion times over the 4 semesters we examined suggests that our student populations do not differ significantly in prior ability.

Similarly, differences in course content delivery, staffing, structure, etc., would be expected to impact both labs and homeworks, as lectures and discussion sections for given topics come before labs and homeworks. To our knowledge, no additional homework support was provided in Spring 2024 beyond 61A Bot—no additional hints or support unique to Spring 2024 were offered in lecture or group sessions.

Finally, our study is entirely observational: we had limited levers for randomization, and there could be uncontrolled causes for the effects we observe—for example, it could be that students are using ChatGPT for their homework or projects anyway, despite the prohibition on ChatGPT use and the availability of 61A Bot. ChatGPT was originally released in November 2022, but we observed no similar effect in Spring 2023 compared with prior terms.

6 Conclusions

Our results suggest that 61A Bot reduces demands on staff and helps students complete homework more quickly, with oversized impacts for students who spent the most time on homework—a benefit that might even disproportionately support goals towards inclusion in CS. But, ideally, this type of scaffolding should recede over time as learners become more confident [34]. 61A Bot has not yet clearly achieved this goal.

Guidelines around the inclusion of AI-based course materials and tools suggest that these should only be incorporated when we have a good understanding that their benefits outweigh their costs [1]. Yet even if the primary outcome of 61A Bot is limited to a reduction in homework completion time with no other benefits, we believe that 61A Bot clears this bar—but that further research into improving outcomes and mitigating the costs we have started to expose is critical and *urgent*.

Acknowledgments

This work was made possible by a few generous sources of support: an Inclusion Research Award from Google, and support for 61A-Bot's use of Azure's OpenAI API by Microsoft.

References

[1] Kavita Bala, Alex Colvin, Morten H. Christiansen, Allison Weiner Heinemann, Sarah Kreps, Lionel Levine, Christina Liang, David Mimno, Sasha Rush, Deirdre Snyder, Wendy E. Tarlow, Felix Thoemmes, Rob Vanderlan, Andrea Stevenson Won, Alan Zehnder, and Malte Ziewitz. 2023. Generative Artificial Intelligence for Education and Pedagogy | Center for Teaching Innovation. https://teaching.cornell.edu/generative-artificial-intelligence/cu-committee-report-generative-artificial-intelligence-education

[2] Patrick Bassner, Eduard Frankford, and Stephan Krusche. 2024. Iris: An AI-Driven Virtual Tutor for Computer Science Education. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 394–400. https://doi.org/10.1145/3649217.3653543

[3] 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. (2023).

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[5] Bruno Pereira Cipriano and Pedro Alves. 2023. GPT-3 vs Object Oriented Programming Assignments: An Experience Report. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. 61–67.

[6] Tyne Crow, Andrew Luxton-Reilly, and Burkhard Wuensche. 2018. Intelligent tutoring systems for programming education: a systematic review. In *Proceedings of the 20th Australasian Computing Education Conference*. 53–62.

[7] Franklin Demana and BK Waits. 2000. Calculators in mathematics teaching and learning. *Past, present, and future. In Learning Mathematics for a New Century* (2000), 51–66.

[8] Paul Denny, Brett A Becker, Juho Leinonen, and James Prather. 2023. Chat Overflow: Artificially Intelligent Models for Computing Education-renAIssance or apocAIypse?. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. 3–4.

[9] Paul Denny, Stephen MacNeil, Jaromir Savelka, Leo Porter, and Andrew Luxton-Reilly. 2024. Desirable Characteristics for AI Teaching Assistants in Programming Education. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 408–414. https://doi.org/10.1145/3649217.3653574

[10] Paul Denny, James Prather, Brett A Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N Reeves, Eddie Antonio Santos, and Sami Sarsa. 2023. Computing Education in the Era of Generative AI. *arXiv preprint arXiv:2306.02608* (2023).

[11] 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 *Proceedings of the 24th Australasian Computing Education Conference*. 10–19.

[12] 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 *Proceedings of the 25th Australasian Computing Education Conference*. 97–104.

[13] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1*.

[14] 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 *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–23.

[15] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24)*.

[16] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. ACM, 124–130.

[17] 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 *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. ACM.

