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?
<table>
<thead>
<tr>
<th>Outcomes</th>
<th>Human-AI (Copilot)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality</td>
<td>☢️ vs. Human-Human: more lines of code deleted in next session (lower quality) [9]</td>
</tr>
<tr>
<td></td>
<td>☢️ vs. Human Solo: significantly improve correctness score and reduce encountered errors for novice students [10]</td>
</tr>
<tr>
<td></td>
<td>☻️ vs. Human Solo: no significant difference in task success [11] or task success rate in given time [12]</td>
</tr>
<tr>
<td>Productivity</td>
<td>☢️ vs. Human-Human: more lines of added code [9]</td>
</tr>
<tr>
<td></td>
<td>☢️ vs. Human Solo: 55.8% reduction in completion time [11]</td>
</tr>
<tr>
<td></td>
<td>☢️ vs. Human Solo: significantly increase task completion and reduce task completion time for novice students [10]</td>
</tr>
<tr>
<td></td>
<td>☻️ vs. Human Solo: no significant difference in the task completion rate in given time [12]</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>☢️ vs. Human Solo: higher self-ratings of satisfaction [12, 16, 17]</td>
</tr>
<tr>
<td>Learning</td>
<td>☻️ 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]</td>
</tr>
<tr>
<td>Cost</td>
<td>No experiment yet. Vaithilingam et al. [12], Bird et al. [16] hypothesized that human-AI may lead to more unnecessary debugging vs. Human Solo</td>
</tr>
</tbody>
</table>
How do you know a code gen model is useful?

A: define metrics more specific for usability tests!
Human-Human vs. Human-AI pAlr Programming

Human-AI pAlr 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-Al pAlr Programming." AIED 2023 workshop
<table>
<thead>
<tr>
<th>Outcomes</th>
<th>Human-AI (Copilot)</th>
</tr>
</thead>
</table>
| 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] |
<p>| 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 |</p>
<table>
<thead>
<tr>
<th>Outcomes</th>
<th>Human-Human vs. Human Solo</th>
<th>Human-Al (Copilot)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality</td>
<td>- significantly lower defect density for complex code [7]</td>
<td>- vs. Human-Human: more lines of code deleted in next session (lower quality) [9]</td>
</tr>
<tr>
<td></td>
<td>- no difference for simpler code [7]</td>
<td>- vs. Human Solo: significantly improve correctness score and reduce encountered errors for novice students [10]</td>
</tr>
<tr>
<td></td>
<td>- significantly higher percentage of test cases passed [8]</td>
<td>- vs. Human Solo: no significant difference in task success [11] or task success rate in given time [12]</td>
</tr>
<tr>
<td>Productivity</td>
<td>- significantly fewer lines of code per person hour writing simpler code [7]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- no significant difference writing more complex code [7]</td>
<td>- vs. Human-Human: more lines of added code [9]</td>
</tr>
<tr>
<td></td>
<td>- 29% shorter time to complete task (pair speed advantage = 1.4) [13]</td>
<td>- vs. Human Solo: significantly increasing code production time and number of detected errors [13]</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>- higher self-ratings of satisfaction [14]</td>
<td>- vs. Human Solo: no significant difference in the task completion rate in given time [12]</td>
</tr>
<tr>
<td></td>
<td>- students with greater self-confidence and self-efficacy less enjoy the pair programming experience [15]</td>
<td></td>
</tr>
<tr>
<td>Learning</td>
<td>- higher grades, exam scores [18], and retention [19]</td>
<td>- 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]</td>
</tr>
<tr>
<td></td>
<td>- significantly higher gains in exam performance in female students than male students  [20]</td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td>- increased management workload to match, schedule a pair, resolve collaboration conflict, assess individual contributions, etc. [21]</td>
<td>No experiment yet. Vaithilingam et al. [12], Bird et al. [16] hypothesized that human-Al may lead to more unnecessary debugging vs. Human Solo</td>
</tr>
<tr>
<td></td>
<td>- reduced teaching staff workload (grading one assignment from a pair) [8]</td>
<td></td>
</tr>
</tbody>
</table>
Example Usability Metric: Syntactic similarity metric

“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.”


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.
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

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

**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

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.”

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

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?
Understand humans qualitatively: Surveys

<table>
<thead>
<tr>
<th>Motivation</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A. For using</strong></td>
<td></td>
</tr>
<tr>
<td>M1 To have an autocomplete or reduce the amount of keystrokes I make.</td>
<td>86%</td>
</tr>
<tr>
<td>M2 To finish my programming tasks faster.</td>
<td>76%</td>
</tr>
<tr>
<td>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.</td>
<td>68%</td>
</tr>
<tr>
<td>M4 To discover potential ways or starting points to write a solution to a problem I’m facing.</td>
<td>50%</td>
</tr>
<tr>
<td>M5 To find an edge case for my code I haven’t considered.</td>
<td>36%</td>
</tr>
</tbody>
</table>

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

#### A. For using

<table>
<thead>
<tr>
<th>Motivation</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1 To have an autocomplete or reduce the amount of keystrokes I make.</td>
<td>86% Very important, 6.2% Not important at all</td>
</tr>
<tr>
<td>M2 To finish my programming tasks faster.</td>
<td>76% Very important, 12% Not important at all</td>
</tr>
<tr>
<td>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.</td>
<td>68% Very important, 14% Not important at all</td>
</tr>
<tr>
<td>M4 To discover potential ways or starting points to write a solution to a problem I’m facing.</td>
<td>50% Very important, 24% Not important at all</td>
</tr>
<tr>
<td>M5 To find an edge case for my code I haven’t considered.</td>
<td>36% Very important, 44% Not important at all</td>
</tr>
</tbody>
</table>

