Code Generation, guest lecture

Human Interactions with Code Gen Models

Sherry Tongshuang Wu

HCII/LTI

@tongshuangwu / sherryw@cs.cmu.edu

Outline

Evaluation:

Metrics inspired by human-human interactions

Quantitative and qualitative user modeling

Design and implementation:

The impact of UI

The focus on user needs

How do you know a code gen model is good?

Metrics evaluating code gen models

Outcomes	Human-AI (Copilot)
Quality	vs. Human-Human: more lines of code deleted in next session (lower quality) [9] vs. Human Solo: significantly improve correctness score and reduce encountered errors for novice students [10] vs. Human Solo: no significant difference in task success [11] or task success rate in given time [12]
Productivity	vs. Human-Human: more lines of added code [9] vs. Human Solo: 55.8% reduction in completion time [11] vs. Human Solo: significantly increase task completion and reduce task completion time for novice students [10] vs. Human Solo: no significant difference in the task completion rate in given time [12]
Satisfaction	vs. Human Solo: higher self-ratings of satisfaction [12, 16, 17]
Learning	vs. Human Solo: no significant difference in immediate and retention post-test performance of novices, students with more prior experiences have more learning gains from AI code generator [10]
Cost	No experiment yet. Vaithilingam et al. [12], Bird et al. [16] hypothesized that human-AI may lead to more unnecessary debugging vs. Human Solo

How do you know a code gen model is **useful**?

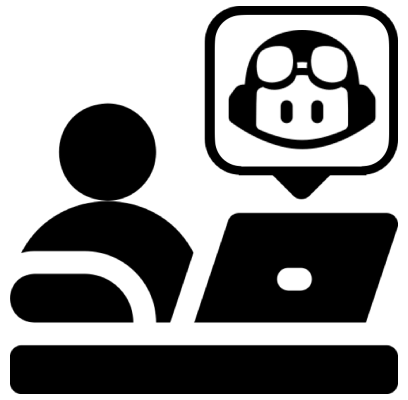
A: define metrics more specific for usability tests!

Human-Human vs. Human-AI pAIr Programming

Human-AI pAIr Programming:

Programmer and LLM work together at the same computer, solving the same task.

Copilot, an LLM-powered programming assistance tool, advertises itself as **"your AI pair programmer"**

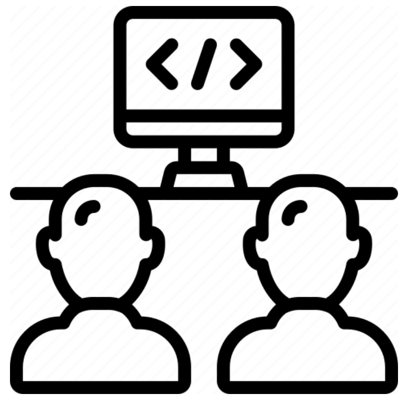


Human-Human Pair Programming

Two programmers work together on the same task using a single device. [Beck, 1999]

Driver: performs the coding

Navigator: aids in planning, reviewing, debugging



Similar to human-human co-programming, human-AI pair programming involves a lot of study and metrics that should capture the interaction aspect.

Qianou Ma, Tongshuang Wu, and Kenneth Koedinger. "Is AI the better programming partner? Human-Human Pair Programming vs. Human-AI pAIr Programming." *AIED 2023 workshop*

Metrics for human-AI interaction

Outcomes	Human-AI (Copilot)
Quality	☹ vs. Human-Human: more lines of code deleted in next session (lower quality) [9] ☺ vs. Human Solo: significantly improve correctness score and reduce encountered errors for novice students [10] 😐 vs. Human Solo: no significant difference in task success [11] or task success rate in given time [12]
Productivity	☺ vs. Human-Human: more lines of added code [9] ☺ vs. Human Solo: 55.8% reduction in completion time [11] ☺ vs. Human Solo: significantly increase task completion and reduce task completion time for novice students [10] 😐 vs. Human Solo: no significant difference in the task completion rate in given time [12]
Satisfaction	☺ vs. Human Solo: higher self-ratings of satisfaction [12, 16, 17]
Learning	😐 vs. Human Solo: no significant difference in immediate and retention post-test performance of novices, students with more prior experiences have more learning gains from AI code generator [10]
Cost	No experiment yet. Vaithilingam et al. [12], Bird et al. [16] hypothesized that human-AI may lead to more unnecessary debugging vs. Human Solo

Outcomes	Human-Human vs. Human Solo	Human-AI (Copilot)
Quality	🙂 significantly lower defect density for complex code [7] 😐 no difference for simpler code [7] 🙂 significantly higher percentage of test cases passed [8]	☹️ vs. Human-Human: more lines of code deleted in next session (lower quality) [9] 🙂 vs. Human Solo: significantly improve correctness score and reduce encountered errors for novice students [10] 😐 vs. Human Solo: no significant difference in task success [11] or task success rate in given time [12]
Productivity	☹️ significantly fewer lines of code per person hour writing simpler code [7] 😐 no significant difference writing more complex code [7] 🙂 29% shorter time to complete task (pair speed advantage = 1.4) [13]	🙂 vs. Human-Human: more lines of added code [9] 🙂 vs. Human Solo: 55.8% [illegible] 🙂 vs. Human Solo: significantly increase task completion and reduce task completion time for novice students [10] 😐 vs. Human Solo: no significant difference in the task completion rate in given time [12]
Satisfaction	🙂 higher self-ratings of satisfaction [14] ☹️ students with greater self-confidence and self-efficacy less enjoy the pair programming experience [15]	🙂 vs. Human Solo: higher self-ratings of satisfaction [12, 16, 17]
Learning	🙂 higher grades, exam scores [18], and retention [19] 🙂 significantly higher gains in exam performance in female students than male students [20]	😐 vs. Human Solo: no significant difference in immediate and retention post-test performance of novices, students with more prior experiences have more learning gains from AI code generator [10]
Cost	☹️ increased management workload to match, schedule a pair, resolve collaboration conflict, assess individual contributions, etc. [21] 🙂 reduced teaching staff workload (grading one assignment from a pair) [8]	No experiment yet. Vaithilingam et al. [12], Bird et al. [16] hypothesized that human-AI may lead to more unnecessary debugging vs. Human Solo

TL;DR: there are many metrics that we can adapt from human-human interactions!

Multi-dimension reflection on quality

Connection to real-world impact (exam)

Reflection on real user benefit (teaching staff workload!)

Example Usability Metric: Syntactic similarity metric

Reference Code Snippet

```
def even_odd_count(num):
  even_count = 0
  odd_count = 0
  for i in str(abs(num)):
      if int(i)%2==0:
          even_count +=1
      else:
          odd_count +=1
  return (even_count, odd_count)
```

Generated Code Snippet

```
def even_odd_count(num):
  even_count = 0
  odd_count = 0
  for i in str(num):
      if int(i) % 2 == 0:
          even_count += 1
      else:
          odd_count += 1
  return even_count, odd_count
```

Functional Metric

pass = **0**

Similarity Metric

edit similarity = **0.93**

Human preference

preference = **0.9**

Figure 1: In the example above (counting even and odd numbers), code suggested by a model fails unit tests but is deemed useful by programmers because adding a short check (*abs* value) fixes the generation.

"While correctness captures high-value generations, programmers still rate **code that fails unit tests as valuable if it reduces the overall effort needed to complete a coding task**. Finally, we propose a hybrid metric that combines functional correctness and syntactic similarity and show that it achieves a 14% stronger correlation with value and can therefore better represent real-world gains when evaluating and comparing models."

Dibia, Victor, et al. "Aligning Offline Metrics and Human Judgments of Value for Code Generation Models." ACL 2023

How do you know a code gen model is **useful for…**

A: Quantitative and qualitative modeling!

Researchers

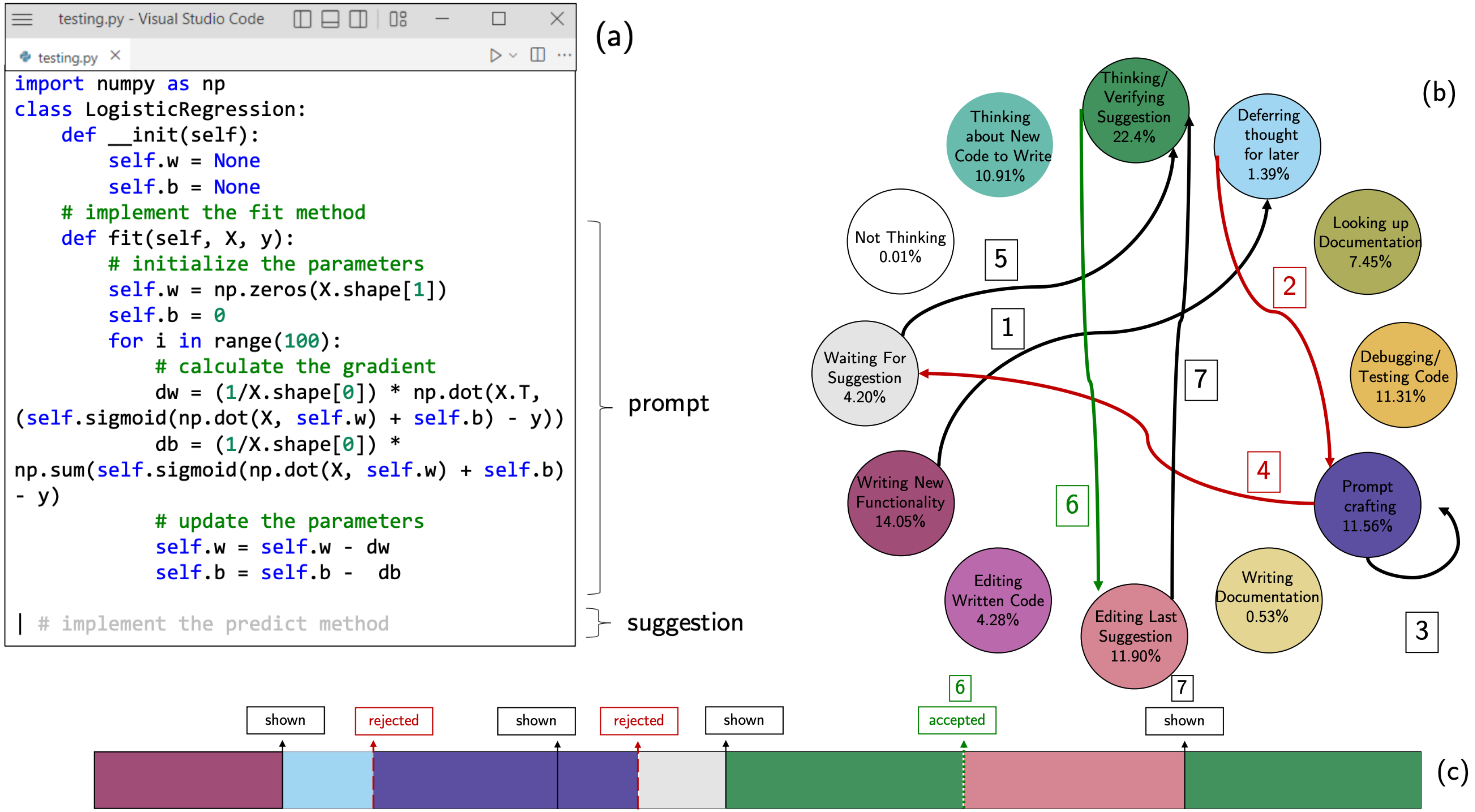
CS1 students

Junior engineers

Senior engineers

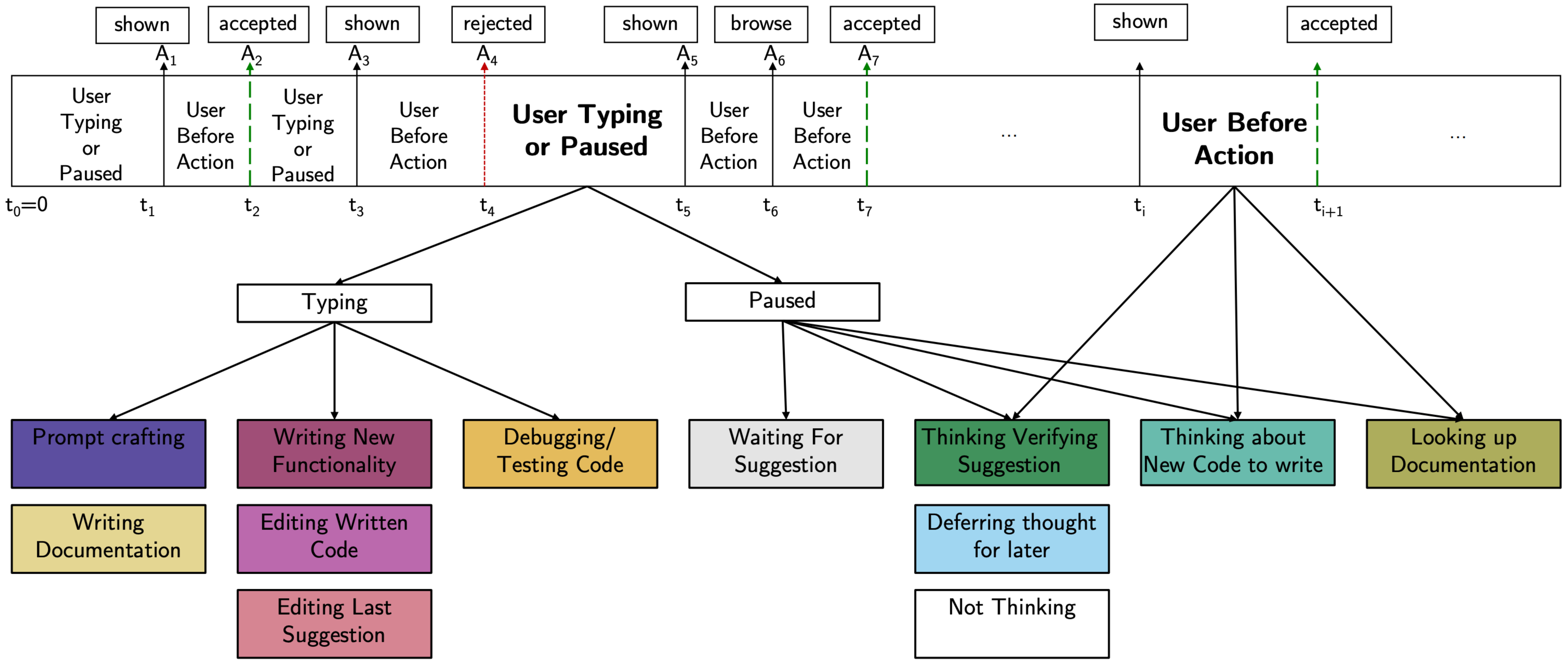
…

Study humans quantitatively: User modeling on Clickstream



Mozannar, Hussein, et al. "Reading between the lines: Modeling user behavior and costs in AI-assisted programming." *CHI 2024*

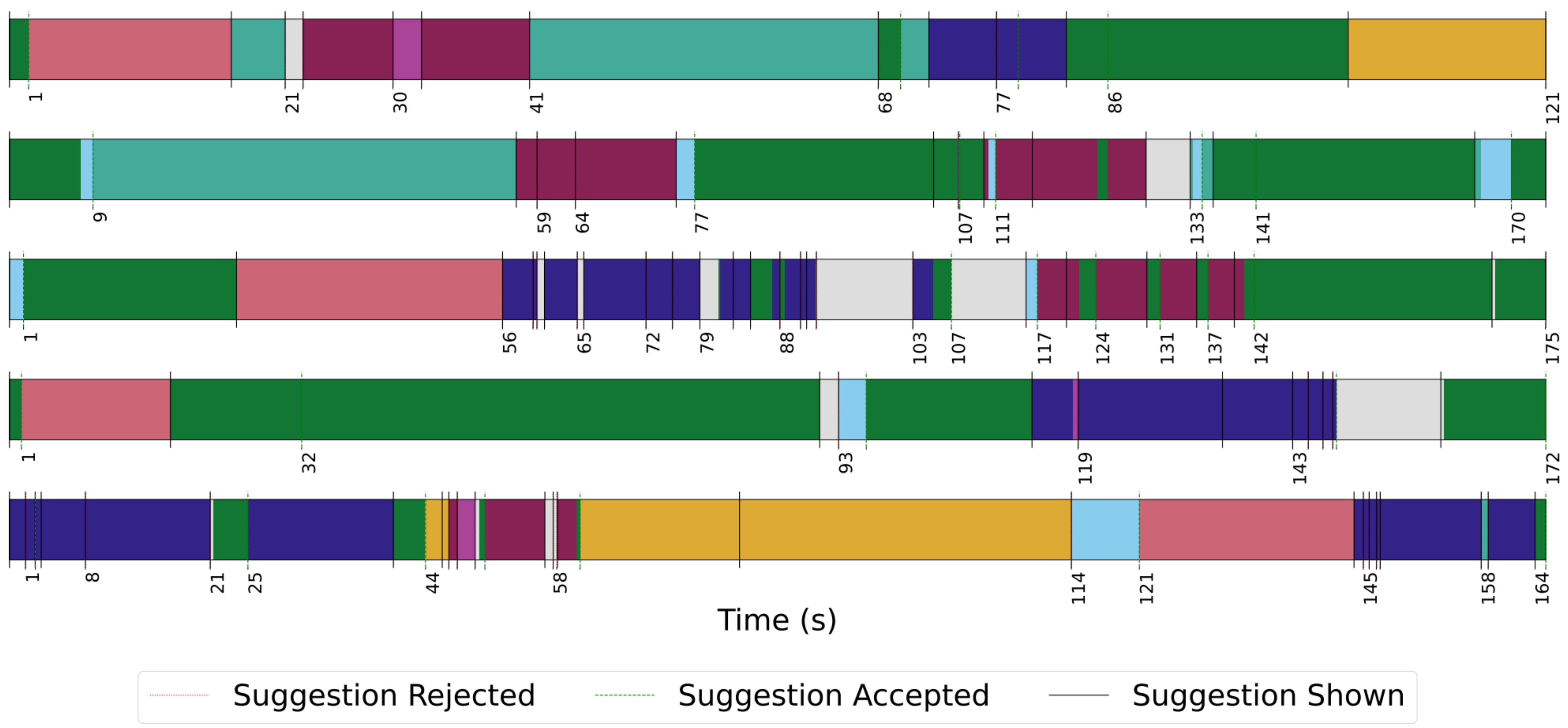
Study humans quantitatively: User modeling on clickstream



Can define and classify what happens in programmer actions (here idle times)

