# NotebookGPT – Facilitating and Monitoring Explicit Lightweight Student GPT Help Requests During Programming Exercises

Samuel George
The University of North Carolina at Chapel Hill
Chapel Hill, NC, USA
sdgeorge@cs.unc.edu

Prasun Dewan
The University of North Carolina at Chapel Hill
Chapel Hill, NC, USA
dewan@cs.unc.edu

## ABSTRACT

The success of GPT with coding tasks has made it important to consider the impact of GPT and similar models on teaching programming. Students' use of GPT to solve programming problems can hinder their learning. However, they might also get significant benefits such as quality feedback on programming style, explanations of how a given piece of code works, help with debugging code, and the ability to see valuable alternatives to their code solutions. We propose a new design for interacting with GPT called Mediated GPT with the goals of (a) providing students with access to GPT but allowing instructors to programmatically modify responses to prevent hindrances to student learning and combat common GPT response concerns, (b) helping students generate and learn to create effective prompts to GPT, and (c) tracking how students use GPT to get help on programming exercises. We demonstrate a first-pass implementation of this design called NotebookGPT.

## CCS CONCEPTS

• Human-centered computing → Graphical user interfaces; Natural language interfaces; • Applied computing → Computer-assisted instruction; Interactive learning environments.

## KEYWORDS

Learning at scale, Computer programming, Intelligent tutoring systems, ChatGPT, GPT

ACM Reference Format:
Samuel George and Prasun Dewan. 2024. NotebookGPT – Facilitating and Monitoring Explicit Lightweight Student GPT Help Requests During Programming Exercises. In 29th International Conference on Intelligent User Interfaces - Companion (IUI Companion '24), March 18–21, 2024, Greenville, SC, USA. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3640544.3645234

## 1 INTRODUCTION & RELATED WORKS

Several recent works have investigated the impact of GPT on programming education for instructors and students [15] [10]. Guo et al. categorize educators into two camps: Those who embrace GPT's usage and those who resist it [10]. Those who embrace GPT identified several potential benefits to students, such as automatic code style feedback and explanations of how code works at various levels via prompting GPT correctly [14]. In addition, they propose that large language models (LLMs) can help students with code completion, code refactoring, code simplification, and debugging code [10].

On the other hand, prior work has shown that entirely unsupervised use of GPT through the chat interface might negatively impact student learning [3] and that LLMs can be prompted in a way that returns offensive language or harmful speech [2].

Therefore, we propose a design for interacting with GPT intended to provide its benefits without drawbacks. We refer to this design as Mediated GPT. The design has four goals to improve the tasks of students, instructors, and future designers and implementors of this idea. (1) Help students learn to create effective prompts for GPT. They may need to learn how to format code, pass errors, or effectively ask questions to get an instructor-like experience from GPT. Prior work shows that optimally formatting questions to ChatGPT is dificult for novice ai-assistant users [16]. (2) Provide students with lightweight facilities to generate effective prompts for GPT. Students would want to minimize the time copying and formatting code prompts to send to GPT. Therefore, our design has features to help students format their working code to send to GPT, making the interaction easier than using ChatGPT directly. (3) Give instructors the ability to modify the responses programmatically. Course policies are likely to consider code reuse from humans or AI as plagiarism. Therefore, our design allows mediation of output to remove code from GPT responses. Such mediation can also prevent other hindrances to student learning and mitigate response concerns, such as offensive language. (4) Track how students use GPT in programming exercises. Understanding how students use GPT during programming exercises has several benefits, such as potentially modifying the design to optimize common use cases, analyzing student interactions with GPT to inform instructors about typical problems in coding assignments, allowing instructors to address common topics in GPT interactions and go into more depth on them during class, and calculating the cost to maintain access to GPT services for a class. Data on how students use GPT during programming exercises (to our knowledge) has yet to be published [10], and this data would be invaluable to the programming education community.

Next, we present a first-pass implementation of our design called NotebookGPT and discuss how we derived its features to meet the given goals.


