GPTutor: Interactive Learning with Structured Debugging Feedback

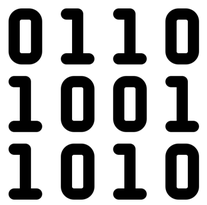
Prototyping with LLMs

Ruoxi Shang

Spring 2023

Background

Learning Barriers and Pain Points for Coding



Understanding complex programming concepts

Lack of immediate help from experienced mentors

Hit-or-miss process of debugging

Lack of motivation to learn about “why”

Examples of Debugging Support

HelpMeOut by Hartmann et al. 2010

A social recommender system that aids the debugging of error messages by suggesting solutions that peers have applied in the past

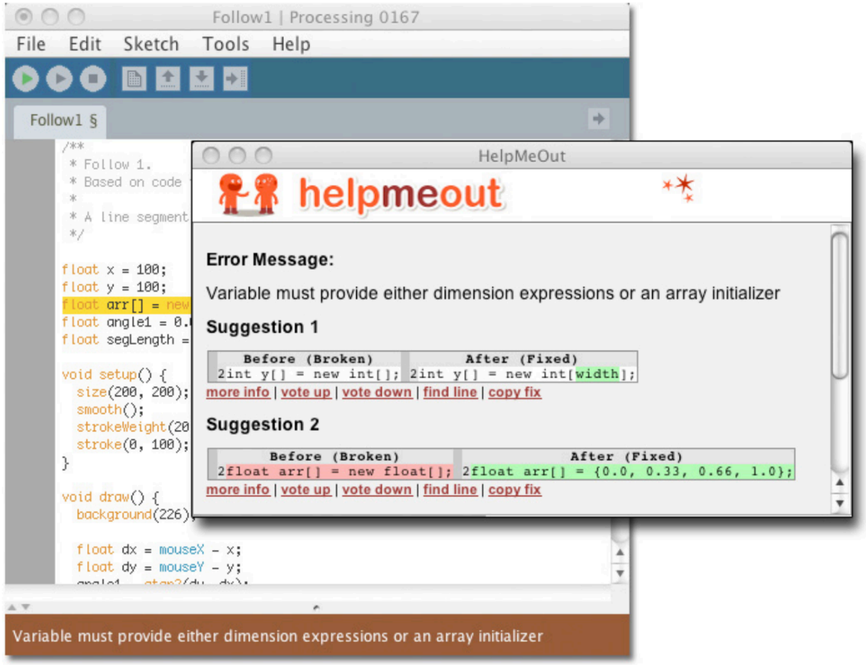


Figure 2. The HelpMeOut Suggestion Panel shows possible corrections for a reported compiler error.

StratCel by Grigoreanu et al. 2010

A strategy-based add-in for Excel that automatically generating a list of to-do items and providing actions related to managing a task list