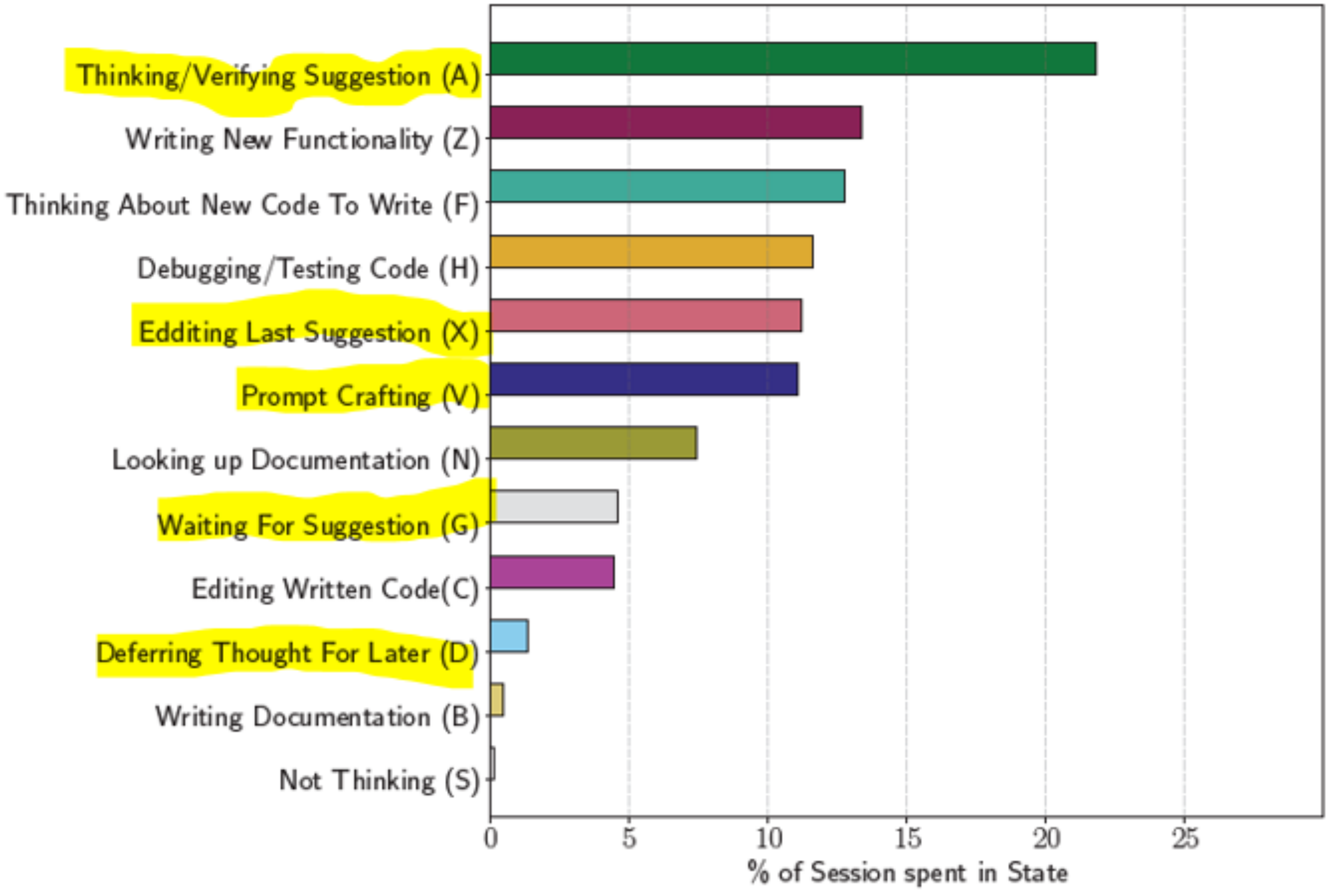
Study humans quantitatively: User modeling on clickstream

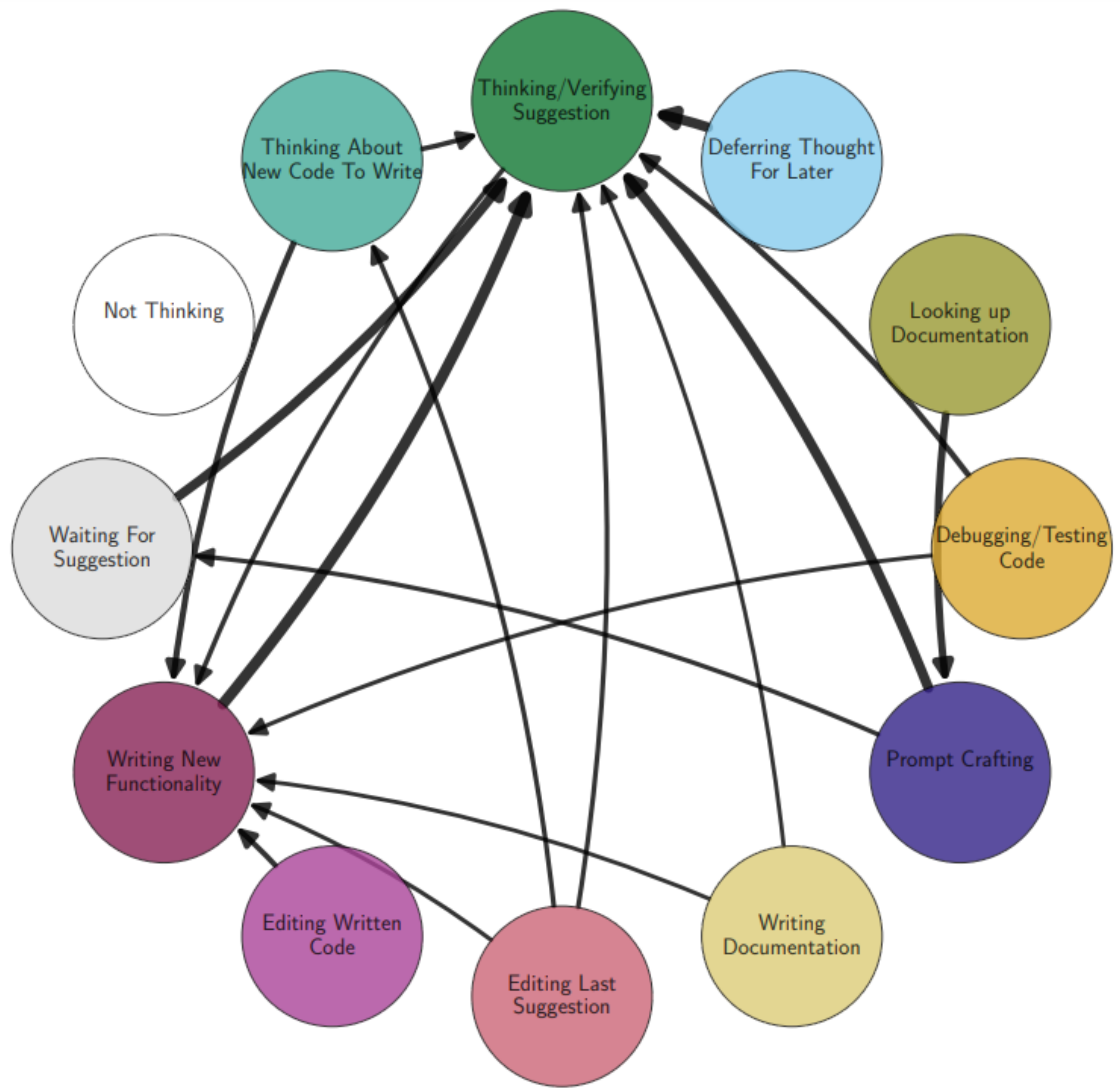
Can uncover interesting patterns for each individual people!



(a) Individual CUPS timelines for 5/21 study participants for the first 180 secs show the richness of and variance in programmer-CodeRec interaction.

Human behavior in aggregation can show avg use patterns

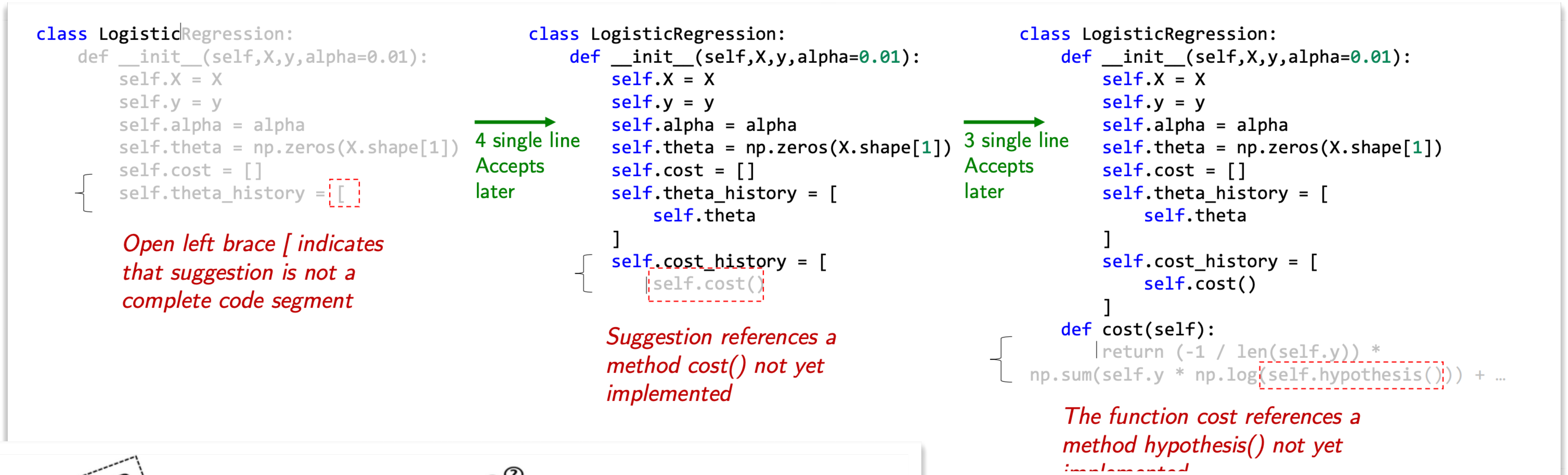


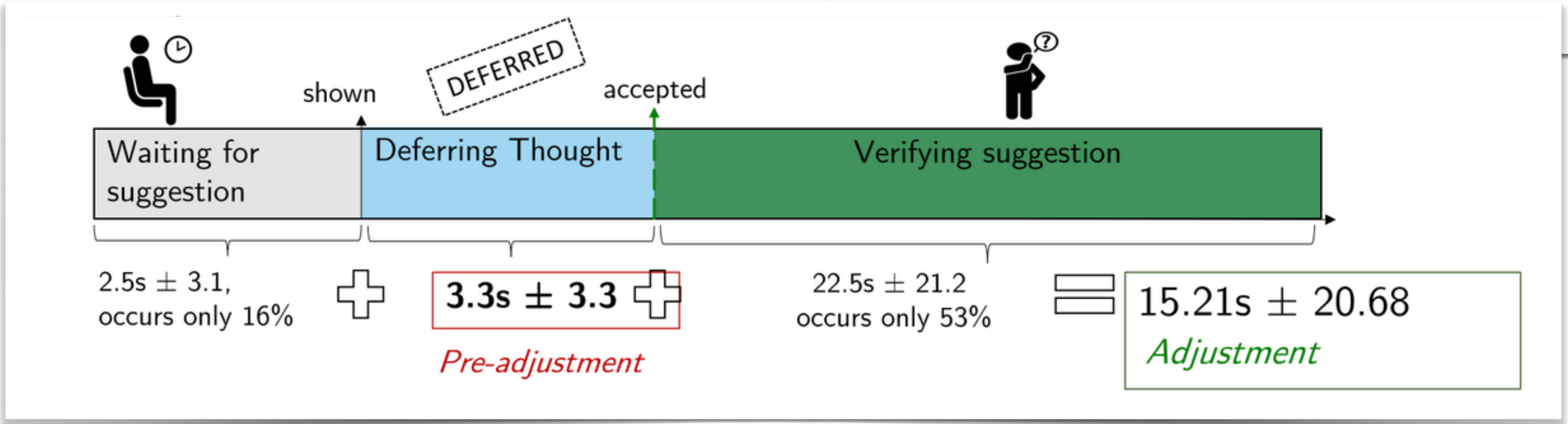


"In a study with 21 programmers, we saw that the most time intensive state is verifying suggestions, and Copilot related states (yellow highlight) occupy on average 51% of task time."

Clickstream can uncover interesting patterns!

"Copilot often forces programmer to accept a sequence of suggestions in a row, teasing them to show the full function/class body, which makes them verify suggestion after accepting them (rather than before)"





Also reveal issues in metrics

Outcomes	Human-Human vs. Human Solo	Human-AI (Copilot)
Productivity	😞 significantly fewer lines of code per person hour writing simpler code [7] 😐 no significant difference writing more complex code [7] 😊 29% shorter time to complete task (pair speed advantage = 1.4) [13]	😊 vs. Human-Human: more lines of added code [9] 😊 vs. Human Solo: 55.8% reduction in completion time [11] 😊 vs. Human Solo: significantly increase task completion and reduce task completion time for novice students [10] 😐 vs. Human Solo: no significant difference in the task completion rate in given time [12]

Human-human – Variance in metrics!

Time and accomplishment? twice the duration, the person-hours required

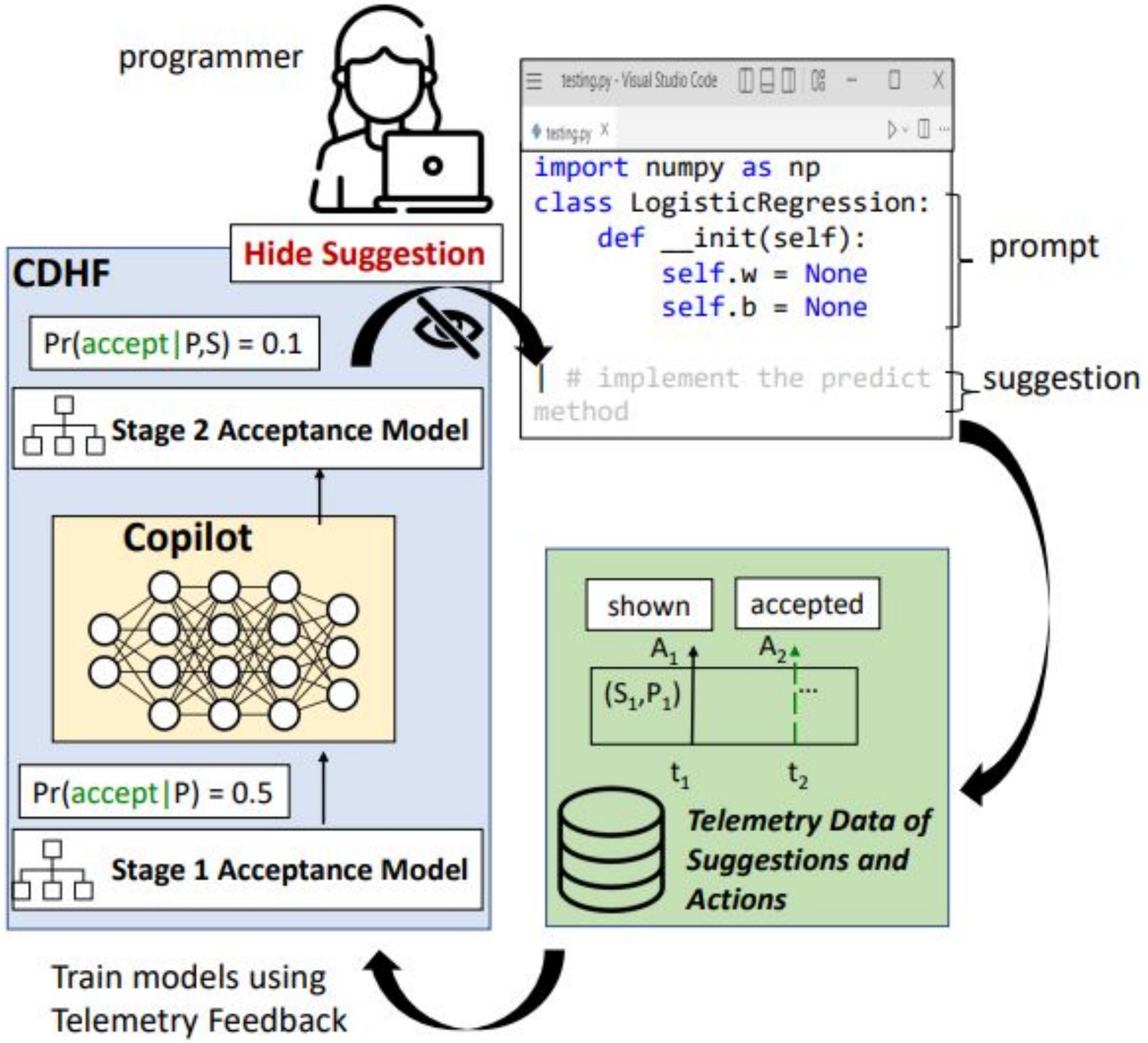
Human-AI – Too simplified metrics?

E.g. the number of lines of added code – the nature of interaction with Copilot (tab to accept suggestions) is a big factor!

Clickstream can help with design!

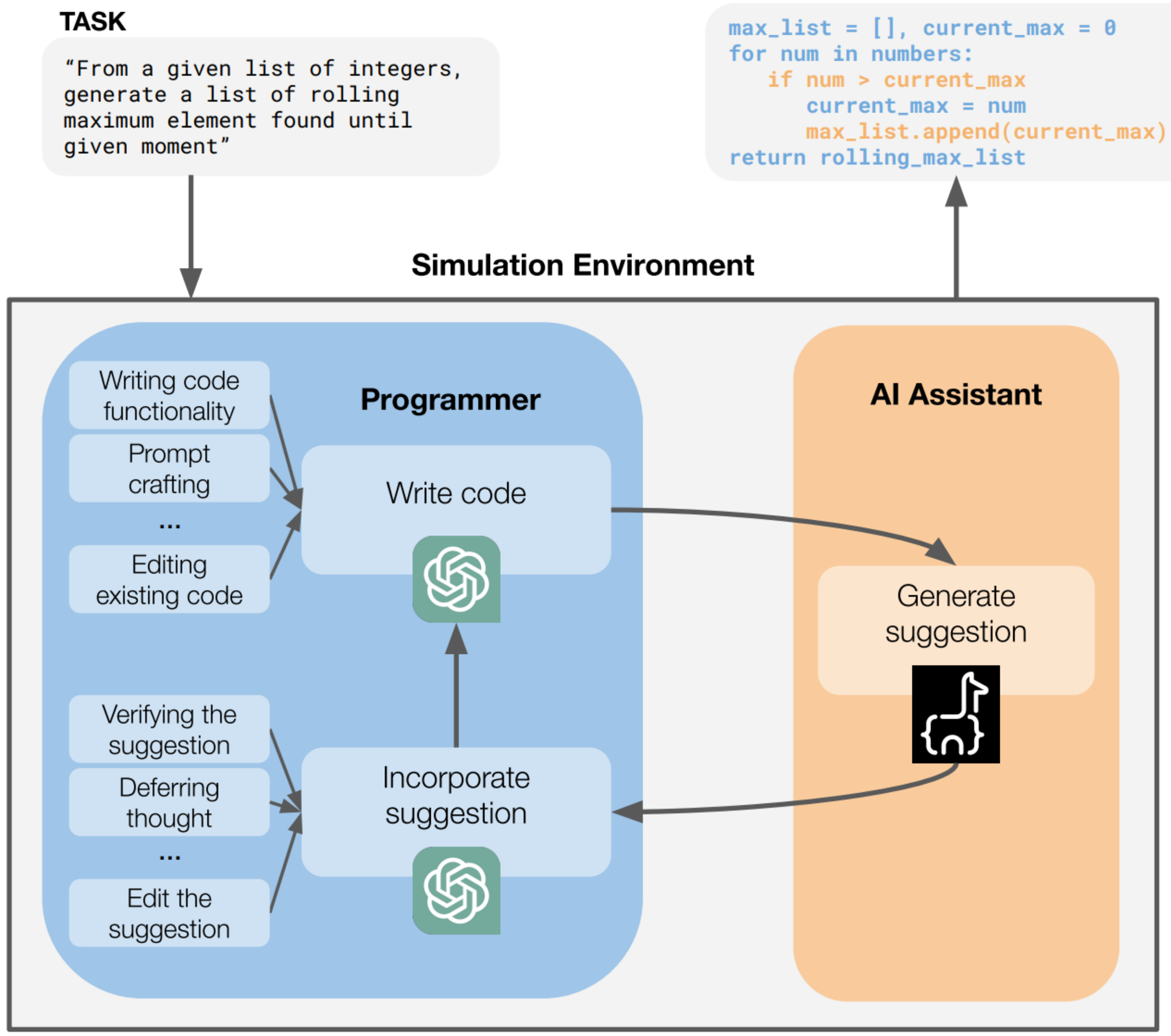
"We propose a utility theory framework, which models [when AI should make intervention to] programmers and decides which suggestions to display."