IUI Companion '24, March 18–21, 2024, Greenville, SC, USA

ACM ISBN 979-8-4007-0509-0/24/03
https://doi.org/10.1145/3640544.3645234

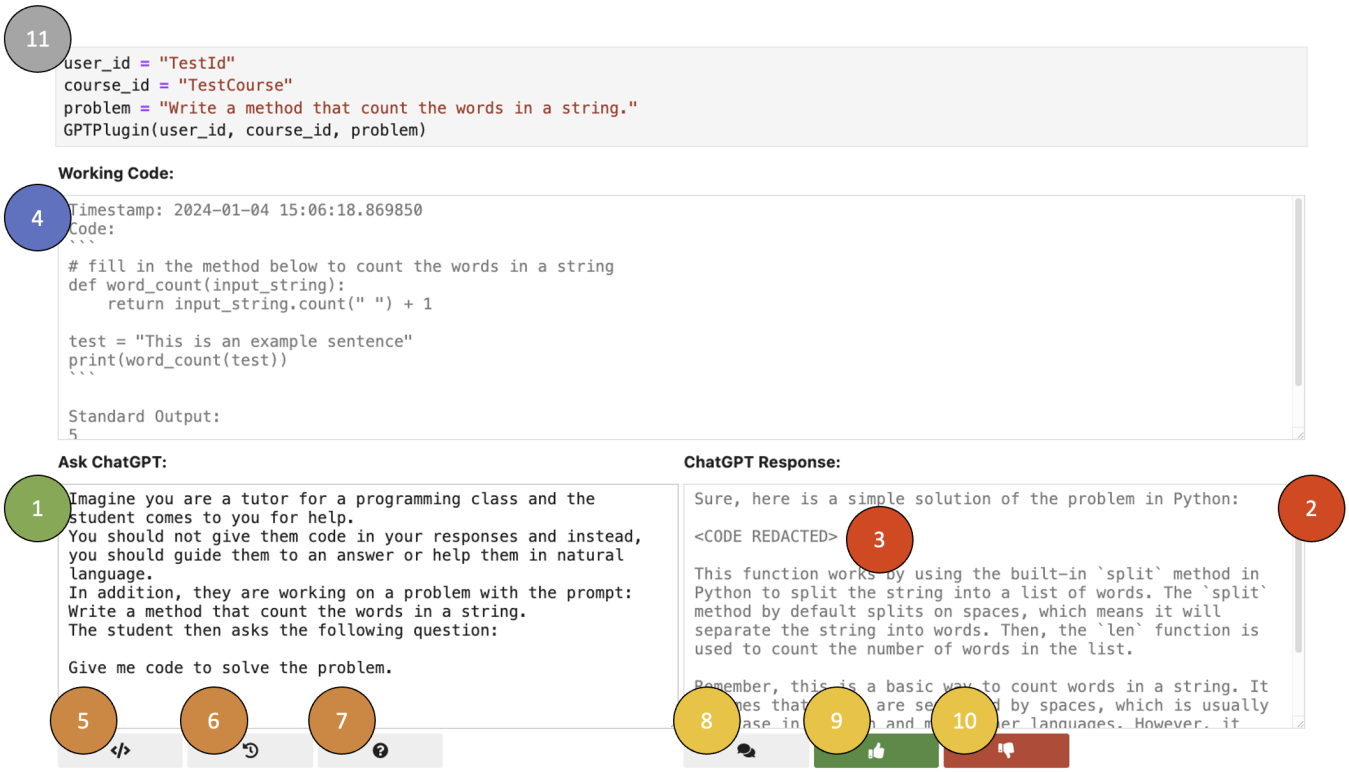


Figure 1: This figure shows a screen capture of the NotebookGPT GUI running in a Jupyter Notebook. In the interface, the student can compose a message to GPT-4 (Features 1,8), receive mediated responses (Features 2-3), see their working code (Feature 4), get help composing effective prompts (Features 5-7), give feedback about GPT's response (Features 9-10), and configure how their data are tracked and on what problem they are working (Feature 11).

## 2 DESIGN IMPLEMENTATION – NOTEBOOKGPT

The first critical points are how students will interact with GPT and what domain we target for our implementation.

### 2.1 Interaction & Domain

Several new integrated development environment (IDE) plugins that facilitate lightweight interactions with LLMs have recently been released. These include Amazon CodeWhisperer [6], Tabnine [11], and Github CoPilot [7]. Their interaction paradigm provides automatic code completion and suggestions as programmers write code. Despite many implementations using this design, prior work identifies many dificulties, such as effectively providing these hints in place in the coding widow and ensuring users recognize suggestions within the GUI [13]. More importantly, they complete student code, defeating the purpose of an assignment. Students in our university and others are expected to code independently on individual assignments but can have natural language discussions on their solutions with other students and instructors. Further, these user interfaces offer implicit, automatic help and thus cannot be used to determine if the programmers want it. In fact, some programmers felt they could have completed code on their own more eficiently [13]. For these reasons, we implemented our interaction with GPT as a chat-based GUI, which offers an "out band" non-coding window to offer NLP help only when explicitly requested.