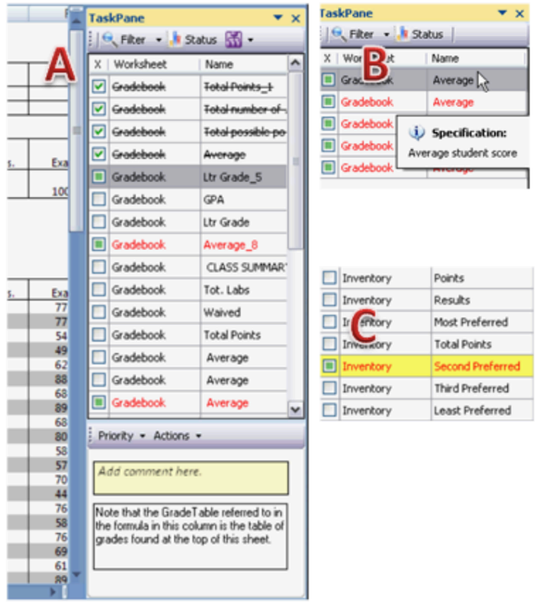
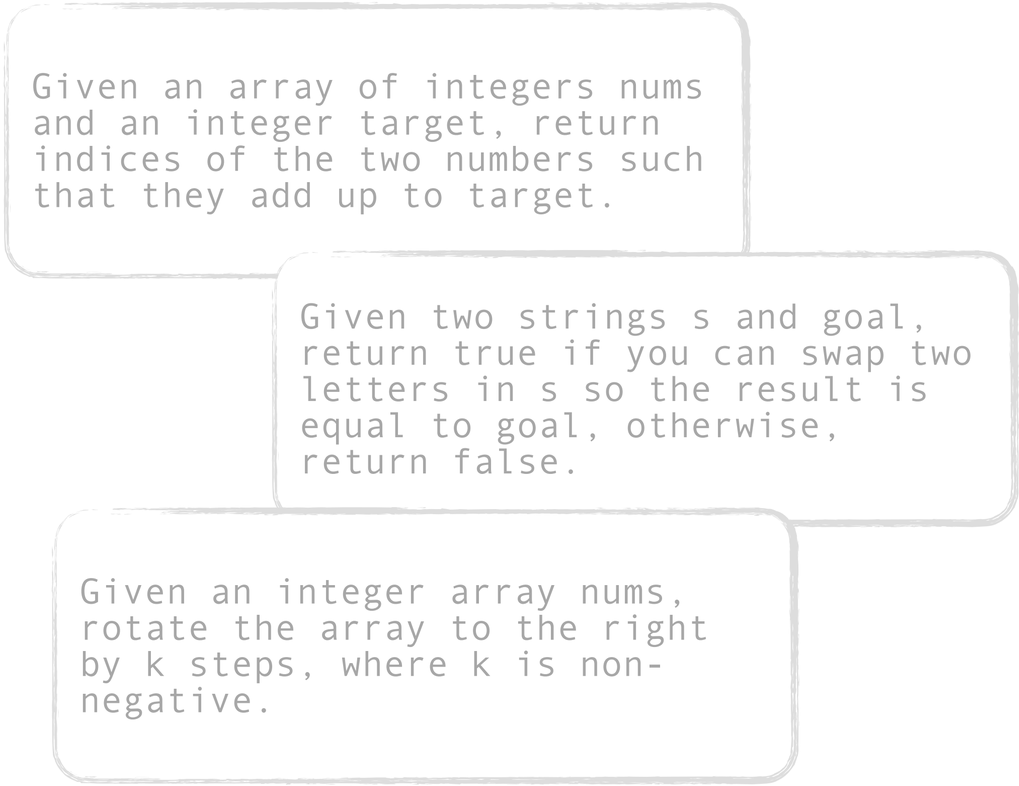


Figure 1. (a) The to-do list task pane is automatically populated with consolidated items and their properties (e.g., worksheet name, a default item name). The user can mark each item as "done" (e.g., Total Points_1), "unchecked" (e.g., GPA), or "to-do" (e.g., Average_8). Other basic to-do list management capabilities include adding a comment, (b) filtering on a status, and (c) assigning priorities to items (the darker the yellow, the higher the priority).