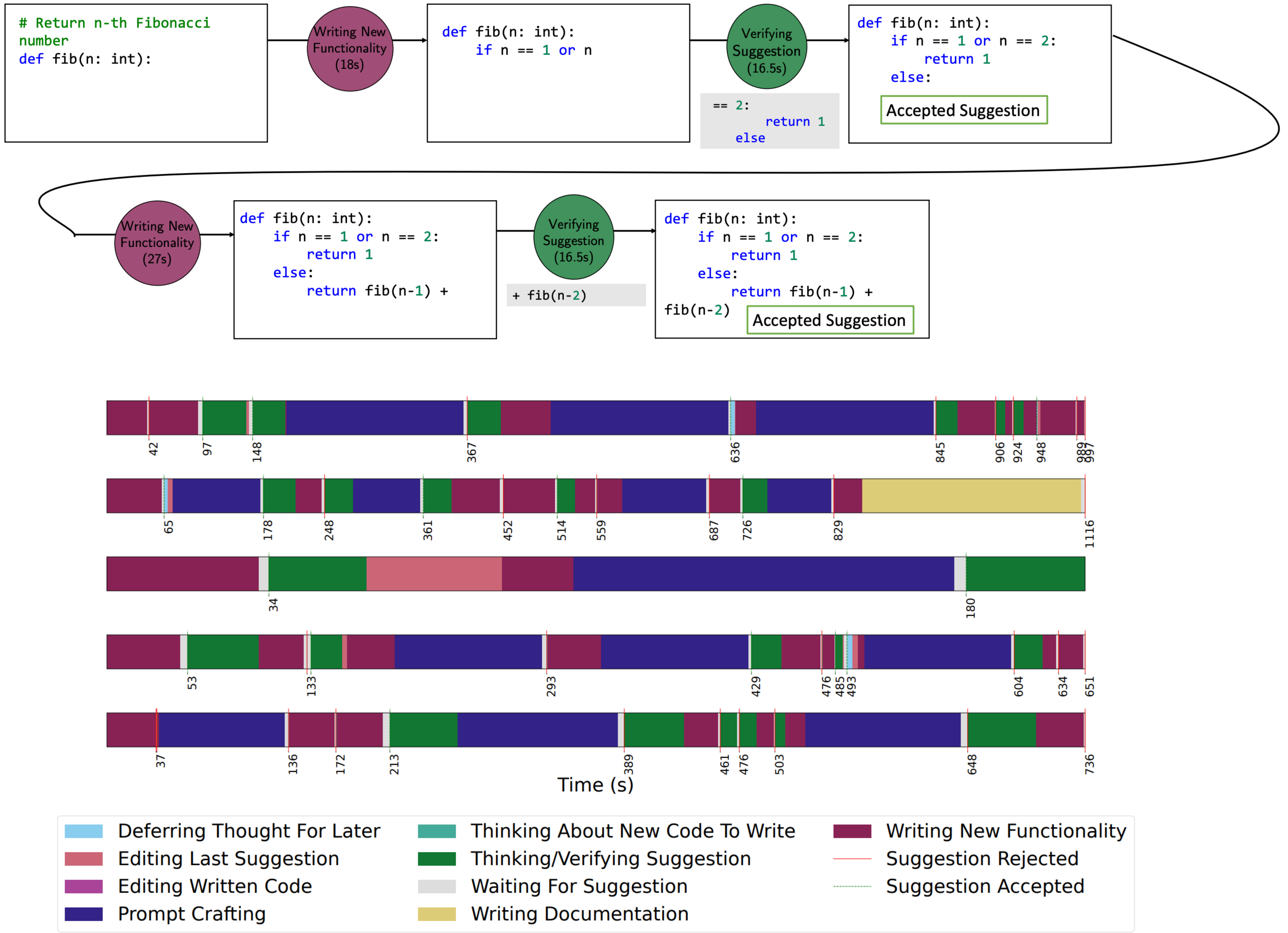
…models that predict suggestion acceptance to selectively hide suggestions reducing both latency and programmer verification time.



Mozannar, Hussein, et al. "When to show a suggestion? integrating human feedback in ai-assisted programming." *AAAI 2024*

Once have some data, can simulate humans

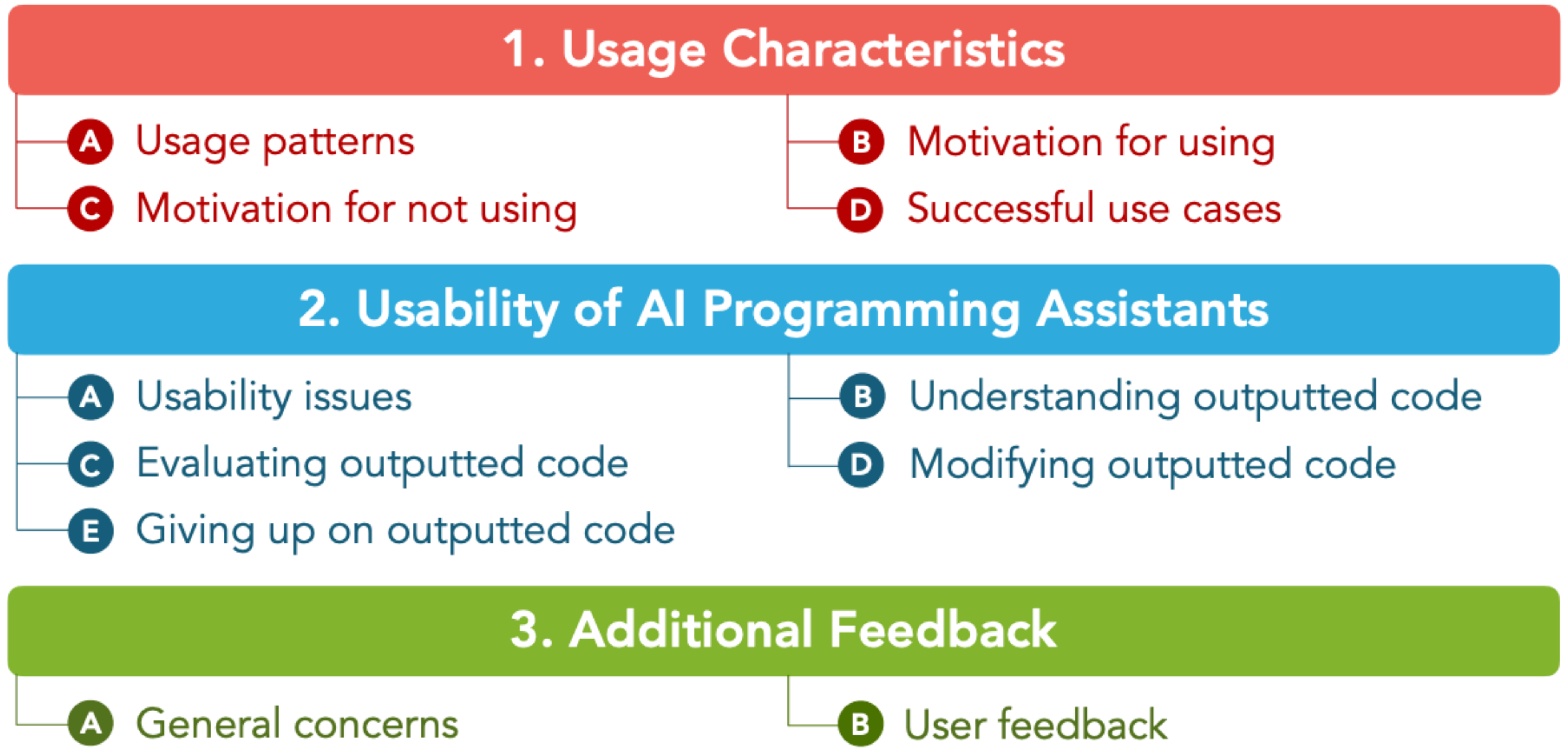




Mozannar, Hussein, et al. "Simulating Iterative Human-AI Interaction in Programming with LLMs." *NeurIPS 2023 Workshop*

Understand humans qualitatively: Surveys

"To understand developers' practices while using these tools and the important usability challenges they face, we administered a survey to a large population of developers and received responses from a diverse set of 410 developers."



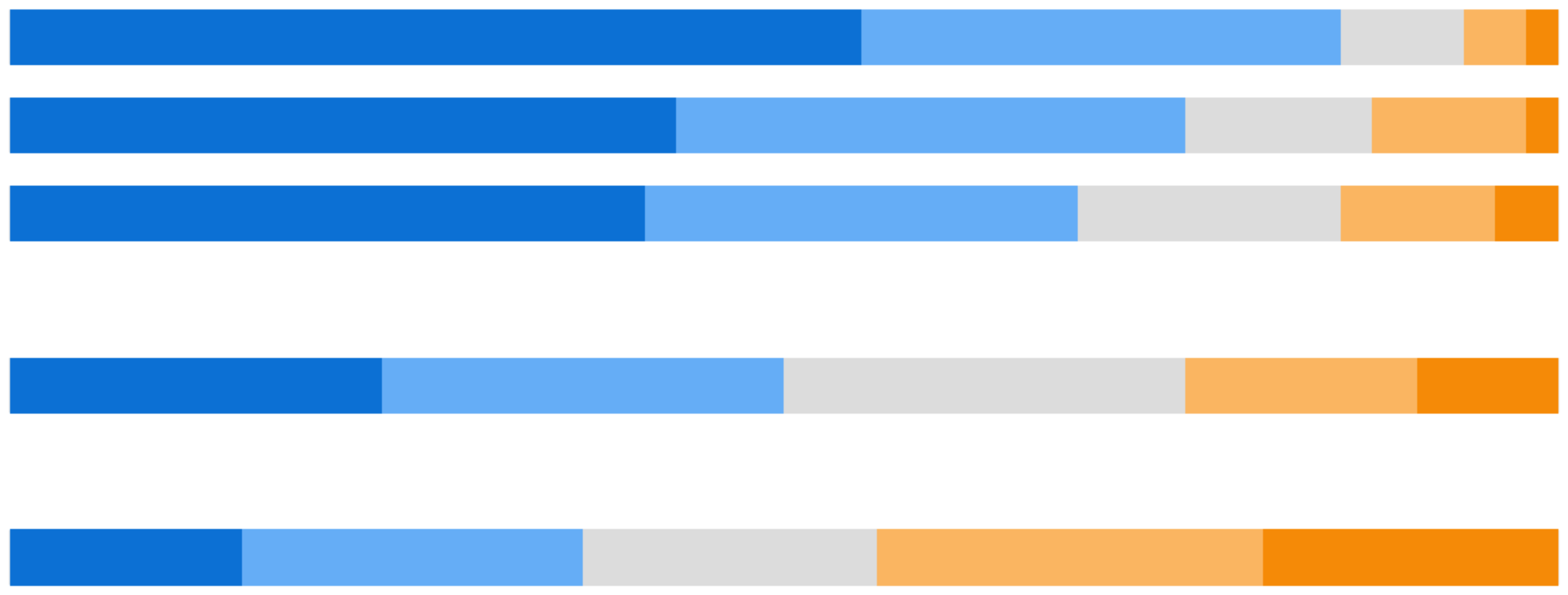
Survey Questions

- For this software project, estimate what percent of your code is written with the help of the following code generation tools.
- For each of the following reasons why you use code generation tools in this software project, rank its importance.
- For each of the following reasons why you do not use code generation tools, rank its importance.
- For your software project, estimate how often you experience the following scenarios when using code generation tools.
- For your software project, estimate how often the following reasons are why you find yourself giving up on code created by code generation tools.
- ★ What types of feedback would you like to give to code generation tools to make its suggestions better? Why?

Liang, Jenny T., Chenyang Yang, and Brad A. Myers. "A large-scale survey on the usability of ai programming assistants: Successes and challenges." *ICSE 2024*

Understand humans qualitatively: Surveys

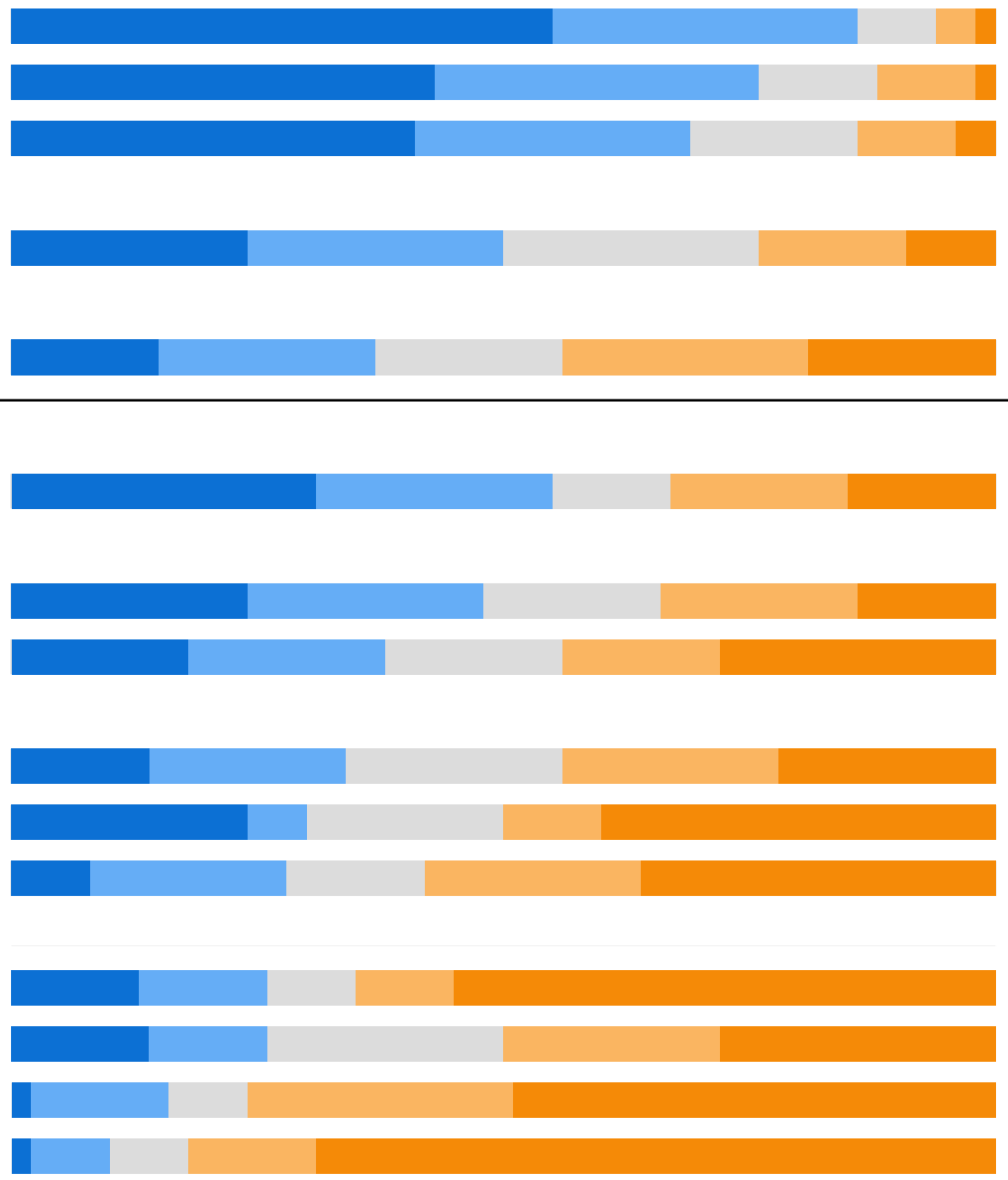
	Motivation	Distribution	
	A. For using		
M1	To have an autocomplete or reduce the amount of keystrokes I make.	*86%*	*6.2%*
M2	To finish my programming tasks faster.	*76%*	*12%*
M3	To skip needing to go online to find specific code snippets, programming syntax, or API calls I'm aware of, but can't remember.	*68%*	*14%*
M4	To discover potential ways or starting points to write a solution to a problem I'm facing.	*50%*	*24%*
M5	To find an edge case for my code I haven't considered.	*36%*	*44%*



Repetitive code (boilerplate code, repetitive endpoints for crud, etc.)
Code with simple logic
Autocomplete ("acceleration")
Quality assurance (e.g. log messages, test cases)
Proof-of-concepts (generate multiple implementations for a given problem)
Learning (of new libraries or programming languages)
Recalling (Find syntax they were familiar with but couldn't recall)
Efficiency
Documentation
Code consistency (e.g., indentation, quickly referencing sources created within the project)

Why humans use or not use programming tools

	Motivation	Distribution	
	A. For using		
M1	To have an autocomplete or reduce the amount of keystrokes I make.	*86%*	*6.2%*
M2	To finish my programming tasks faster.	*76%*	*12%*
M3	To skip needing to go online to find specific code snippets, programming syntax, or API calls I'm aware of, but can't remember.	*68%*	*14%*
M4	To discover potential ways or starting points to write a solution to a problem I'm facing.	*50%*	*24%*
M5	To find an edge case for my code I haven't considered.	*36%*	*44%*
	B. For not using		
M6	Code generation tools write code that doesn't meet functional or non-functional (e.g., security, performance) requirements that I need.	*54%*	*34%*
M7	It's hard to control code generation tools to get code that I want.	*48%*	*36%*
M8	I spend too much time debugging or modifying code written by code generation tools.	*38%*	*45%*
M9	I don't think code generation tools provide helpful suggestions.	*34%*	*46%*
M10	I don't want to use a tool that has access to my code.	*30%*	*51%*
M11	I write and use proprietary code that code generation tools haven't seen before and don't generate.	*28%*	*59%*
M12	To prevent potential intellectual property infringement.	*26%*	*66%*
M13	I find the tool's suggestions too distracting.	*26%*	*51%*
M14	I don't understand the code written by code generation tools.	*16%*	*76%*
M15	I don't want to use open-source code.	*10%*	*89%*



Very important | Important | Moderately important | Slightly important | Not important at all

Bonus: Taxonomy of software requirements

Functional requirements characterize units of functionality that we may want to group into coarser-grained functionalities that the software should support.

Non-functional requirements

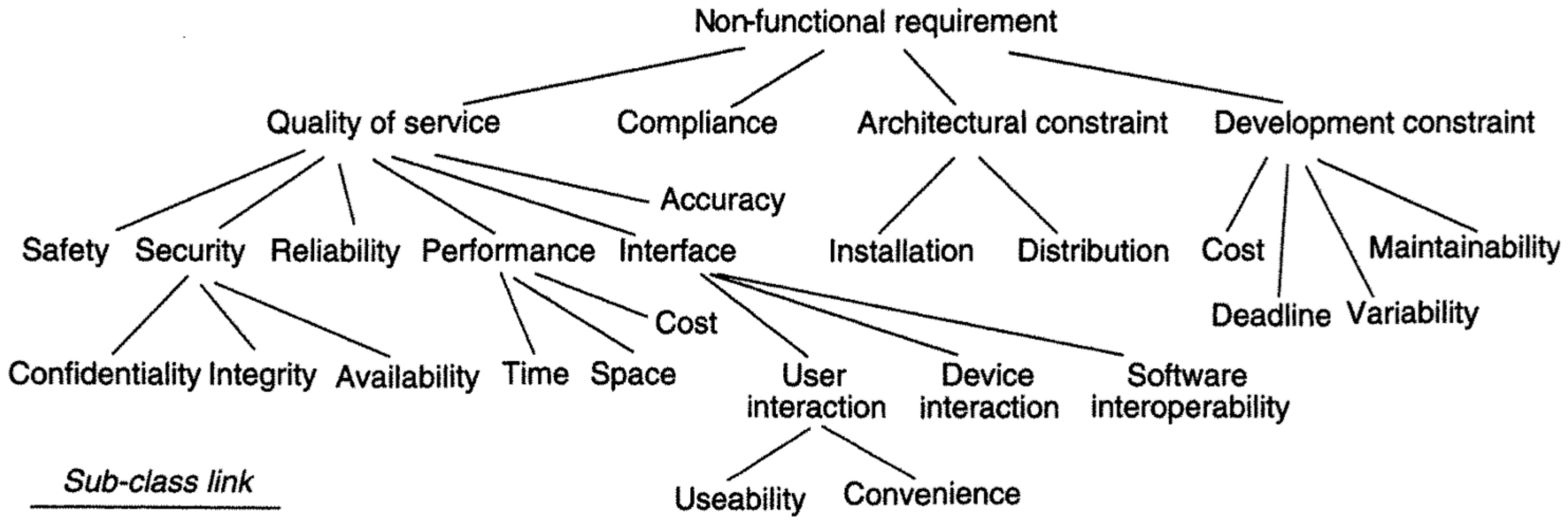


Figure 1.5 A taxonomy of non-functional requirements

More here: https://www.cs.cmu.edu/~ckaestne/17313/2018/20180913-requirements-solicitation-and-doc.pdf

Some other findings you might find interesting…

Engineers' prompting / input strategies (ranked by frequency)

Clear explanations, through "doctoring comments" and test cases
No strategy
Adding code
Follow conventions (e.g., well-named variables)
Break down instructions
Existing code context ("use it at advanced stages of project, for it to give better suggestions based on my project's history")
Prompt engineering

User-envisioned additional functionalities

Better understanding of code context. e.g., Code from the same workspace; Don't use deprecated API
Tool configuration: Have configurable parameters for suggestion frequency, distinguish[ing when to do] long code generation and short code [generation]
Natural language interactions
Code analysis, add annotation for functional and syntactic correctness
Explanations, e.g., link directly to documentation
More suggestions
Account for non-functional requirements

How do you know a code gen model is **useful?**

More metrics from more traditional human-human studies
Analyze actual human interactions
Hear what they have to say

How do you **make** a code gen model useful?