Regarding domain, Jupyter Notebooks are a popular programming environment that benefits students' learning programming [8]. They have been used in a variety of courses to teach big data analysis, high-performance computing, and engineering [4] [5] [1]. Therefore, we implemented our chat-based GUI in Jupyter Notebooks as a plugin. We call this implementation NotebookGPT and, to our knowledge, this is the first GUI for GPT in Jupyter Notebooks targeting students learning programming.

### 2.2 NotebookGPT GUI

Figure 1 shows our NotebookGPT GUI. The numbers (1-11) represent the critical features of the UI, discussed and motivated below.

2.2.1 Mediated Chat Features. A student can formulate a message to GPT – we use GPT-4 in our current implementation, but the version of GPT is configurable – using the text box on the bottom left (Feature 1) and then send a message using the button beneath the bottom right text box (Feature 8). The response from GPT comes back in the box on the bottom right (Feature 2). This response, however, is filtered to remove whole blocks of code (Feature 3) to prevent students from simply copying and pasting code. These code blocks are identified via a regular expression, as GPT responses put

code blocks in a regular format. Once identified, a "Code Redacted" string replaces these blocks. Thus, students must use the natural language around the code for advice. This programmatic modification represents our first step toward mediating GPT responses to ensure GPT does not hinder students' learning. We also intend to explore potential useful replacements for these "Code Redacted" blocks in future versions of NotebookGPT. In addition, we will also include programmatic filtering of offensive language, and there is already work that uses generative AI to detect such speech with mixed results [12]. However, more research and effort is needed to implement these features in our interface. In addition to programmatic modification of GPT responses, other work has explored human modification of GPT responses [9]. We are also considering future enhancements that allow instructors to modify automatically flagged problematic responses if a reliable programmatic-only approach to replace code or remove offensive speech proves challenging. All these features represent controlled interaction with a generative AI via output mediation. The student can give positive or negative feedback about the helpfulness of the mediated response by clicking the "thumbs up" (feature 9) or "thumbs down" (feature 10) buttons, respectively. Beyond interacting with GPT, we provide additional features to help enhance student input interaction with GPT, making it more lightweight.

2.2.2 Code Context & Prompting. In addition to the chat window, our GUI provides a box that shows the code, standard output, and standard error output for the student's current working cell (Feature 4). The student can then use the buttons below the GPT message box to generate prompts automatically from the working code (Features 5-7). Clicking the far left button (Feature 5) will copy the working code, standard output, and standard error output into the message box and automatically generate a prompt explaining the code format to GPT (Not Shown). In addition, the GUI tracks all the activities inside the Jupyter Notebook and keeps a history of these activities. Clicking the middle button (Feature 6) will generate a prompt from the student's code history containing their last five code edits – a number chosen based on intuition. Finally, the far right button (Feature 7) will generate a prompt that the student can combine with their question about the problem they are working on and hypothetically get instructor-like responses from GPT (See prompt in Feature 1). Finally, automatic prompt generation allows the student to send a message to GPT with a code history (Feature 6). This history can open new potential use cases like asking ChatGPT to give additional code implementations for a problem they have solved in multiple ways – send the history of different solutions in the same cell – or generate a next step hint given the student's current working trajectory. These features provide input mediation to help students interact more effectively with generative AI and represent a first cut at meeting the prompting goals for our design. To understand how students use the chat, prompting, and feedback features within the GUI, we track all student actions within it.

2.2.3 Usage Tracking. NotebookGPT records all the interactions with the GUI and the activity within the Jupyter Notebook. To distinguish different students' data and assignments, we provide a few input variables to the NotebookGPT Python package (GPTPlugin) that allows data organization by student and course (feature 11). In addition, a problem variable can be passed to the plugin on initialization. This variable is optional, and the other features, such as prompting, will adapt to the presence of this field. These variables allow us to collect data about our GUI's use and generate more effective prompts based on context.