Problem Scope

Overview

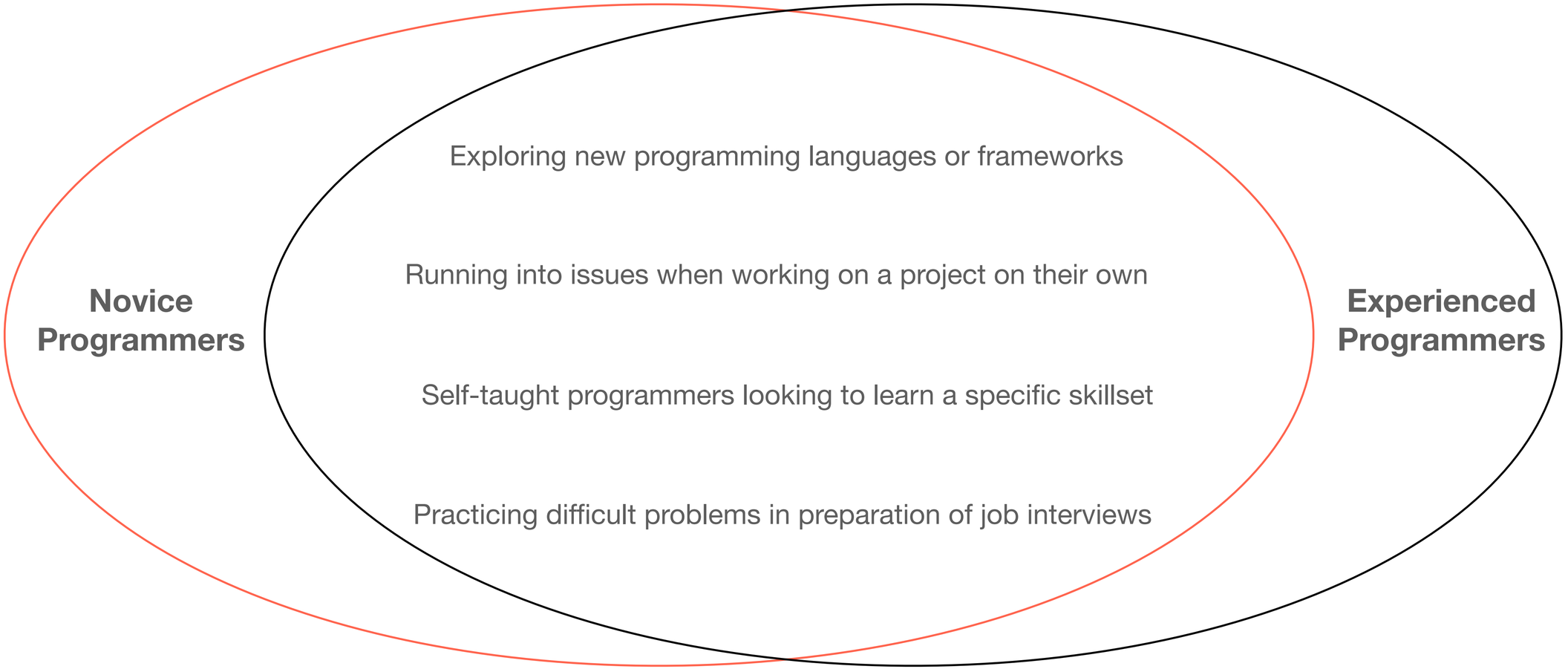
- Facilitate a learning environment that encourages users to value **iteration**, **experimentation**, and **feedback** as essential components of the problem-solving process.
- Support learners with **functional** and **performance** issues for short, isolated coding problems.



LeetCode

Target Users

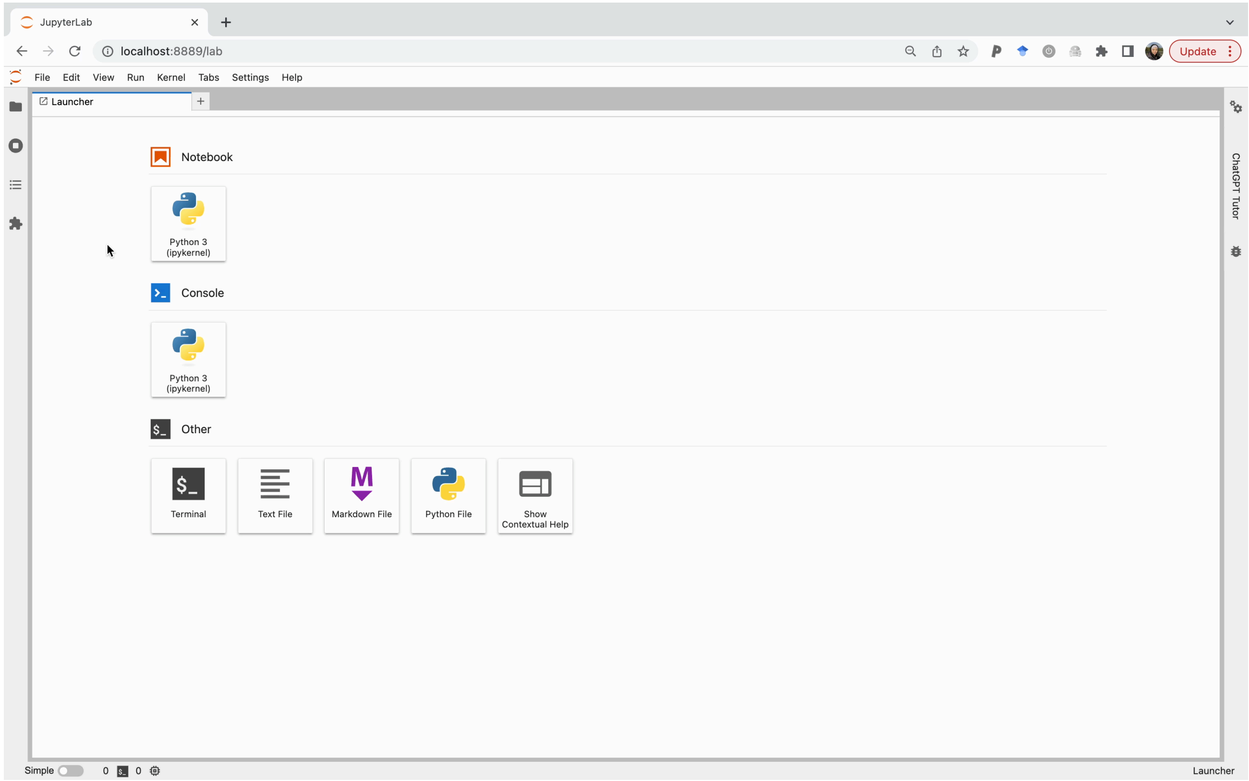
Novice Programmers & Experienced Programmers