Again first some human-human reference…

Moderators	Human-Human vs. Human Solo	Human-AI (Copilot)
Task Types & Complexity	Complex task improve quality, simple one does not [7]; debugging is perceived as less enjoyable or effective than comprehension or refactoring [22]	N/A
Compatibility (E.g., Expertise)	Random pairing led to incompatible partners and conflicts during work [18]. Expertise: improve quality more effectively if pair is similarly skilled [14]; less-skilled students learn more and enjoy more [20, 22]; if knowledge gap is large, less-skilled programmers may tend to be more passive and disengaged [23]	N/A
Communica-tion	Conversations with intermediate-level details contribute to pair programming success [24]; different types of discourse lead to more attempts or more debug success [25]	N/A
Collaboration	Over-reliance leads to conflicts and impedes satisfaction and learning, as work is entirely burdened on one partner [4, 18]; educators recommend regular role-switching to ensure equitable learning in collaboration [2]	N/A
Logistics	Scheduling difficulties [26], teaching & evaluating individual responsibility and accountability are important to collaboration success [27], but can lead to increased management costs [21, 28]	N/A

Qianou Ma, Tongshuang Wu, and Kenneth Koedinger. "Is AI the better programming partner? Human-Human Pair Programming vs. Human-AI pAIr Programming." *AIED 2023 workshop*

Human-Human challenges to opportunities

Moderating Factors	Human-Human Challenges	Human-AI Opportunities
Task Types & Complexity: pair work better if the task is not too simple and good for collaboration [7, 22]	Hard to design suitable tasks of appropriate complexity level	AI may be used to generate collaboration tasks and adjust tasks complexity
Compatibility: pairs with similar skill levels and compatible working styles work better [14, 22]	Hard to find a similarly skilled or compatible partner	AI partner should adjust to human skill level and adapt to be compatible with different people
Communication: pairs work better with productive conversations [24], and critiques lead to more debugging success [25]	Hard to teach effective communication and constructive criticism	AI partner should support productive conversations and provide critiques
Collaboration: pairs work better with positive interdependence [27] and clear and balanced responsibilities [18]	Hard to teach collaboration and prevent free riders	AI should support positive social interactions and collaboration and avoid over-assist that eliminates human's need to engage
Logistics: pair programming is costly to implement because of management challenges [21, 28]	Hard to schedule and assess individual contributions in a pair	Scheduling is no longer a problem, but humans should be accountable and responsible when using AI-generated code

How do you make a code gen model useful?

A: Design and iterate on the interface

Communication via UI: Inline explanations

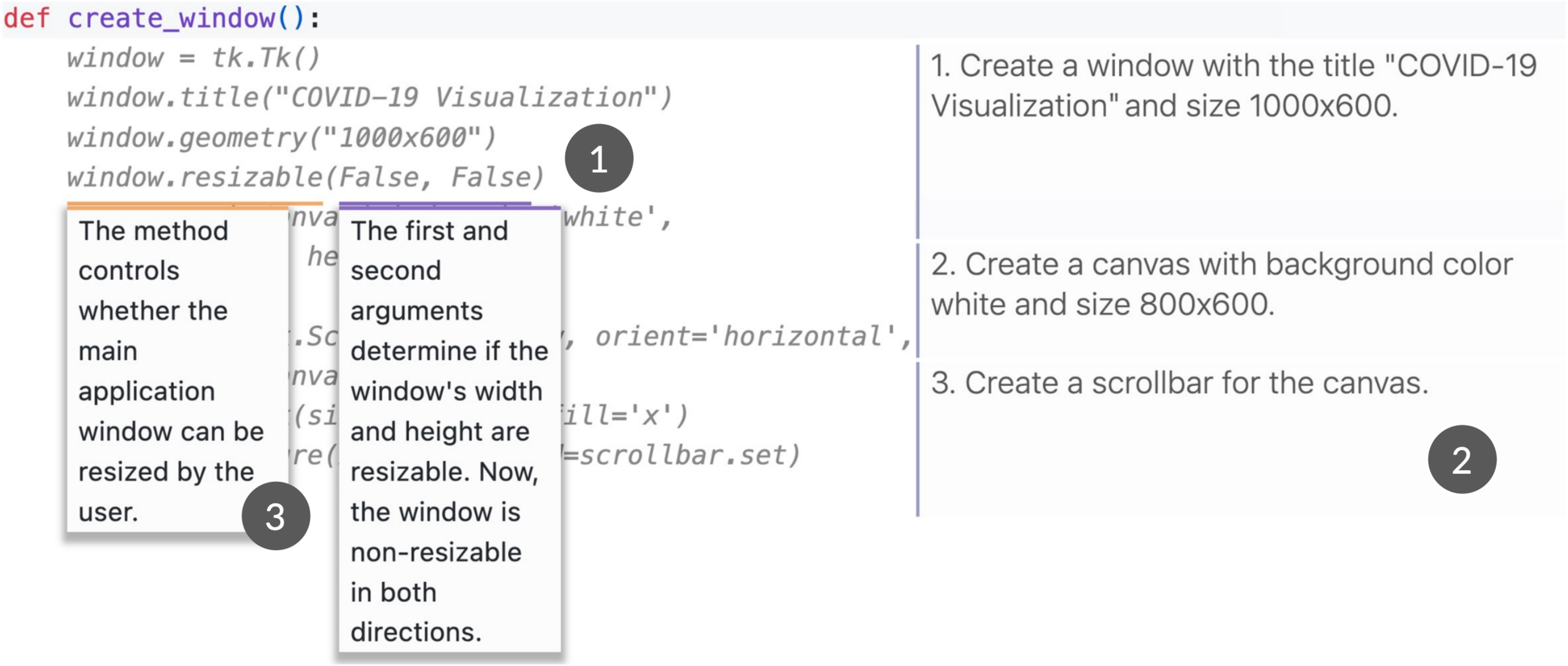


Figure 1: Ivie augments the interactive programming assistant with instant explanations that help programmers examine generated code. When a programming assistant suggests code (*italic* text above, ❶), Ivie annotates it with brief, informative explanations. Explanations appear at the level of blocks of code (in the right margin, ❷) and expressions (anchored beneath the line the programmer hovers over, ❸). For single-line suggestions, expression explanations appear automatically. Ivie's explanations help programmers break up complex or unfamiliar suggestions into pieces that can be more readily understood.

Yan, Litao, et al. "Ivie: Lightweight Anchored Explanations of Just-Generated Code." *CHI 2024*

Communication via UI: Inline explanations

```
df_all.merge(df_Apr, on='City', how='left', suffixes=('_all', '_apr'))
```

Combines df_all with another DataFrame df_Apr.

Only rows with matching 'City' values in both DataFrames will be merged.

Retains all rows from df_all, with matching rows from df_Apr. If no match is found,

In case of column name conflicts, df_all columns end with _all and df_Apr with

Figure 4: Explaining expressions. After a progra
consists of a single line, IVIE reveals explanations of
calls. The purpose of these explanations is to mak
programmer seeking to understand the precise beh

```
def visualize_data(df, max_temp_city, max_rain_city):
    fig, ax = plt.subplots(2, 1, figsize=(14, 10))

    df[df['City'] == max_temp_city]['Temperature'].plot(ax=ax[0])
    ax[0].set_title(f'Yearly Average Temperature for {max_temp_city}')
    ax[0].set_xlabel('Year')
    ax[0].set_ylabel('Temperature (°C)')
    ax[0].yaxis.set_major_formatter(ticker.FormatStrFormatter('%0.1f'))

    df[df['City'] == max_rain_city]['Rainfall'].plot(ax=ax[1], color='green')
    ax[1].set_title(f'Yearly Average Rainfall for {max_rain_city}')
    ax[1].set_xlabel('Year')
    ax[1].set_ylabel('Rainfall (mm)')
    ax[1].yaxis.set_major_locator(ticker.MaxNLocator(nbins=6, integer=True))

    plt.tight_layout()
    plt.show()
```

1. Creates a figure with two vertical subplots.

2. Plot the temperature data for the city with the highest temperature in the top subplot.

3. Plot the rainfall data for the city with the highest rainfall in the bottom subplot.

4. Format the figure and display it.

Figure 5: Explaining multi-line suggestions. When a programming assistant suggests multiple lines of code, IVIE splits up and explains that code. Its explanations appear in the right margin of the editor. The explanations are meant to help a programmer get a high-level understanding of the code. In the pictured scenario, these explanations might help the programmer understand that the two longest sections of the code suggestion configure each of two subplots, each with a different slice of the data.

Communication via UI: Inline explanations

Usually no need to be super sophisticated methods but just clean communication!

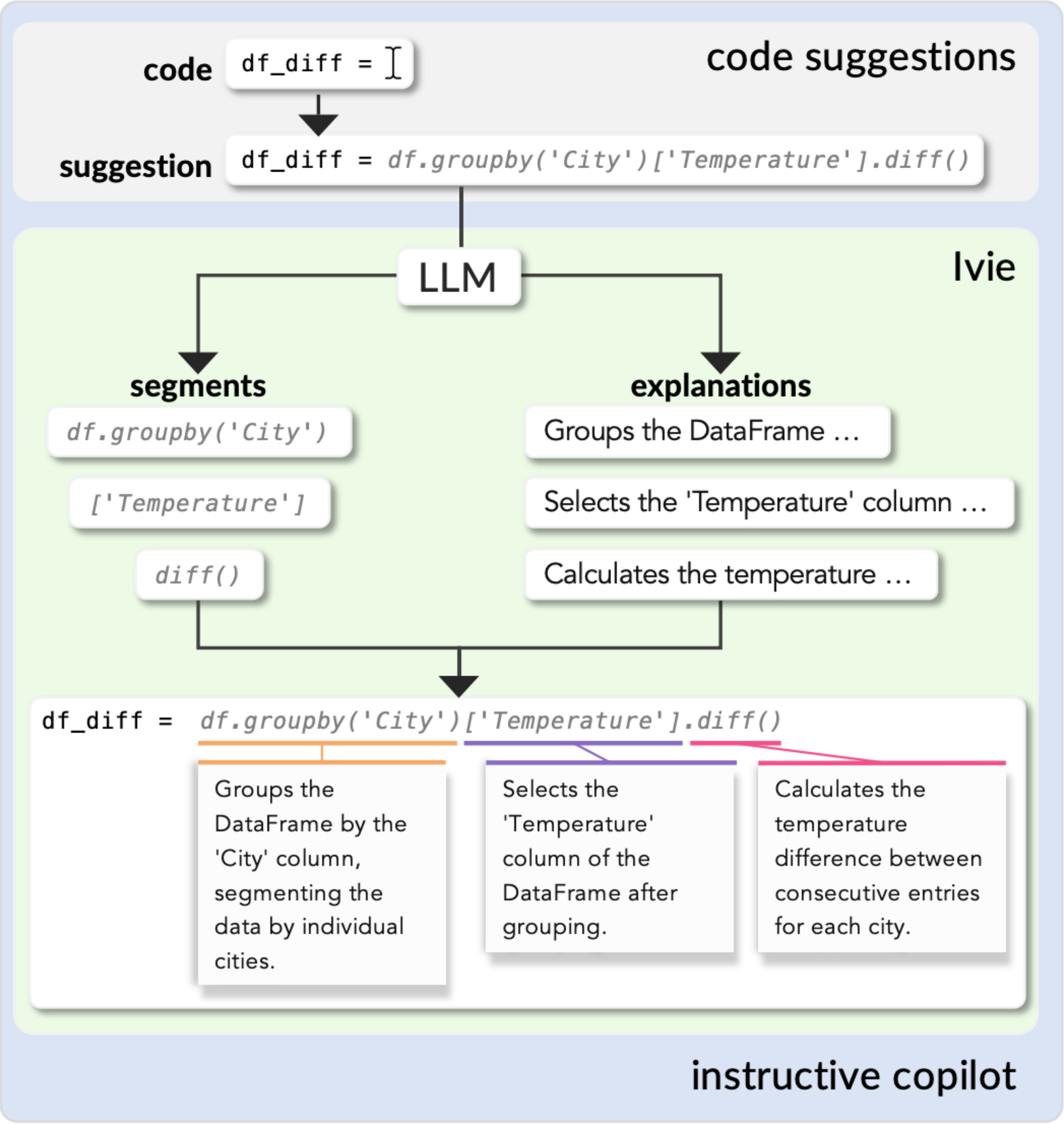


Figure 2: The implementation of an instructive copilot for programming. Ivie creates interactive overlays that explain suggestions made by a programming assistant. When the programming assistant (e.g., Copilot) displays the suggestion, Ivie submits that suggestion in a prompt to an LLM, requesting that the code be segmented and explained. Ivie then integrates the explanations into the editor as overlays beneath the expressions they explain.

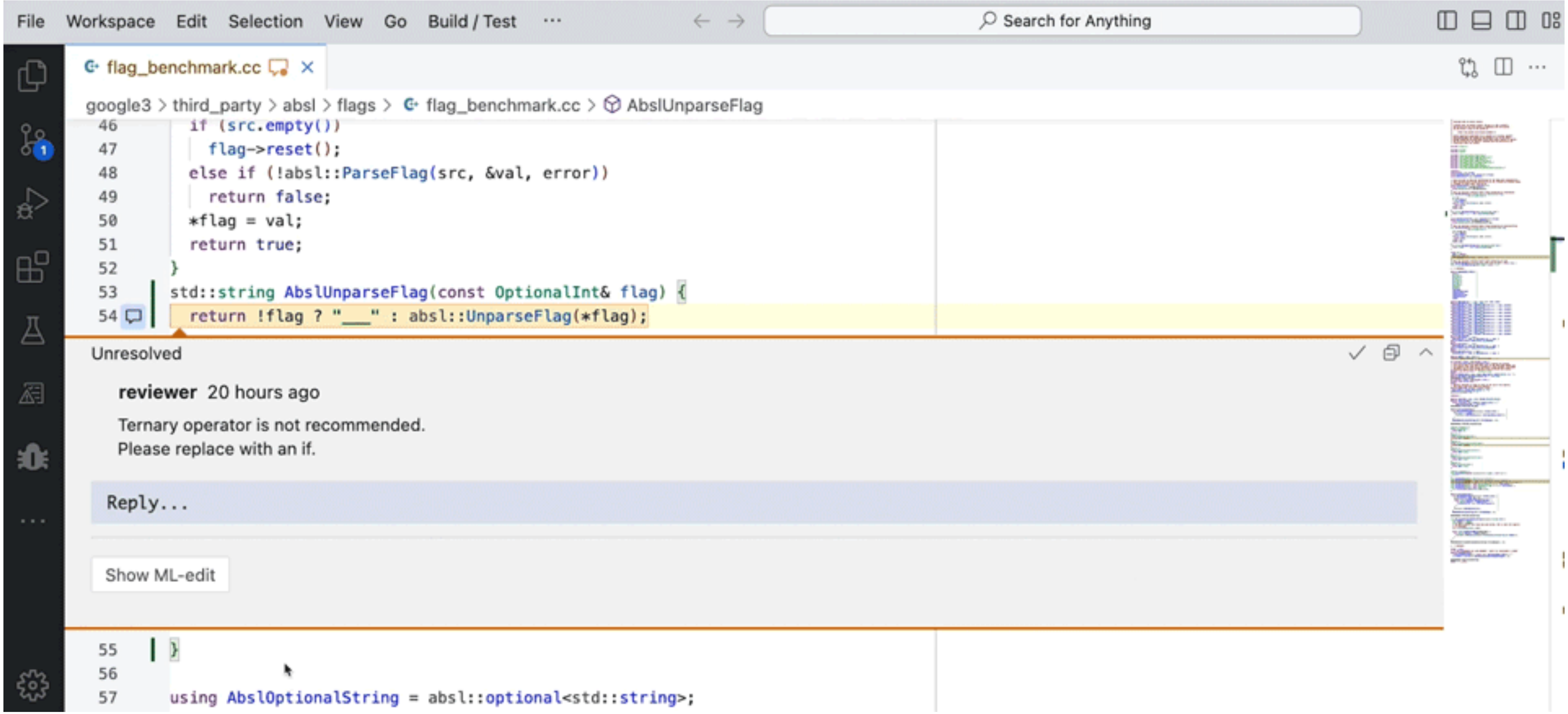
```
Please dissect the following line of code, and explain
the unfamiliar vocabulary and structures with less than
15 words each. Include ranges for parameter values and
describe how changes in these parameters will affect
the output.

Prompt:
fig, ax = plt.subplots(2, 1, figsize=(14, 10))
Output:
plt.subplots $#$ Create a figure and set of subplots.
2, 1 $#$ 2 rows, 1 column of subplots.
figsize=(14, 10) $#$ Width and height of entire figure.
fig $#$ The whole window/figure containing subplots.
ax $#$ Array of individual subplot axes.

Prompt:
df_all.merge(df_Apr, on='City', how='left',
suffixes=('_all', '_apr'))
Output:
```

Figure 3: A prompt for requesting expression-level explanations of generated code. This prompt requests explanations of suggested code. It provides a single example of how it would like code suggestions to be split into expressions with accompanying brief explanations of those expressions.

How do you make a code gen model useful?

A: Design and iterate for specific use cases (case studies!)