## 3 SUMMARY, DISCUSSION & FUTURE WORKS

We have introduced here a new design, Mediated GPT, for providing GPT to students learning programming. In addition, we developed a first-pass implementation of this design and highlighted its critical features.

While we intend for NotebookGPT to provide students with quality programming assistance during assignments, we are concerned about a few potential shortcomings. These shortcomings include but are not limited to students not finding GPT helpful and preferring traditional assistance, such as ofice hours, or finding the user interface's use of extra screen space cumbersome and abandoning its use entirely. In addition, we are relying on the honor code to prevent students from interacting with GPT or other generative AIs outside of NotebookGPT. Students who were not considering the use of GPT for coding exercises may find it so useful that they would use an unmediated version of it to do undetected plagiarism.

At least two forms of evaluation are possible with our implementation. First, it is possible to deploy it in a researcher-controlled lab user study to observe the effectiveness of our mediation and user interface. This controlled study can help gauge its ability to meet our design goals and evaluate our potential shortcomings. Second, it can be made available in an uncontrolled field study using tracked data to understand its benefits.

## ACKNOWLEDGMENTS

This work was funded in part by NSF awards OAC 1829752 and 1924059. The detailed and insightful comments on the paper improved the discussion of our technical contributions.

## REFERENCES

[1] Alberto Cardoso, Joaquim Leitão, and César Teixeira. 2019. Using the Jupyter Notebook as a Tool to Support the Teaching and Learning Processes in Engineering Courses. In The Challenges of the Digital Transformation in Education, Michael E. Auer and Thrasyvoulos Tsiatsos (Eds.). Springer International Publishing, Cham, 227–236.

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG]

[3] Marian Daun and Jennifer Brings. 2023. How ChatGPT Will Change Software Engineering Education. In Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (, Turku, Finland,) (ITiCSE 2023). Association for Computing Machinery, New York, NY, USA, 110–116. https://doi.org/10.1145/3587102.3588815

[4] Roland DePratti. 2019. Using Jupyter Notebooks in a Big Data Programming Course. J. Comput. Sci. Coll. 34, 6 (apr 2019), 157–159.

[5] Ben Glick and Jens Mache. 2018. Using Jupyter Notebooks to Learn High-Performance Computing. J. Comput. Sci. Coll. 34, 1 (oct 2018), 180–188.

[6] Amazon Inc. 2024. Amazon CodeWhisperer. https://aws.amazon.com/codewhisperer/ Accessed: 2024-01-02.
[7] Github Inc. 2024. Github CoPilot. https://github.com/features/copilot Accessed: 2024-01-02.
[8] Jeremiah W. Johnson. 2020. Benefits and Pitfalls of Jupyter Notebooks in the Classroom. In Proceedings of the 21st Annual Conference on Information Technology Education (Virtual Event, USA) (SIGITE '20). Association for Computing Machinery, New York, NY, USA, 32–37. https://doi.org/10.1145/3368308.3415397
[9] Mason Laney and Prasun Dewan. 2024. Human-AI Collaboration in a Student Discussion Forum. In Companion Proceedings of the 29th International Conference on Intelligent User Interfaces.
[10] 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 2023 ACM Conference on International Computing Education Research - Volume 1 (Chicago, IL, USA) (ICER '23). Association for Computing Machinery, New York, NY, USA, 106–121. https://doi.org/10.1145/3568813.3600138
[11] Codota Dot Com Ltd. 2024. Tabnine. https://www.tabnine.com/ Accessed: 2024-01-02.
[12] Sagi Pendzel, Tomer Wullach, Amir Adler, and Einat Minkov. 2023. Generative AI for Hate Speech Detection: Evaluation and Findings. arXiv:2311.09993 [cs.CL]
[13] 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 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). 185–195. https://doi.org/10.1109/ICSE-SEIP58684.2023.00022
[14] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. arXiv:2302.11382 [cs.SE]
[15] Ramazan Yilmaz and Fatma Gizem Karaoglan Yilmaz. 2023. Augmented intelligence in programming learning: Examining student views on the use of ChatGPT for programming learning. Computers in Human Behavior: Artificial Humans 1, 2 (2023), 100005. https://doi.org/10.1016/j.chbah.2023.100005
[16] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (, Hamburg, Germany,) (CHI '23). Association for Computing Machinery, New York, NY, USA, Article 437, 21 pages. https://doi.org/10.1145/3544548.3581388