Features and functionalities

Jupyter Lab Integration

- Jupyter Widget API framework for easy tooling extensibility
- Iterative code experimentation with failure-driven learning paradigm



Features and functionalities

Contextualized and Structured guidance

- Diagnostic questioning targeting specific code constructs and runtime behavior
- Step-wise reasoning decomposition
- Progressive hint escalation based on error classification
- Guided debugging through Socratic dialogue

Hint 1: Are you looping through the array more than once?

Hint 2: Are you checking if the two indices you are comparing are the same?

Hint 3: Are you checking if the sum of the two indices you are comparing is equal to the target?

Solution: The current code is not optimal because you are looping through the array more than once. To optimize the code, you should use a single loop and use a dictionary to store the indices of the numbers that you have already seen. Then, you can check if the difference between the target and the current number is in the dictionary. If it is, then you have found the two indices that add up to the target.

Hint 1: Have you considered the case where the same element is used twice?

Hint 2: Are you sure that the loop is iterating over the correct elements?

Hint 3: Are you sure that the loop is iterating over the entire array?

Solution: It looks like your code is not considering the case where the same element is used twice. You need to make sure that the loop is iterating over the entire array and that the two elements you are comparing are not the same. You can do this by adding an additional condition to your loop that checks if the two elements are not the same.