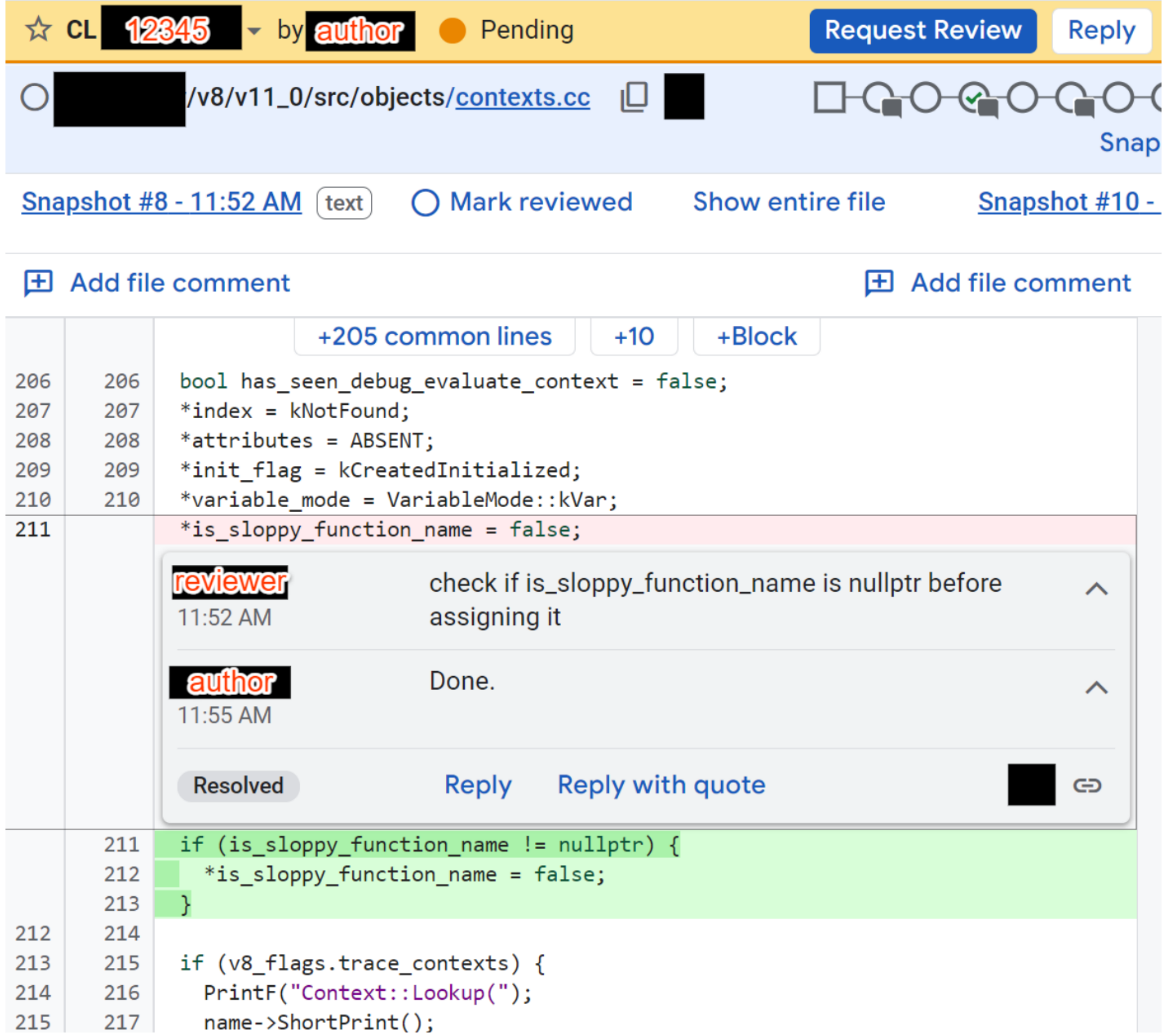
Design for Use Case: Addressing Code Review



Alexander Frömmgen, et al. "Resolving Code Review Comments with Machine Learning." *ICSE 2024*

Design for Use Case: Addressing Code Review

An example of a review comment in Critique. The reviewer asked for a defensive coding practice. The author addressed the comment by updating their changelist with a new review snapshot. The update is shown via colors: green for added text and red for removed. The author responded to the comment with “Done.” and marked it “Resolved”.



Design for Use Case: Addressing Code Review

"We started by training a model that predicts code edits needed to address reviewer comments. The model is pre-trained on various coding tasks and related developer activities (e.g., renaming a variable, repairing a broken build, editing a file). It's then fine-tuned for this specific task with reviewed code changes, the reviewer comments, and the edits the author performed to address those comments."

```
... +8 common lines ...
import datetime

... +67 common lines ...

def _temporal_split_from_datetime(
    example_datetime: datetime.datetime) -> Optional[str]:
  """Finds the split name using the example datetime."""
  end_train_datetime = datetime.datetime.strptime('2022-01-01', '%Y-%m-%d')
  end_valid_datetime = datetime.datetime.strptime('2022-05-01', '%Y-%m-%d')
  end_test_datetime = datetime.datetime.strptime('2022-06-01', '%Y-%m-%d')

  if example_datetime < end_train_datetime:
    return 'train'
  el if example_datetime < end_valid_datetime:
    return 'validation'
  el if example_datetime < end_test_datetime:
    return 'test'
  else:
    return None
```

Model quality: In-product measurements

Offline evaluation, by computing the recall@X metric described above over a held-out test dataset

Online evaluation / user feedback, by measuring the number of code-review comments produced during day-to-day business, the number of predictions the model made, the number of those predictions that were previewed, and of those how many were applied, or received thumbs up/thumbs down.

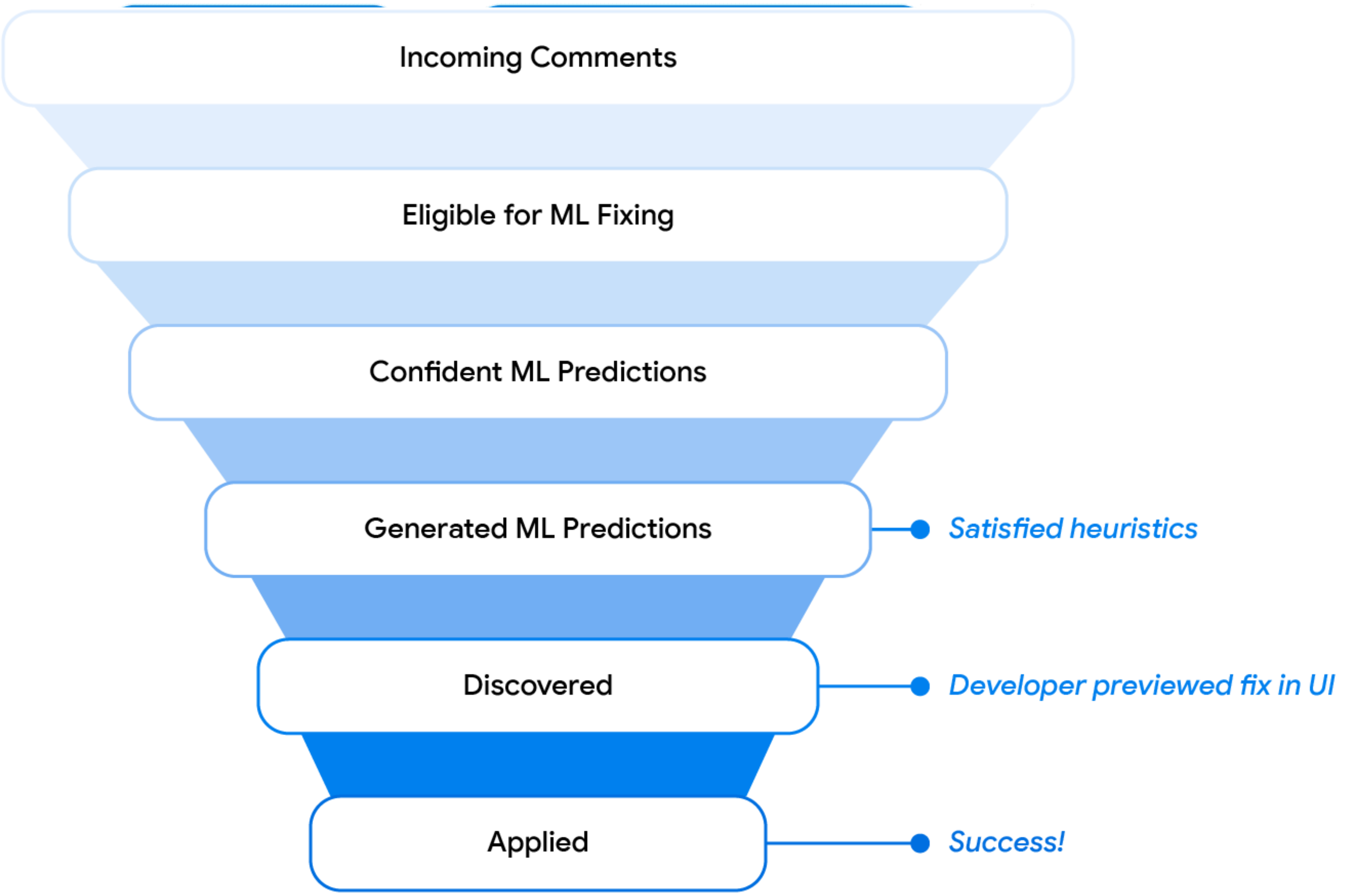
All types of such evaluation are meant to detect an increase in developer productivity, but act as easier-tomeasure proxies of that measure.

Acceptance rate: the fraction of comments resolved by an accepted ML suggestion

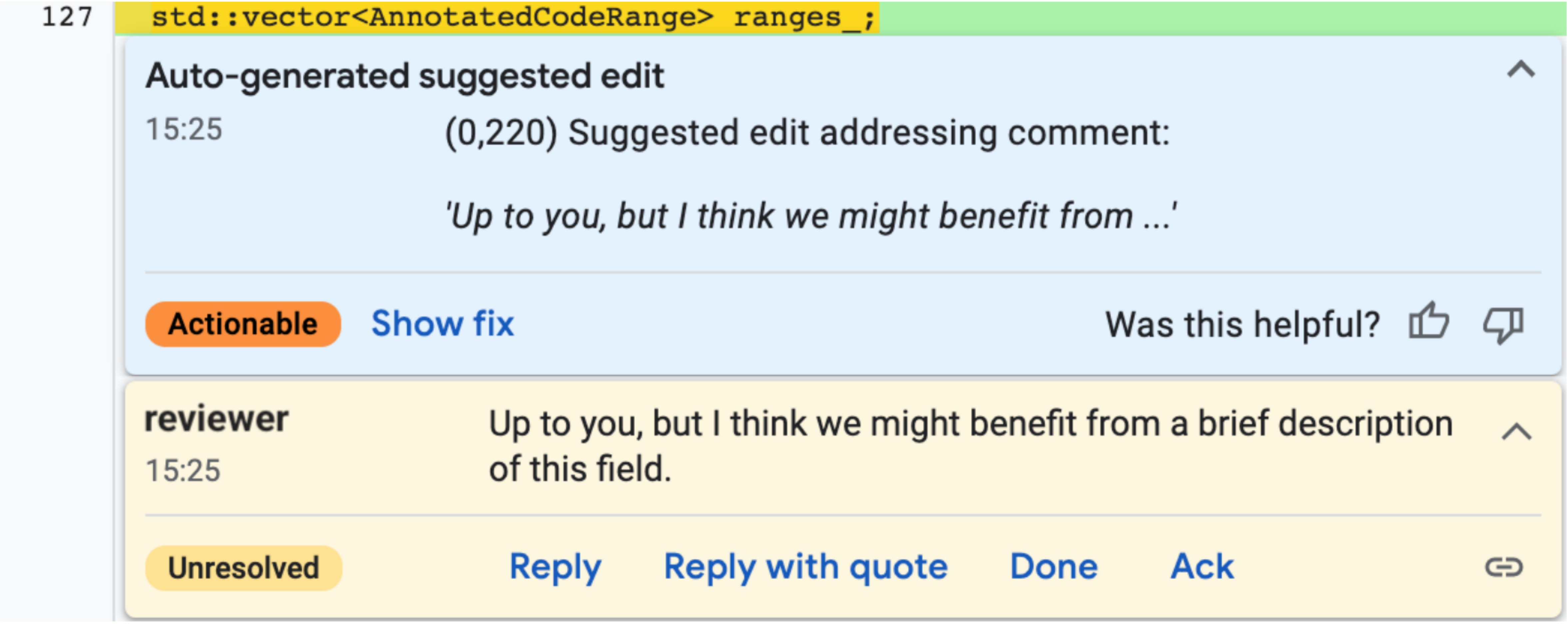
Discoverability: the fraction of surfaced suggestions that were previewed by system users.

Filters between model and usage

“For every new reviewer comment, we generate the model input in the same format that is used for training, query the model, and generate the suggested code edit. **If the model is confident in the prediction and a few additional heuristics are satisfied**, we send the suggested edit to downstream systems.”

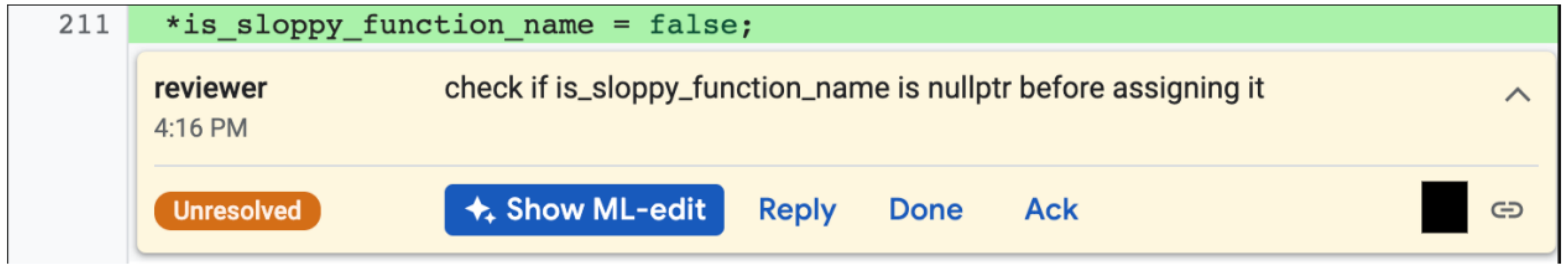


The impact of UIs



Original: A separate, asynchronous analyzer queried the model and produced the suggested edit as an independent code finding, in a *separate annotation*.

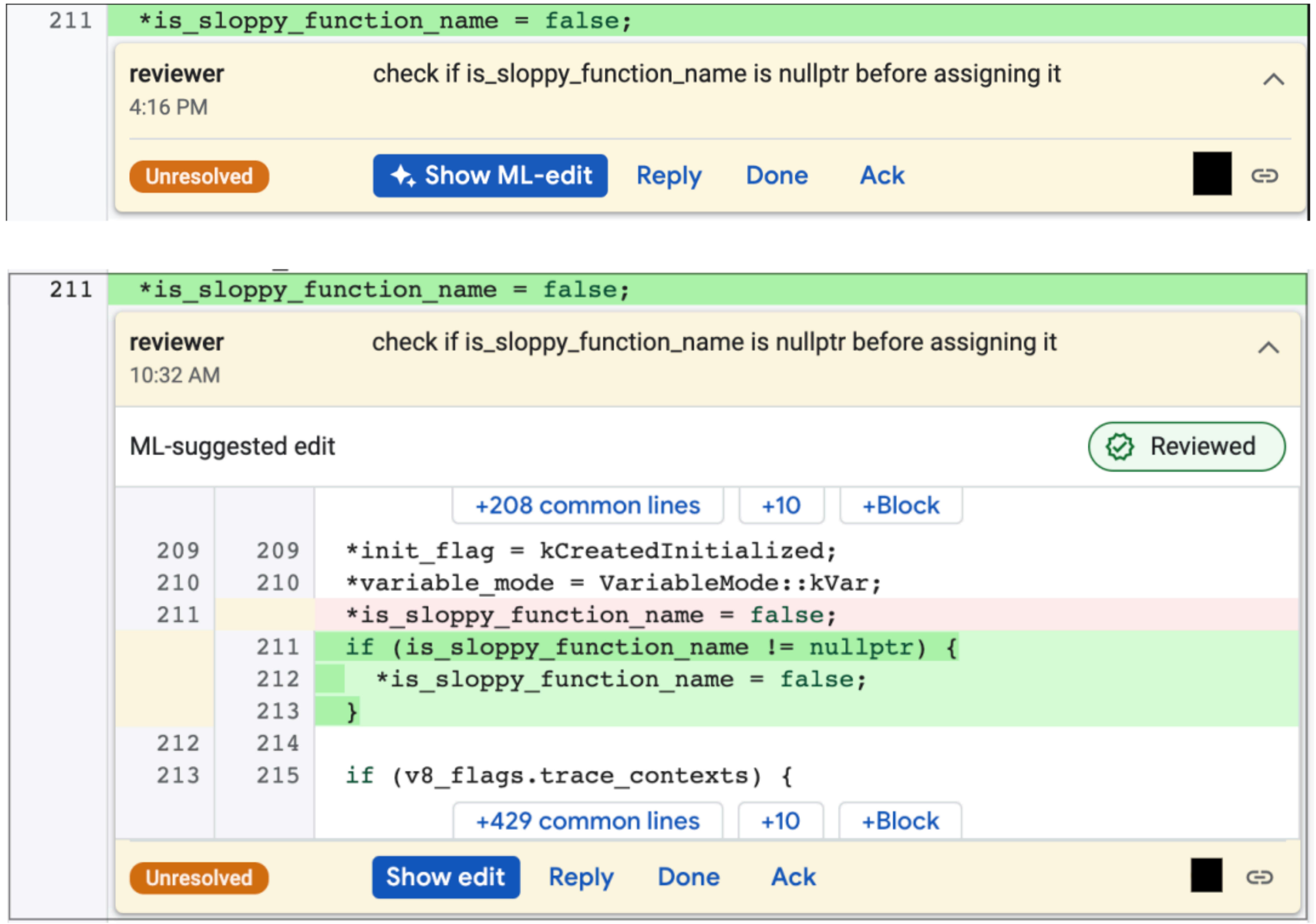
Pitfall: decoupled comment and suggested edit.
-> Duplication of information, wasted precious UI real-estate, and confused the prevailing visual language of review comments.



Revision: combine the two sources of information, by placing a “Show ML edit” in the same box where the reviewer comment appears

Result: improved discoverability considerably

The impact of UIs

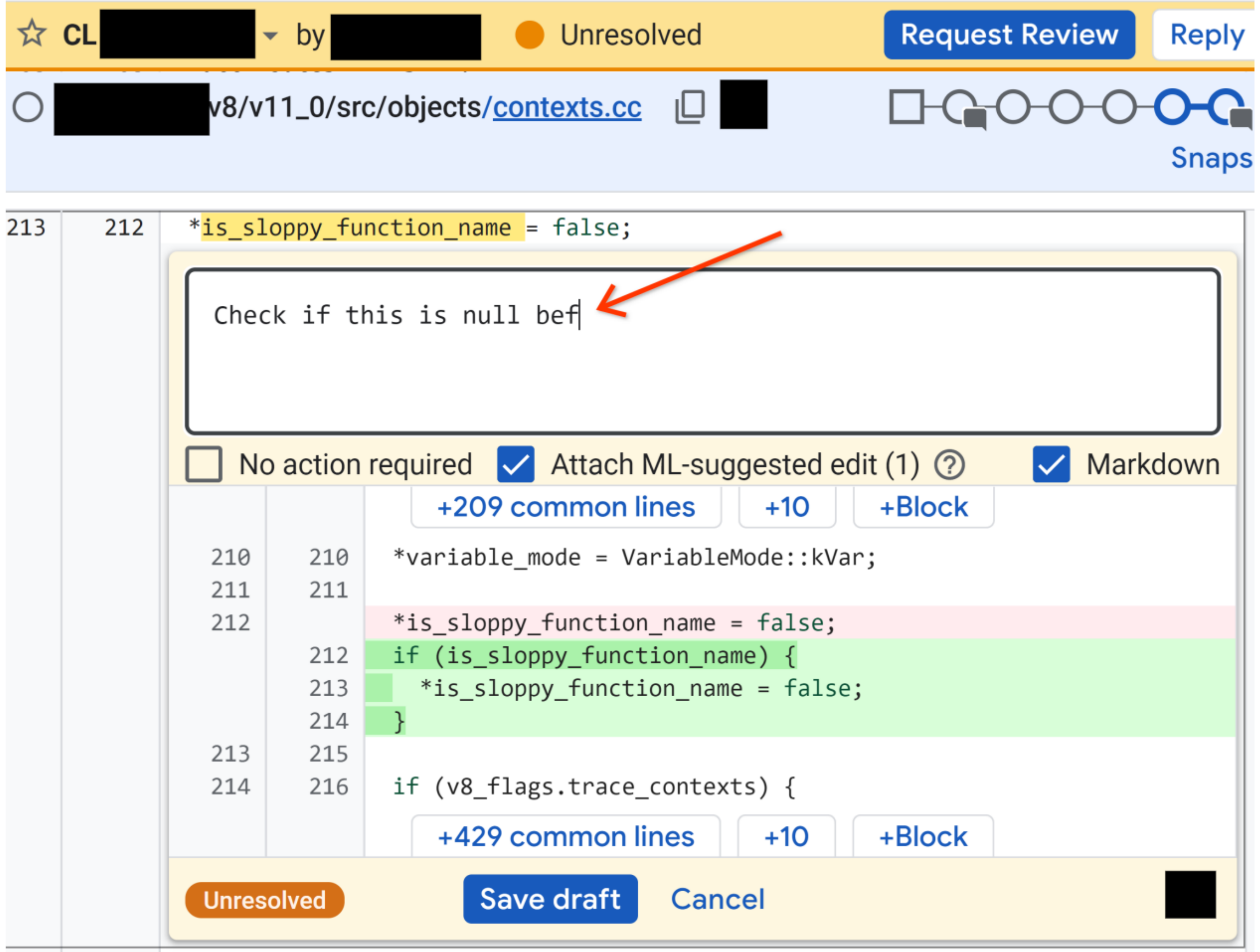


Pitfall: click to view. Since code shepherding (i.e., editing the changelist in light of the reviewer comments) takes a significant fraction of developers' time—one study at Google measured the median to be around 60 minutes [7]—efficiency in addressing comments is important

Revision: Showing the suggested edit immediately next to the reviewer comment, rather than requiring a click of the "Show ML edit" button.