[18] Mark Liffiton, Brad Sheese, Jaromir Savelka, and Paul Denny. 2023. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. arXiv:2308.06921 [cs.CY]

[19] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J. Malan. 2024. Teaching CS50 with AI: Leveraging Generative Artificial Intelligence in Computer Science Education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 750–756. https://doi.org/10.1145/3626252.3630938

[20] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J. Malan. 2024. Teaching CS50 with AI: Leveraging Generative Artificial Intelligence in Computer Science Education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1927.

[21] Stephen MacNeil, Joanne Kim, Juho Leinonen, Paul Denny, Seth Bernstein, Brett A Becker, Michel Wermelinger, Arto Hellas, Andrew Tran, Sami Sarsa, et al. 2023. The Implications of Large Language Models for CS Teachers and Students. In *Proc. of the 54th ACM Technical Symposium on Computer Science Education*, Vol. 2.

[22] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from using code explanations generated by large language models in a web software development e-book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 931–937.

[23] Julia M Markel and Philip J Guo. 2021. Inside the mind of a CS undergraduate TA: A firsthand account of undergraduate peer tutoring in computer labs. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 502–508.

[24] Samim Mirhosseini, Austin Z Henley, and Chris Parnin. 2023. What is your biggest pain point? an investigation of cs instructor obstacles, workarounds, and desires. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 291–297.

[25] Chancharik Mitra, Mihran Miroyan, Rishi Jain, Vedant Kumud, Gireeja Ranade, and Narges Norouzi. 2024. Elevating Learning Experiences: Leveraging Large Language Models as Student-Facing Assistants in Discussion Forums. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2* (Portland, OR, USA) *(SIGCSE 2024)*.

[26] Erik Ofgang. 2023. What is Khanmigo? The GPT-4 learning tool explained by Sal Khan. *Tech & Learn* (2023).

[27] Zachary A Pardos and Shreya Bhandari. 2023. Learning gain differences between ChatGPT and human tutor generated algebra hints. *arXiv preprint arXiv:2302.06871* (2023).

[28] Chris Piech, Mehran Sahami, Daphne Koller, Steve Cooper, and Paulo Blikstein. 2012. Modeling how students learn to program. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. 153–160.

[29] James Prather, Juho Leinonen, Natalie Kiesler, Jamie Gorson Benario, Sam Lau, Stephen MacNeil, Narges Norouzi, Simone Opel, Virginia Pettit, Leo Porter, et al. 2024. How Instructors Incorporate Generative AI into Teaching Computing. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 2*. 771–772.

[30] 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. *arXiv preprint arXiv:2304.02491* (2023).

[31] Brent Reeves, Sami Sarsa, James Prather, Paul Denny, Brett A Becker, Arto Hellas, Bailey Kimmel, Garrett Powell, and Juho Leinonen. 2023. Evaluating the Performance of Code Generation Models for Solving Parsons Problems With Small Prompt Variations. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. 299–305.

[32] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*. 27–43.

[33] Brad Sheese, Mark Liffiton, Jaromir Savelka, and Paul Denny. 2024. Patterns of Student Help-Seeking When Using a Large Language Model-Powered Programming Assistant. In *Proceedings of the 26th Australasian Computing Education Conference*. 49–57.

[34] Elliot Soloway, Mark Guzdial, and Kenneth E. Hay. 1994. Learner-centered design: the challenge for HCI in the 21st century. *Interactions* 1 (1994), 36–48.

[35] Ryo Suzuki, Gustavo Soares, Andrew Head, Elena Glassman, Ruan Reis, Melina Mongiovi, Loris D'Antoni, and Bjoern Hartmann. 2017. Tracediff: Debugging unexpected code behavior using trace divergences. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 107–115.

[36] Sierra Wang, John Mitchell, and Chris Piech. 2024. A Large Scale RCT on Effective Error Messages in CS1. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. 1395–1401.

[37] Yu-Chieh Wu, Andrew Petersen, and Lisa Zhang. 2022. Student Reactions to Bots on Course Q&A Platform. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 2*. 621–621.

[38] Zhiqiang Yuan, Junwei Liu, Qiancheng Zi, Mingwei Liu, Xin Peng, and Yiling Lou. 2023. Evaluating instruction-tuned large language models on code comprehension and generation. *arXiv preprint arXiv:2308.01240* (2023).