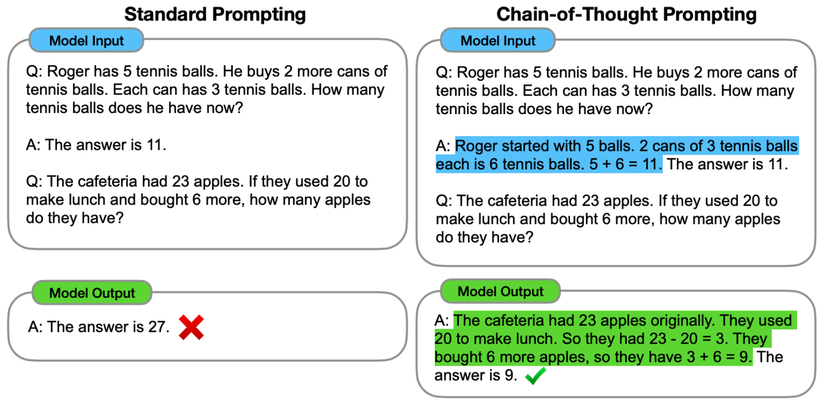
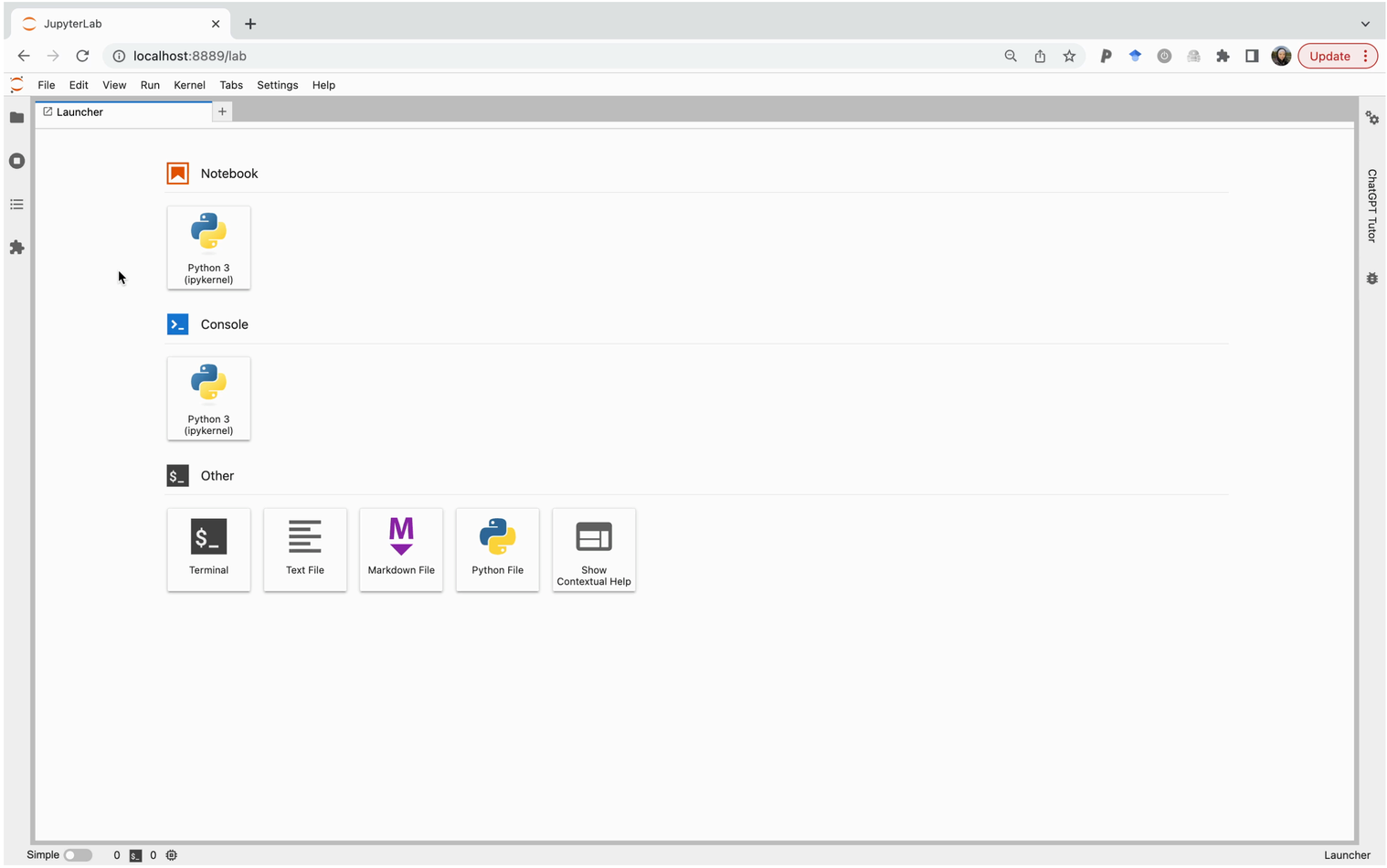


Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.

Wei et al. 2022

Demo

Full Video Walkthrough

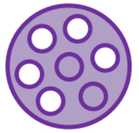


Click to play video

Reflection

Rethinking engineering methods and contributions

- What part of the design and engineering work can LLMs automate?
- How does the prevalence of generative AI tools impact the way we think about novelty in contributions?
- How might the accelerated development process impact the traditional engineering methods?
- How might this impact the traditional user-centered research process aimed to build adaptive and personalized experiences?



Includes novel techniques

Achieves novel functionality

Figure 3: Interactive systems that include both novel functionality (i.e., the outer *what*) and novel techniques (i.e., the inner *how*) can often be motivated and validated in either contribution.

Includes known techniques

Achieves novel functionality

Figure 4: When underlying techniques are known (i.e., the inner *how*), the question is whether their combination in new functionality is a significant contribution (i.e., the outer *what*).

Includes novel techniques

Achieves known functionality

Figure 5: When applied in known overall functionality (i.e., the outer *what*), the question is whether implications of novel inner techniques are a significant contribution (i.e., the inner *how*).

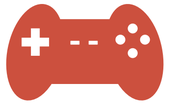
Image adopted from Fogarty, James. "Code and contribution in interactive systems research." Workshop HCITools: Strategies and Best Practices for Designing, Evaluating and Sharing Technical HCI Toolkits at CHI. 2017.

Future Work

How to improve

Reduce friction in interaction

Explore programming learning literature



Gamification of learning process

Adapt to different workflows, expertise level, and stages of coding