Result: ML-suggested edit discoverability for the changelist author improved.

The impact of UIs



The location of the comment and the mention of "check" and "null" were sufficient to trigger the assistant to suggest the intended edit.

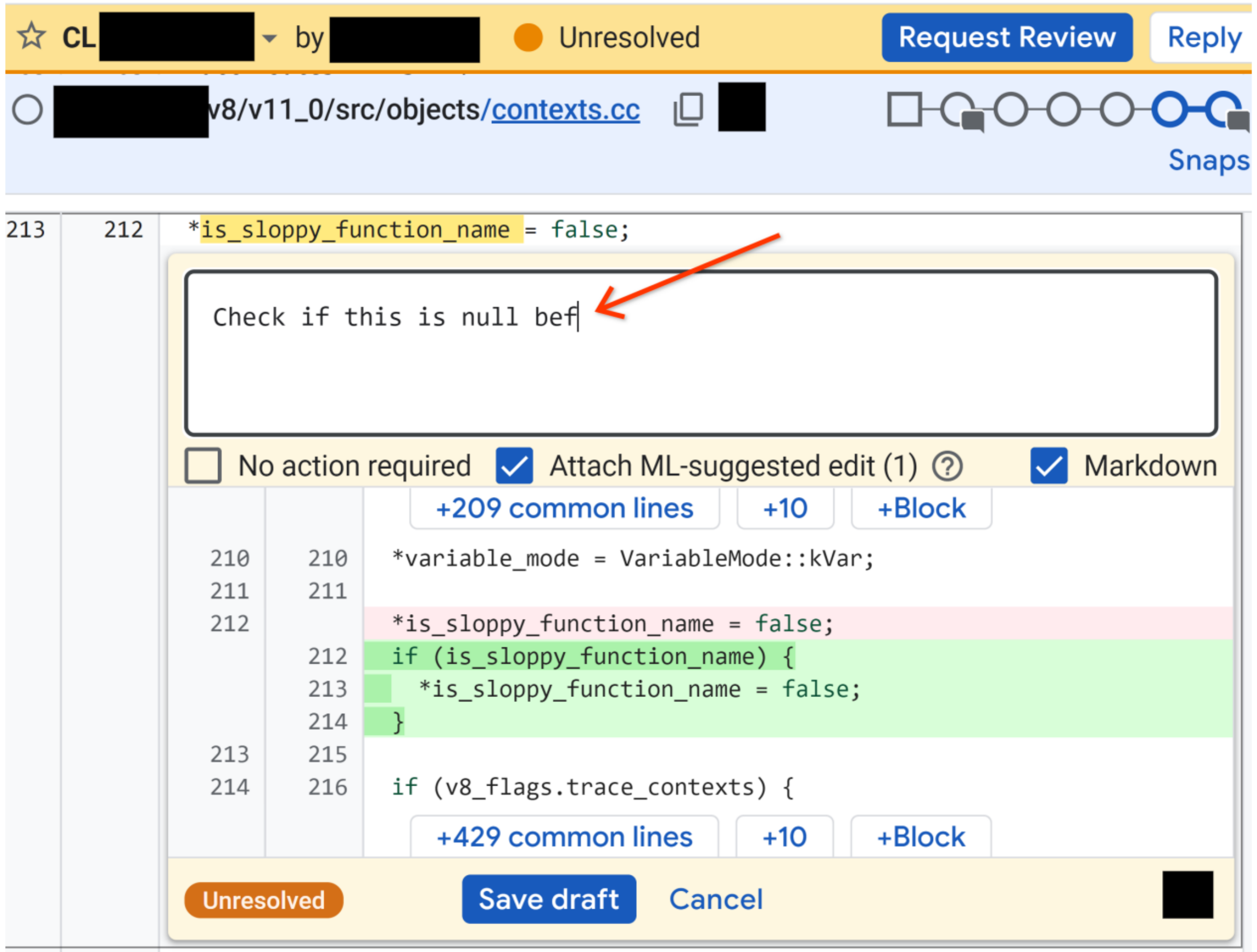
Original: Just show suggested revision to the code author but not the reviewer.

Pitfall: decoupled reviewer from ML assistant.
Reviewers who were uncomfortable having an ML model "interpret" their comment into a suggested edit, and would prefer to preview the suggestion before providing it to the code author. "the pedagogical function of code review – It is often how new engineers are trained on local conventions and programming discipline."

Revision: The reviewer is shown the ML-suggested edit as they type their comment. The reviewer can decide to reject the suggested edit (in which case the author will only see the reviewer's comment).

Result: Many incorrect suggestions are pre-filtered out; Can use a less lower auto-filter because human filter is in the loop!

The impact of UIs



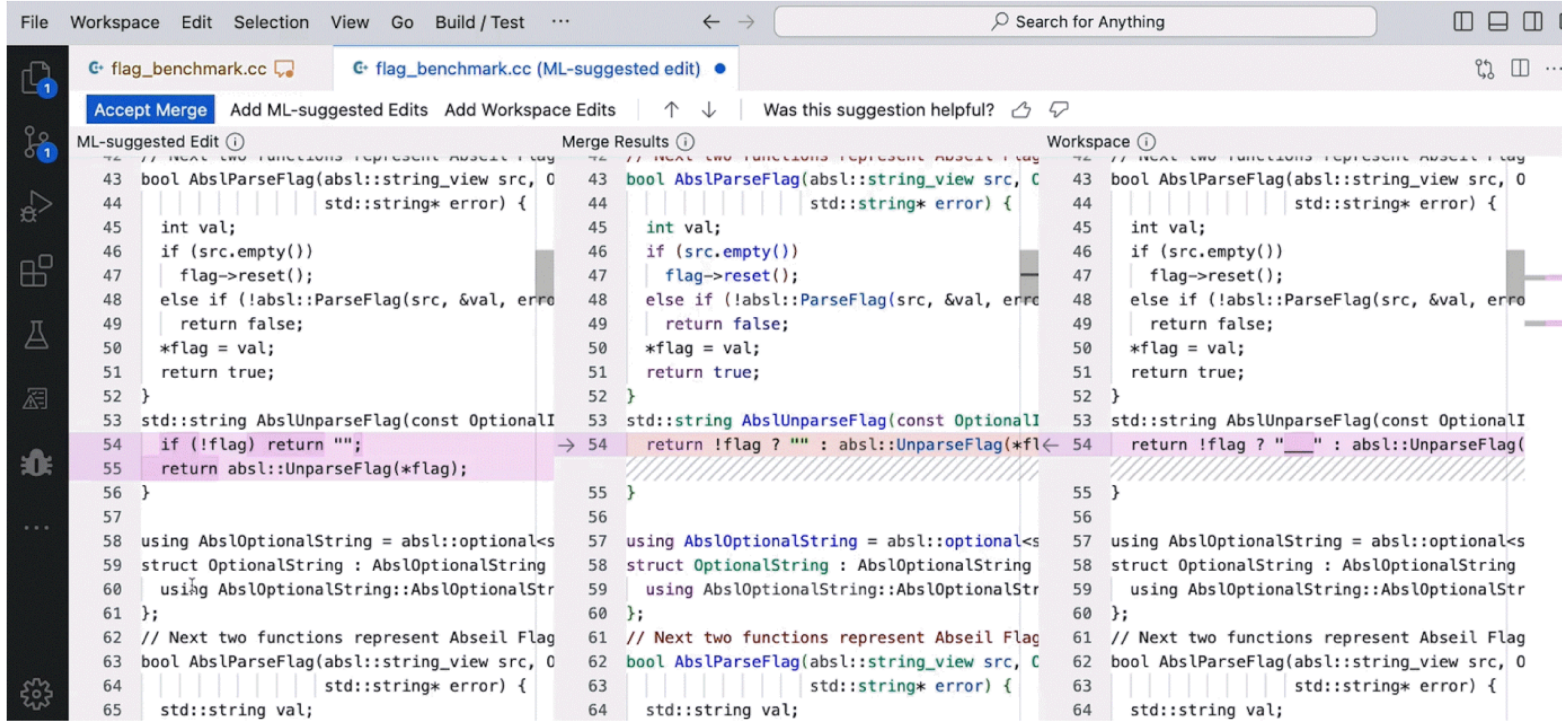
Original: Reviewers are typically pressed for time, and may move on quickly from comment to comment. In an attempt to reduce back-end prediction load, and to avoid showing reviewers suggested edits before they have typed enough of their comment, we set the triggering delay between when the reviewer starts typing a comment and when a prediction is requested to 1500ms.

Pitfall: slow-to-predict edits.; Between this triggering delay, and the additive prediction latency of the model, many predictions "missed" the reviewer, who had already moved on.

Revision: Further reduced the triggering delay to 500ms. and improved the prediction latency through considerable engineering effort.

Result: number of suggested edits previewed by reviewers increased by 12%, and the acceptance rate of ML-suggested edits by authors improved by 18%.

The impact of UIs



Original: Our original design of the UI assumed that the changelist author and reviewers operate in lock step: one stops when the other starts working on the changelist.

Pitfall: code review is serialized.; This is not how code review operates in practice. Sometimes the changelist is edited by the author as the reviewer is reviewing, or perhaps the reviewer thinks of a new comment after they have passed the bulk of their review to the author, and sometimes the reviewer attaches a comment to an older review snapshot of the changelist. All in all, this means that sometimes even an MLsuggested edit that the author wishes to accept is incompatible with the current state of the code.

Revision: detect those cases, and opening a three-way merge window (Figure 10) for the author to resolve any merge conflicts.

Result: the number of accepted suggested edits increased.

Some takeaways

When a model is in a specific use case it usually means **blending into existing workflows**

Test-in-product is not the most ideal but usually quite **useful**

There will be **metrics** not relevant to model, but just relevant to **usability** (e.g. discoverability)

Little things like latency in suggestion can easily change usability

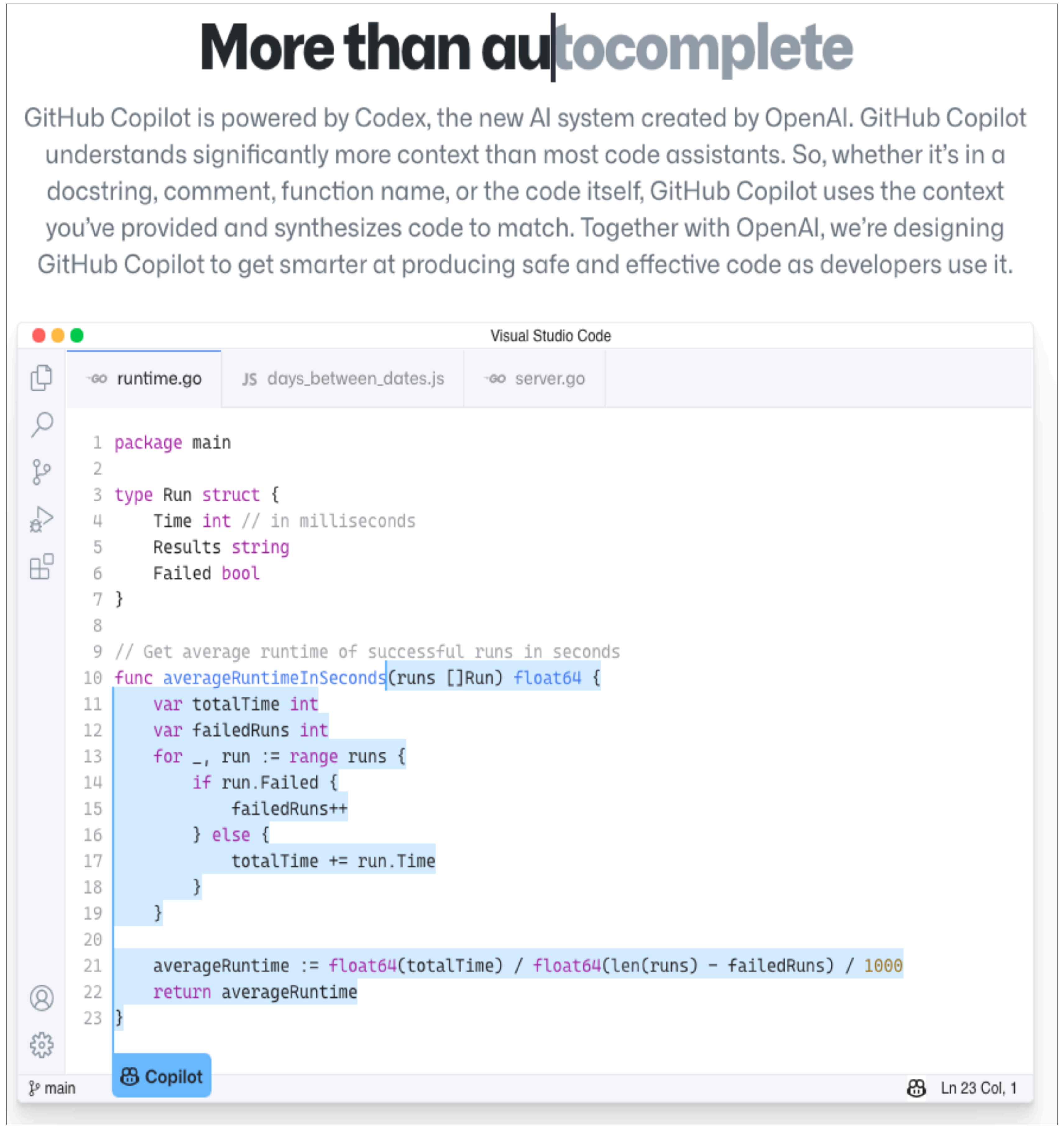
UI iteration is a BIG aspect

Need to consider **all users touching** the system (reviewers, and authors).

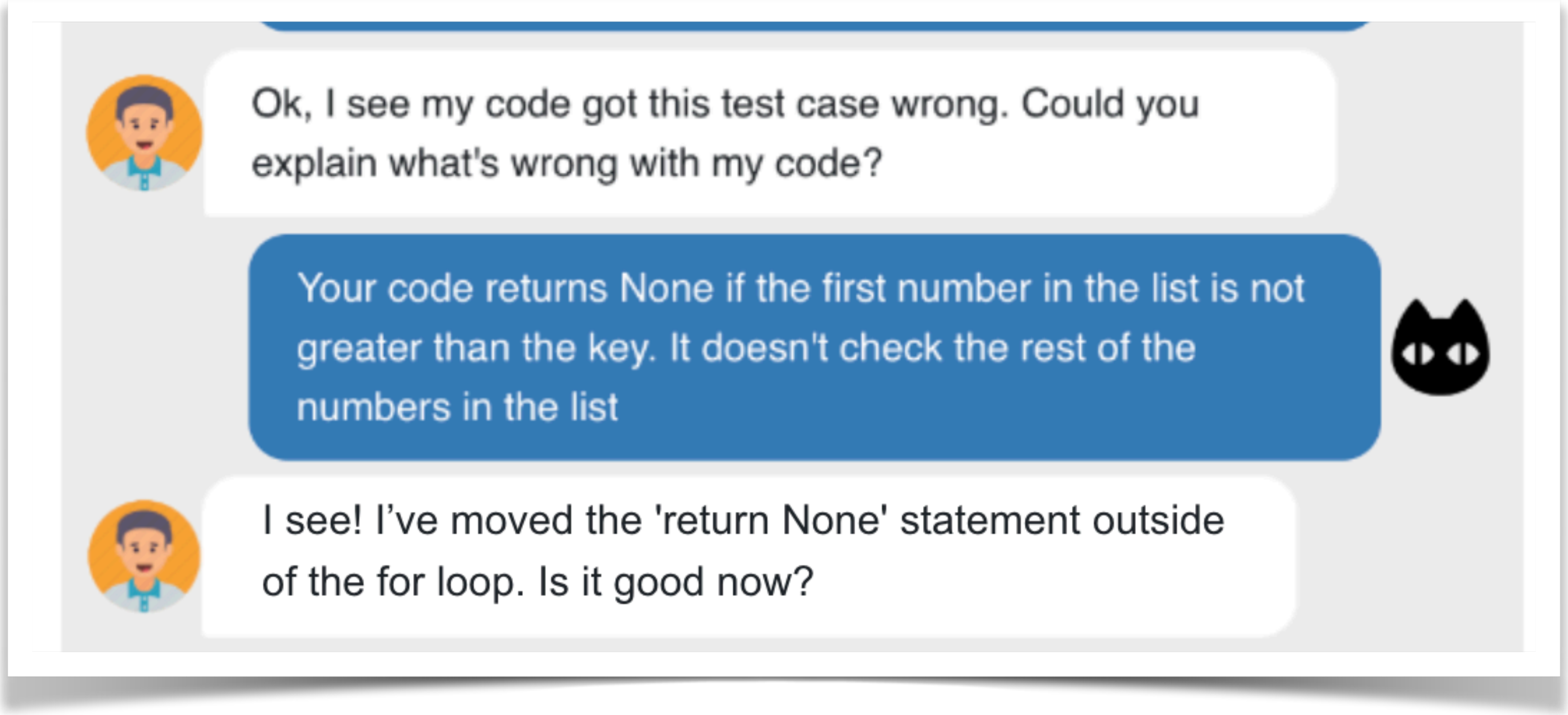
Also consider the **original objective** of the task (a bit of education and training going on!)

Case Study: LLM for CS Education

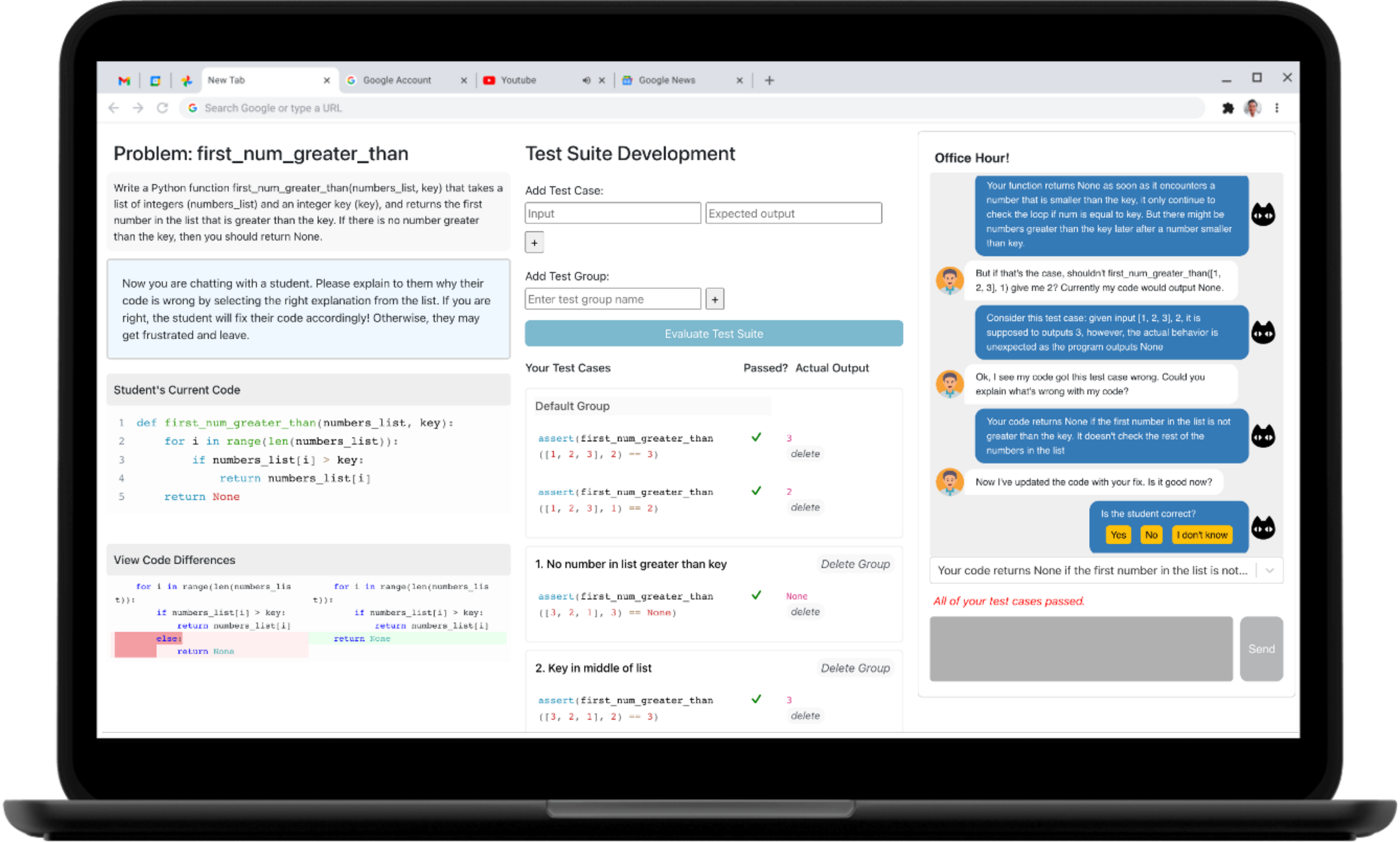
When AI writes code

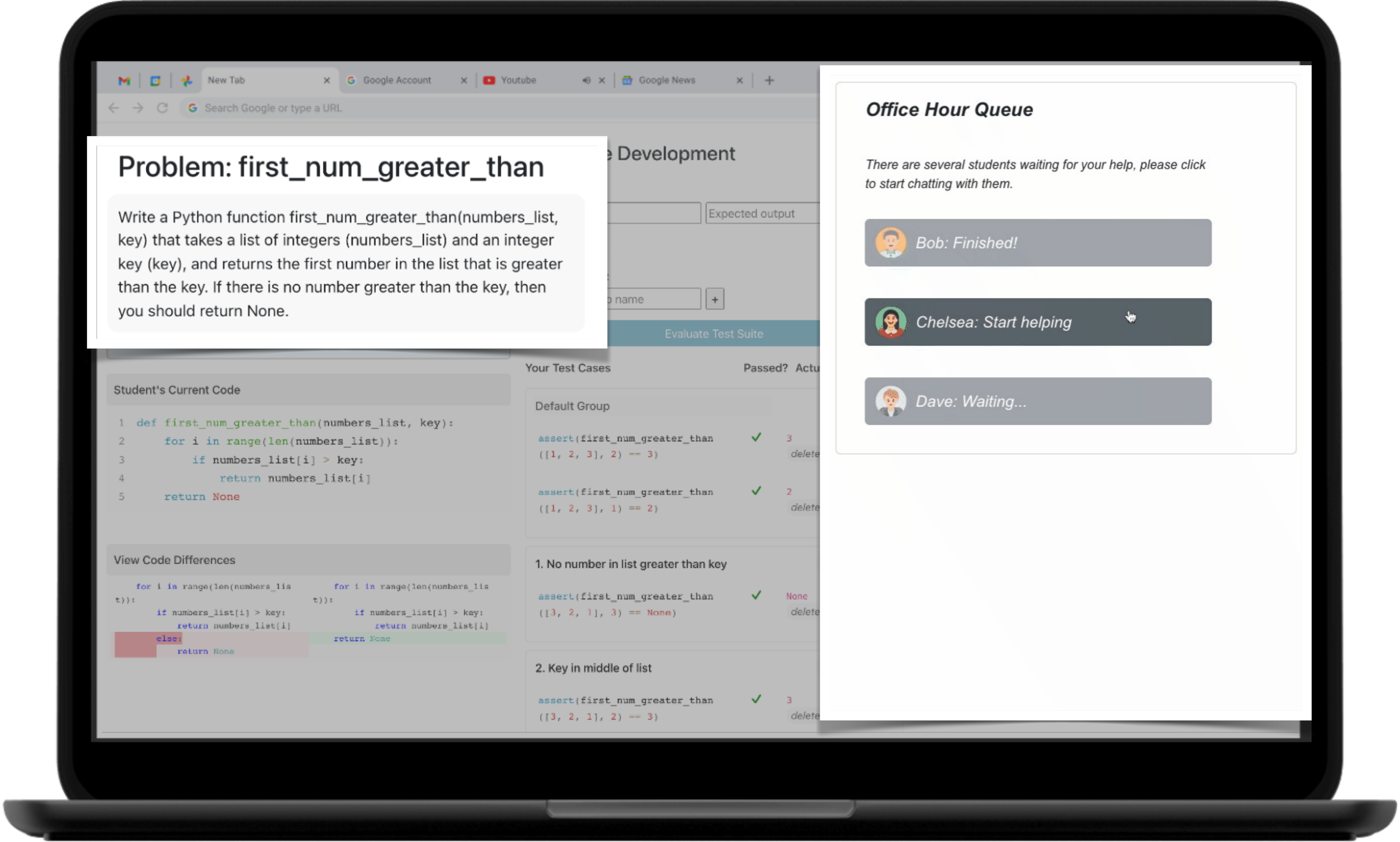


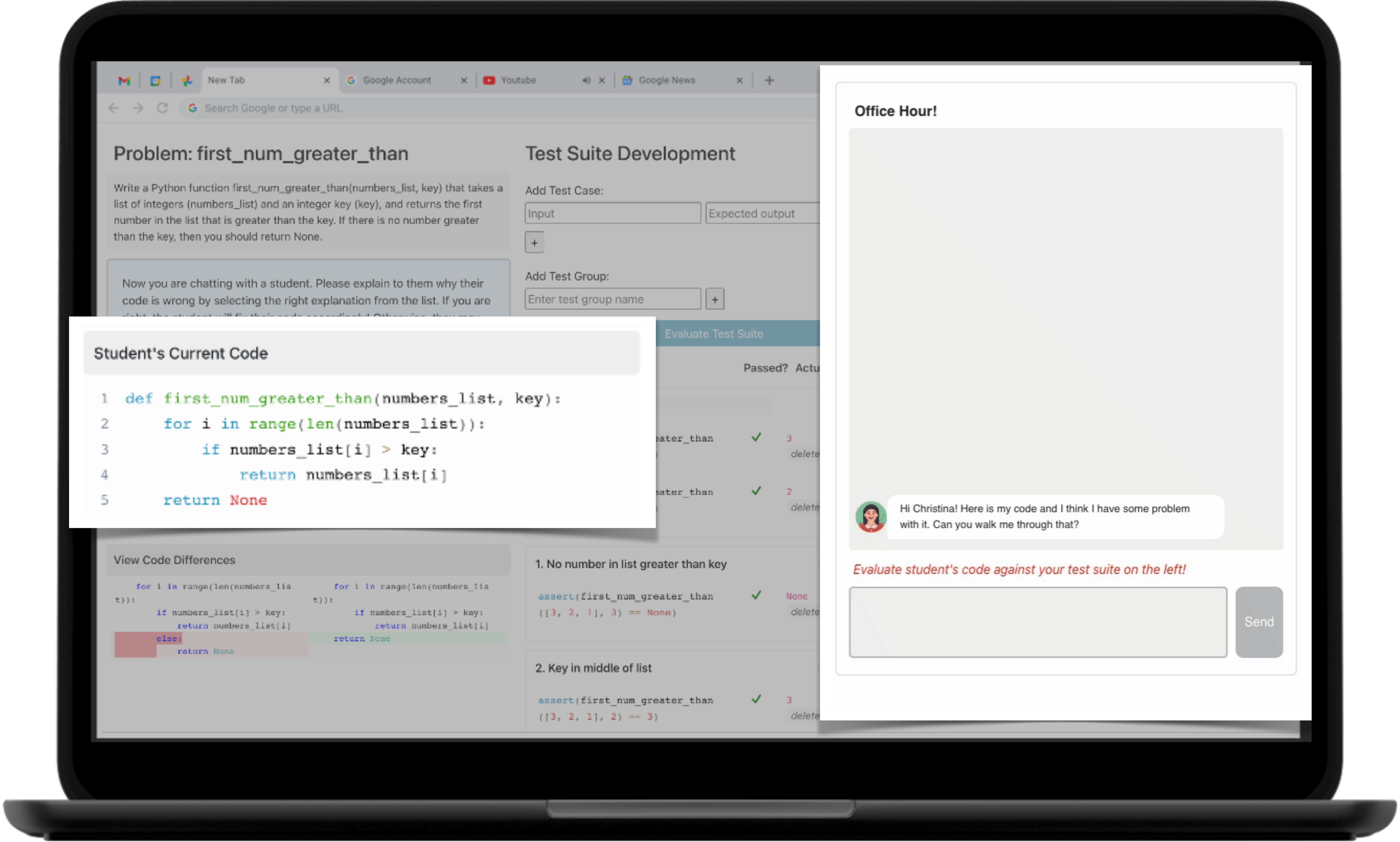
Humans might do more debugging!

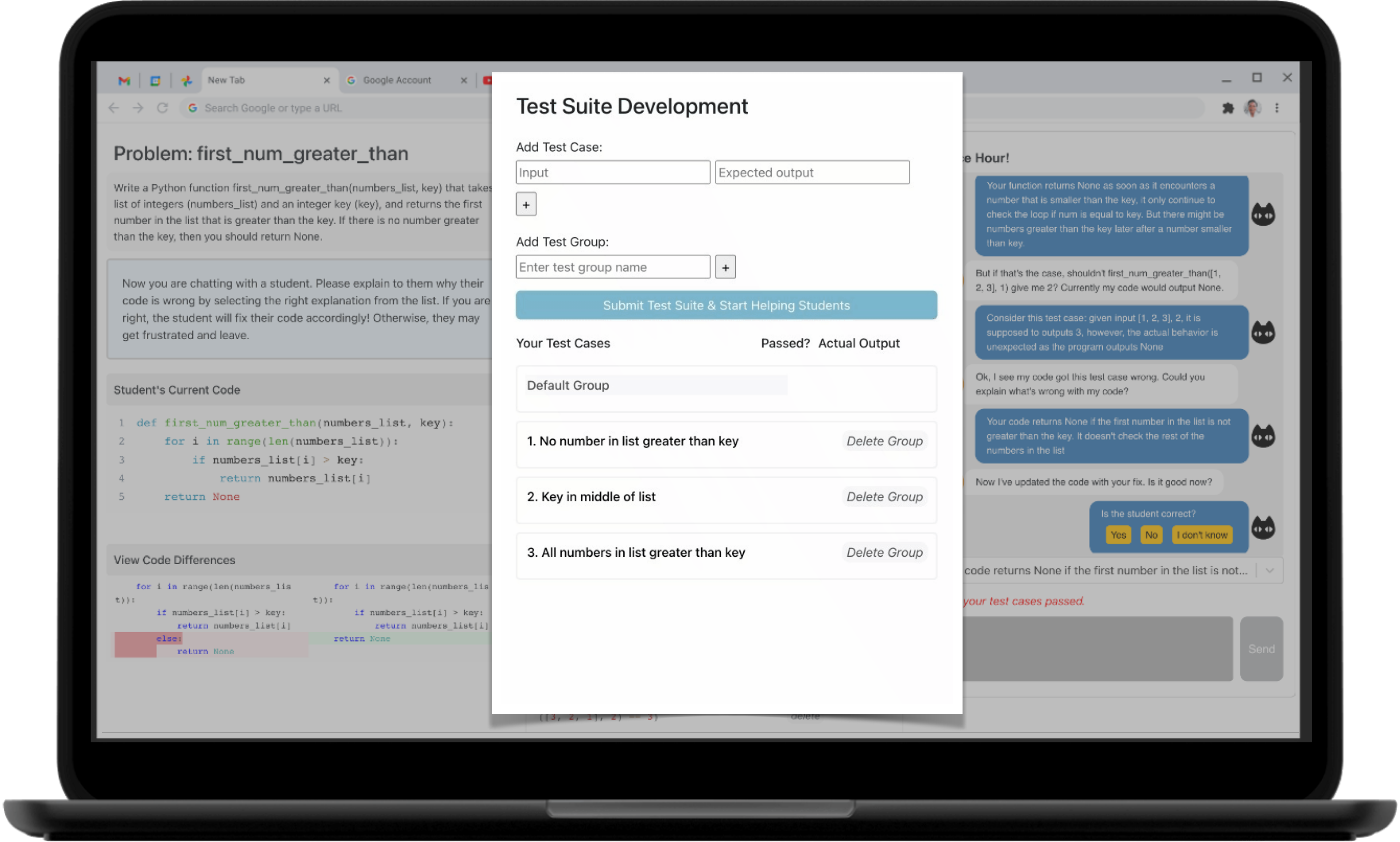


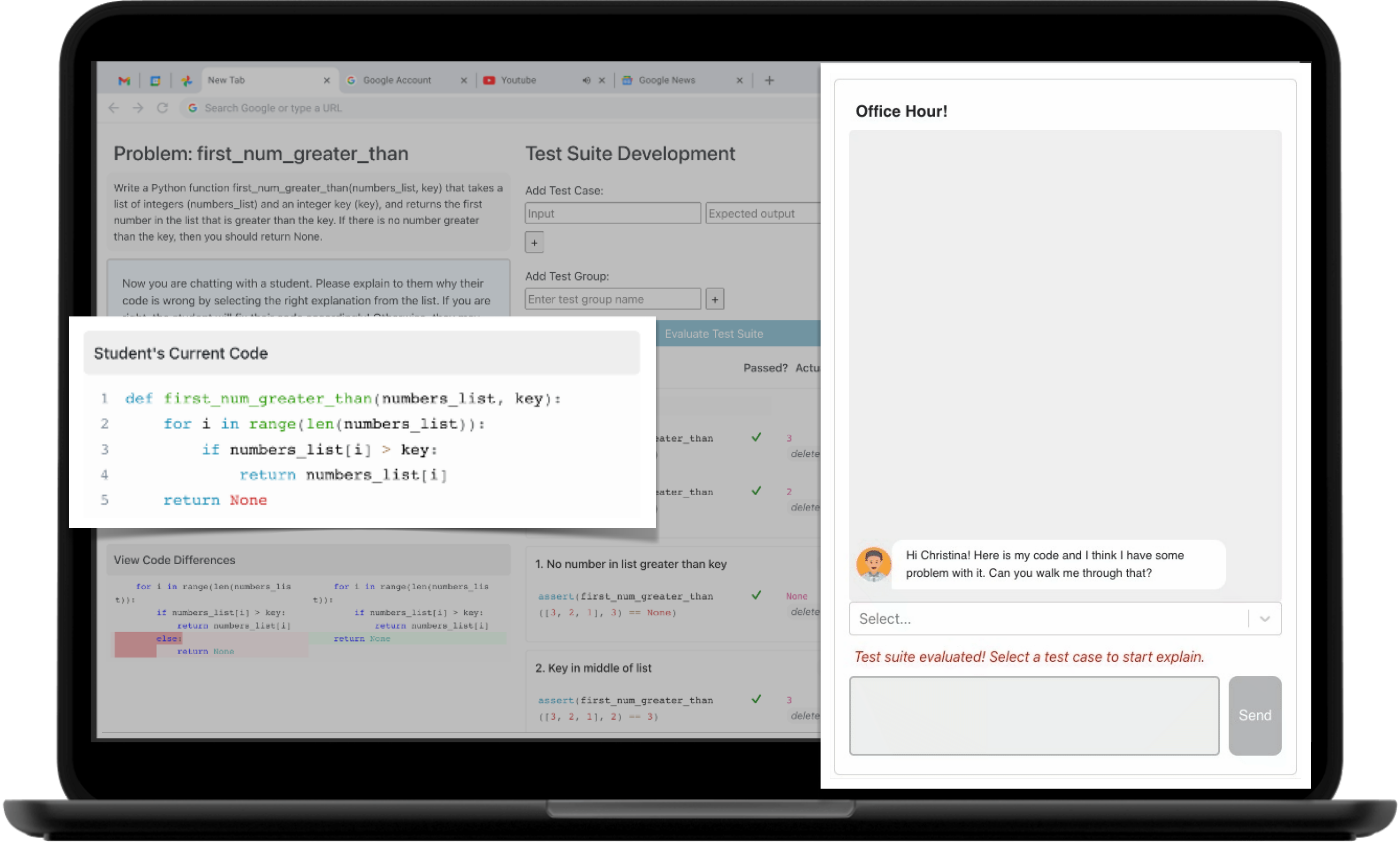
Qianou Ma, et al. "How to Teach Programming in the AI Era? Using LLMs as a Teachable Agent for Debugging." *ICSE 2024*


New Tab
Google Account
Youtube
Google News
Search Google or type a URL
Problem: first_num_greater_than
Write a Python function first_num_greater_than(numbers_list, key) that takes a list of integers (numbers_list) and an integer key (key), and returns the first number in the list that is greater than the key. If there is no number greater than the key, then you should return None.
Now you are chatting with a student. Please explain to them why their code is wrong by selecting the right explanation from the list. If you are right, the student will fix their code accordingly! Otherwise, they may get frustrated and leave.
Student's Current Code
def first_num_greater_than(numbers_list, key):
for i in range(len(numbers_list)):
if numbers_list[i] > key:
return numbers_list[i]
return None
View Code Differences
for i in range(len(numbers_list)):
if numbers_list[i] > key:
return numbers_list[i]
else:
return None
for i in range(len(numbers_list)):
if numbers_list[i] > key:
return numbers_list[i]
return None
Test Suite Development
Add Test Case:
Input
Expected output
+
Add Test Group:
Enter test group name
+
Evaluate Test Suite
Your Test Cases
Passed?
Actual Output
Default Group
assert(first_num_greater_than([1, 2, 3], 2) == 3)
3
delete
assert(first_num_greater_than([1, 2, 3], 1) == 2)
2
delete
1. No number in list greater than key
Delete Group
assert(first_num_greater_than([3, 2, 1], 3) == None)
None
delete
2. Key in middle of list
Delete Group
assert(first_num_greater_than([3, 2, 1], 2) == 3)
3
delete
Office Hour!
Your function returns None as soon as it encounters a number that is smaller than the key, it only continue to check the loop if num is equal to key. But there might be numbers greater than the key later after a number smaller than key.
But if that's the case, shouldn't first_num_greater_than([1, 2, 3], 1) give me 2? Currently my code would output None.
Consider this test case: given input [1, 2, 3], 2, it is supposed to outputs 3, however, the actual behavior is unexpected as the program outputs None
Ok, I see my code got this test case wrong. Could you explain what's wrong with my code?
Your code returns None if the first number in the list is not greater than the key. It doesn't check the rest of the numbers in the list
Now I've updated the code with your fix. Is it good now?
Is the student correct?
Yes
No
I don't know
Your code returns None if the first number in the list is not...
All of your test cases passed.
Send

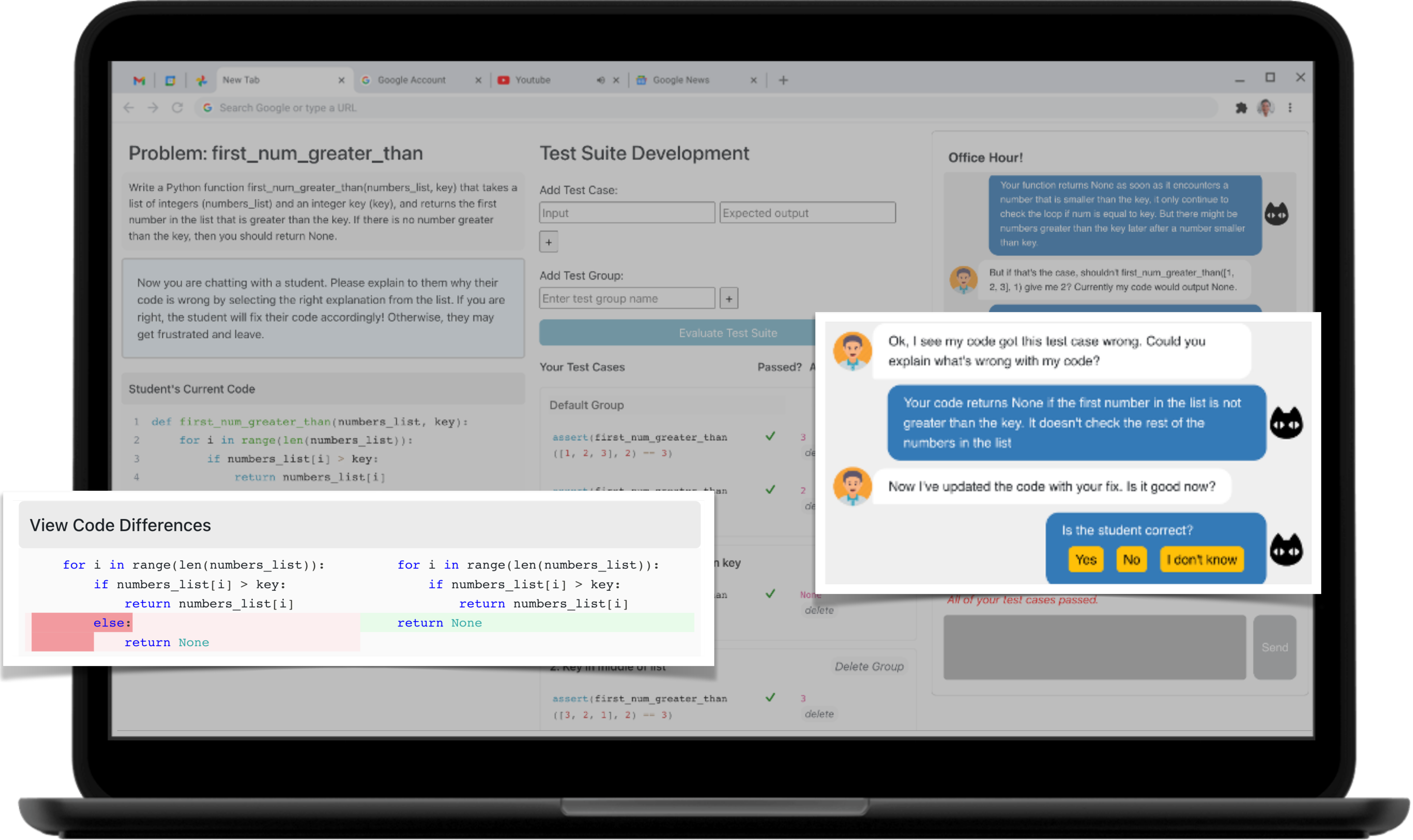

New Tab
Google Account
Youtube
Google News
Search Google or type a URL
Problem: first_num_greater_than
Write a Python function first_num_greater_than(numbers_list, key) that takes a list of integers (numbers_list) and an integer key (key), and returns the first number in the list that is greater than the key. If there is no number greater than the key, then you should return None.
Development
Expected output
Evaluate Test Suite
Your Test Cases
Passed?
Student's Current Code
def first_num_greater_than(numbers_list, key):
for i in range(len(numbers_list)):
if numbers_list[i] > key:
return numbers_list[i]
return None
Default Group
assert(first_num_greater_than([1, 2, 3], 2) == 3)
assert(first_num_greater_than([1, 2, 3], 1) == 2)
View Code Differences
1. No number in list greater than key
assert(first_num_greater_than([3, 2, 1], 3) == None)
2. Key in middle of list
assert(first_num_greater_than([3, 2, 1], 2) == 3)
Office Hour Queue
There are several students waiting for your help, please click to start chatting with them.
Bob: Finished!
Chelsea: Start helping
Dave: Waiting...


New Tab
Google Account
Youtube
Google News
Search Google or type a URL
Problem: first_num_greater_than
Write a Python function first_num_greater_than(numbers_list, key) that takes a list of integers (numbers_list) and an integer key (key), and returns the first number in the list that is greater than the key. If there is no number greater than the key, then you should return None.
Now you are chatting with a student. Please explain to them why their code is wrong by selecting the right explanation from the list. If you are
Student's Current Code
1 def first_num_greater_than(numbers_list, key):
2 for i in range(len(numbers_list)):
3 if numbers_list[i] > key:
4 return numbers_list[i]
5 return None
View Code Differences
Test Suite Development
Add Test Case:
Input
Expected output
Add Test Group:
Enter test group name
Evaluate Test Suite
Passed? Actu
1. No number in list greater than key
assert(first_num_greater_than([3, 2, 1], 3) == None)
2. Key in middle of list
assert(first_num_greater_than([3, 2, 1], 2) == 3)
Office Hour!
Hi Christina! Here is my code and I think I have some problem with it. Can you walk me through that?
Evaluate student's code against your test suite on the left!
Send

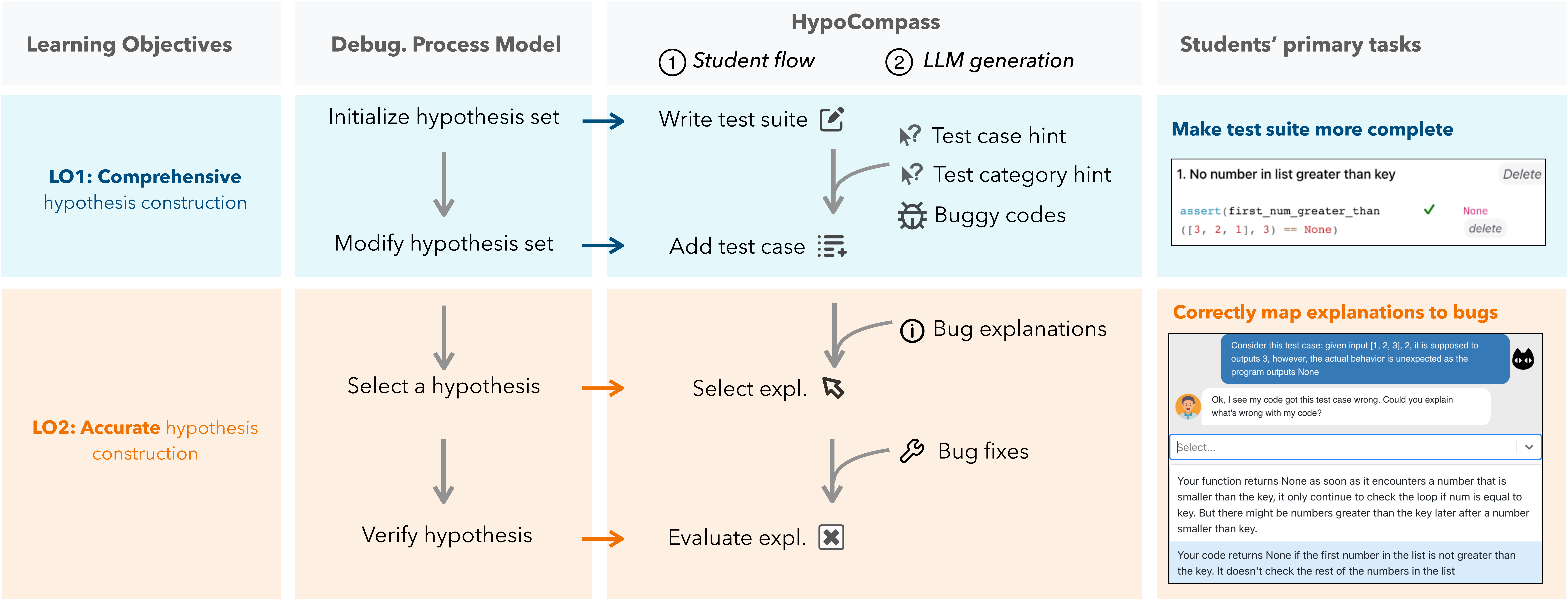

New Tab
Google Account
Search Google or type a URL
Problem: first_num_greater_than
Write a Python function first_num_greater_than(numbers_list, key) that takes list of integers (numbers_list) and an integer key (key), and returns the first number in the list that is greater than the key. If there is no number greater than the key, then you should return None.
Now you are chatting with a student. Please explain to them why their code is wrong by selecting the right explanation from the list. If you are right, the student will fix their code accordingly! Otherwise, they may get frustrated and leave.
Student's Current Code
def first_num_greater_than(numbers_list, key):
for i in range(len(numbers_list)):
if numbers_list[i] > key:
return numbers_list[i]
return None
View Code Differences
Test Suite Development
Add Test Case:
Input
Expected output
Add Test Group:
Enter test group name
Submit Test Suite & Start Helping Students
Your Test Cases
Passed?
Actual Output
Default Group
1. No number in list greater than key
Delete Group
2. Key in middle of list
Delete Group
3. All numbers in list greater than key
Delete Group
Your function returns None as soon as it encounters a number that is smaller than the key, it only continue to check the loop if num is equal to key. But there might be numbers greater than the key later after a number smaller than key.
But if that's the case, shouldn't first_num_greater_than([1, 2, 3], 1) give me 2? Currently my code would output None.
Consider this test case: given input [1, 2, 3], 2, it is supposed to outputs 3, however, the actual behavior is unexpected as the program outputs None
Ok, I see my code got this test case wrong. Could you explain what's wrong with my code?
Your code returns None if the first number in the list is not greater than the key. It doesn't check the rest of the numbers in the list
Now I've updated the code with your fix. Is it good now?
Is the student correct?
Yes
No
I don't know
your test cases passed.
Send



New Tab
Google Account
Youtube
Google News
Search Google or type a URL
Problem: first_num_greater_than
Write a Python function first_num_greater_than(numbers_list, key) that takes a list of integers (numbers_list) and an integer key (key), and returns the first number in the list that is greater than the key. If there is no number greater than the key, then you should return None.
Now you are chatting with a student. Please explain to them why their code is wrong by selecting the right explanation from the list. If you are
Student's Current Code
def first_num_greater_than(numbers_list, key):
    for i in range(len(numbers_list)):
        if numbers_list[i] > key:
            return numbers_list[i]
    return None
View Code Differences
for i in range(len(numbers_list)):
if numbers_list[i] > key:
return numbers_list[i]
else:
return None
return None
Test Suite Development
Add Test Case:
Input
Expected output
+
Add Test Group:
Enter test group name
+
Evaluate Test Suite
Passed? Actu
1. No number in list greater than key
assert(first_num_greater_than([3, 2, 1], 3) == None)
None
2. Key in middle of list
assert(first_num_greater_than([3, 2, 1], 2) == 3)
3
Office Hour!
Hi Christina! Here is my code and I think I have some problem with it. Can you walk me through that?
Select...
Test suite evaluated! Select a test case to start explain.
Send


New Tab
Google Account
Youtube
Google News
Search Google or type a URL
Problem: first_num_greater_than
Write a Python function first_num_greater_than(numbers_list, key) that takes a list of integers (numbers_list) and an integer key (key), and returns the first number in the list that is greater than the key. If there is no number greater than the key, then you should return None.
Now you are chatting with a student. Please explain to them why their code is wrong by selecting the right explanation from the list. If you are right, the student will fix their code accordingly! Otherwise, they may get frustrated and leave.
Student's Current Code
def first_num_greater_than(numbers_list, key):
for i in range(len(numbers_list)):
if numbers_list[i] > key:
return numbers_list[i]
Test Suite Development
Add Test Case:
Input
Expected output
Add Test Group:
Enter test group name
Evaluate Test Suite
Your Test Cases
Passed?
Default Group
assert(first_num_greater_than ([1, 2, 3], 2) == 3)
Delete Group
assert(first_num_greater_than ([3, 2, 1], 2) == 3)
delete
Office Hour!
Your function returns None as soon as it encounters a number that is smaller than the key, it only continue to check the loop if num is equal to key. But there might be numbers greater than the key later after a number smaller than key.
But if that's the case, shouldn't first_num_greater_than([1, 2, 3], 1) give me 2? Currently my code would output None.
All of your test cases passed.
Send
View Code Differences
for i in range(len(numbers_list)):
if numbers_list[i] > key:
return numbers_list[i]
else:
return None
for i in range(len(numbers_list)):
if numbers_list[i] > key:
return numbers_list[i]
return None
Ok, I see my code got this test case wrong. Could you explain what's wrong with my code?
Your code returns None if the first number in the list is not greater than the key. It doesn't check the rest of the numbers in the list
Now I've updated the code with your fix. Is it good now?
Is the student correct?
Yes
No
I don't know

HypoCompass: Learning Theory Inspired Design



Explicit & deliberate training just on debugging, by off-loading other necessary sub-tasks to LLMs (e.g., writing the bug, correcting the bug, etc.)

HypoCompass: Effectiveness

HypoCompass consistently generates high-quality training materials: 90% success rate + only take 15 minutes to label and edit (reduce instructor effort!)

HypoCompass brings learning gain: In a pre-to-post test setup, 10 novices improved their performances by **17%**, with a reduced completion time of **13%.**

It is possible to eventually train students to be better at debugging!

Question 2 Select all that apply.

Imagine a solution to this problem that fails on all of these following test case(s) because of the same bug. What could the bug be?

```python
assert(num_smaller([10, 10, 10, 20, 30], 10) == 0)
assert(num_smaller([10, 20, 20, 30, 30], 20) == 1)
assert(num_smaller([10, 10, 20, 30, 30], 30) == 3)
```

Select all that apply.

A. The buggy codes may have overlooked duplicated elements in seq
B. The buggy codes may have overlooked x when it is the smallest element in seq
C. The buggy codes may only handle the case where x is not in seq
D. The buggy codes may only handle the case where x is already in seq

Answer:

Question 3.1 Select one answer.

Given th
addition
which te
suite wo

```
Pytho
asse
asse
asse
```

A.
B.

Question 7 Select one answer.

Test case 2: `assert(remove_extras_code2([1, 1, 2, 3]) == [1, 2, 3])`
Actual behavior: `'TypeError' 'int' object is not iterable.`

```python
def remove_extras_code2(lst):
    new_lst = []
    for i in lst:
        if i == lst[i+1]:
            continue
        else:
            new_lst += i
    return new_lst
```

What's the bug exposed by this test case?

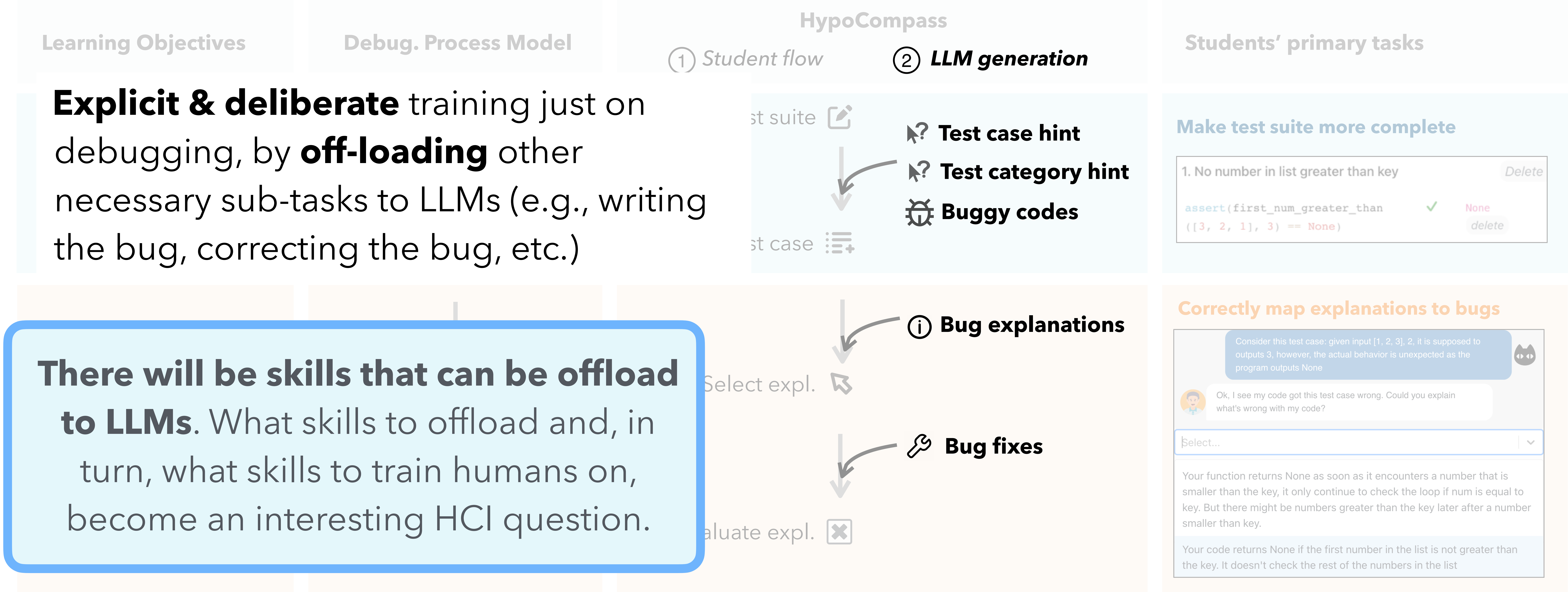
A. The bug occurs because the loop variable **i** is mistakenly used as both the element and index of the list. This leads to incorrect comparisons and triggers a **TypeError** in **lst[i+1]** because **i** is an element of the list, not an index.

B. The bug is caused by not initializing the **new_lst** properly. The code fails to explicitly assign an empty list to **new_lst**, so when concatenating elements to **new_lst** using the **+=** operator, a **TypeError** occurs because **new_lst** is not iterable.

C. The bug is due to an incorrect conditional statement. The code incorrectly compares **i** with **lst[i+1]** instead of comparing adjacent elements of the list, which triggers **TypeError** when trying to compare an integer **i** with a list element.

D. The bug occurs because the code incorrectly assumes that **i** is iterable when concatenating it to **new_lst** with the **+=** operator. In this case, **i** is an integer, which is not iterable, and it causes a **TypeError**.

HypoCompass: Learning Theory Inspired Design



Explicit & deliberate training just on debugging, by **off-loading** other necessary sub-tasks to LLMs (e.g., writing the bug, correcting the bug, etc.)

There will be skills that can be offload to LLMs. What skills to offload and, in turn, what skills to train humans on, become an interesting HCI question.

Explicit & deliberate training just on debugging, by off-loading other necessary sub-tasks to LLMs (e.g., writing the bug, correcting the bug, etc.)

Task formation in HypoCompass (bug fixing)

LLM task: To edit the buggy code according to the fix instruction without over- or under- fix

```python
def first_num_greater_than (numbers_list, key):
    for i in range(len(numbers_list)):
        if numbers_list[i] <= key:
            return num
        else:
            return None
```

"Change the comparison line to be larger than key."

```
You fix bugs in Python code closely following the instructions.
Original code: {buggy_code};
Code modification: {explanation}
Modified code:
```

```python
def first_num_greater_than (numbers_list, key):
    for i in range(len(numbers_list)):
        if numbers_list[i] > key:
            return num
    return None
```

Over-fixing, because LLM wants to continue to generate correct code!

Task formation in HypoCompass (bug fixing)

LLM task: To edit the buggy code according to the fix instruction without over- or under- fix

```python
def first_num_greater_than (numbers_list, key):
    for i in range(len(numbers_list)):
        if numbers_list[i] <= key:
            return num
        else:
            return None
```

"Change the comparison line to be larger than key."

```
You fix bugs in Python code closely following the instructions.
Original code: {buggy_code};
Code modification: {explanation}

Translate the statement into actual, minimal code change in this format:
{original code snippet: ""copy the lines of code that need editing""
-> edited code snippet: ""write the edited code snippet""}
```

`numbers_list[i] <= key` → `numbers_list[i] > key`

Task formation helps avoid competing tasks of code editing and code completion!

That's all for today!!