#### B. For not using

<table>
<thead>
<tr>
<th>Motivation</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>M6 Code generation tools write code that doesn’t meet functional or non-functional (e.g., security, performance) requirements that I need.</td>
<td>54% Very important, 34% Not important at all</td>
</tr>
<tr>
<td>M7 It’s hard to control code generation tools to get code that I want.</td>
<td>48% Very important, 36% Not important at all</td>
</tr>
<tr>
<td>M8 I spend too much time debugging or modifying code written by code generation tools.</td>
<td>38% Very important, 45% Not important at all</td>
</tr>
<tr>
<td>M9 I don’t think code generation tools provide helpful suggestions.</td>
<td>34% Very important, 46% Not important at all</td>
</tr>
<tr>
<td>M10 I don’t want to use a tool that has access to my code.</td>
<td>30% Very important, 51% Not important at all</td>
</tr>
<tr>
<td>M11 I write and use proprietary code that code generation tools haven’t seen before and don’t generate.</td>
<td>28% Very important, 59% Not important at all</td>
</tr>
<tr>
<td>M12 To prevent potential intellectual property infringement.</td>
<td>26% Very important, 66% Not important at all</td>
</tr>
<tr>
<td>M13 I find the tool’s suggestions too distracting.</td>
<td>26% Very important, 51% Not important at all</td>
</tr>
<tr>
<td>M14 I don’t understand the code written by code generation tools.</td>
<td>16% Very important, 76% Not important at all</td>
</tr>
<tr>
<td>M15 I don’t want to use open-source code.</td>
<td>10% Very important, 89% Not important at all</td>
</tr>
</tbody>
</table>

Legend: 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*

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...

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

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

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

A: Design and iterate on the interface
Communication via UI: Inline explanations

```python
def create_window():
    window = tk.Tk()
    window.title("COVID-19 Visualization")
    window.geometry("1000x600")
    window.resizable(False, False)
    canvas = tk.Canvas(window, background='white',
                        width=800, height=600,
                        orient='horizontal',
                        highlightthickness=0)
    scrollbar = tk.Scrollbar(window, orient='vertical',
                              command=canvas.yview)
    scrollbar.pack(side='right', fill='y')
    canvas.config(yscrollcommand=scrollbar.set)
    canvas.pack(side='bottom', fill='both', expand=True)
```

1. Create a window with the title "COVID-19 Visualization" and size 1000x600.
2. Create a canvas with background color white and size 800x600.
3. Create a scrollbar for the canvas.

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, ➊), IVIE annotates it with brief, informative explanations. Explanations appear at the level of blocks of code (in the right margin, ➋) and expressions (anchored beneath the line the programmer hovers over, ➌). 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.

Communication via UI: Inline explanations

Figure 4: Explaining expressions. After a programmer makes a call, IVIE reveals explanations of the code. The purpose of these explanations is to make the programmer seek understanding the precise behavior of their code.

```python
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 '_apr'.

```python
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('Yearly Average Temperature for (max_temp_city)')
    ax[0].set_ylabel('Year')
    ax[0].set_ylabel('Temperature (°C)')
    ax[0].yaxis.set_major_formatter(ticker.FormatStrFormatter('%.1f'))

    df[df['City'] == max_rain_city]['Rainfall'].plot(ax=ax[1], color='green')
    ax[1].set_title('Yearly Average Rainfall for (max_rain_city)')
    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!

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:**
```python
fig, ax = plt.subplots(2, 1, figsize=(14, 10))
```
**Output:**
```python
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:**
```python
df_all.merge(df_Apr, on='City', how='left', suffixes=('_all', '_apr'))
```
**Output:**

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

**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

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.”
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-to-measure 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.
"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."

Filters between model and usage
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

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 location of the comment and the mention of “check” and “null” were sufficient to trigger the assistant to suggest the intended edit.
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 ML-suggested 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

GitHub Copilot is powered by Codex, the new AI system created by OpenAI. GitHub Copilot understands significantly more context than most code assistants. So, whether it’s in a doctstring, comment, function name, or the code itself, GitHub Copilot uses the context you’ve provided and synthesizes code to match. Together with OpenAI, we’re designing GitHub Copilot to get smarter at producing safe and effective code as developers use it.

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
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`, 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

```python
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

```python
for i in range(len(numbers_list)):
    if numbers_list[i] > key:
        return numbers_list[i]
```

Test Suite Development

Add Test Case:

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Add Test Group:

Enter test group name: *

Evaluate Test Suite

Your Test Cases | Passed? Actual Output
---|---

Passed Group

- Default Group

Passed: 3

1. No number in list greater than key

Passed: None

2. Key in middle of list

Passed: 3

3. No number in list greater than key

Passed: None

All your test cases pass!
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.
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 unsure, please select 'I don’t know'.

Student's Current Code

```python
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
```
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

```python
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

```python
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: 

Submit Test Suite & Start Helping Students

Your Test Cases

<table>
<thead>
<tr>
<th>Default Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. No number in list greater than key</td>
</tr>
<tr>
<td>2. Key in middle of list</td>
</tr>
<tr>
<td>3. All numbers in list greater than key</td>
</tr>
</tbody>
</table>
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`, 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 not sure, please select the last choice.

Student's Current Code

```python
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
```

- No number in list greater than key
- Key in middle of list
- Key in first position
- All numbers in list less than key
- Key is greater than any number in list
- Key is equal to any number in list
- None of the above

Test suite evaluated! Select a test case to start explain.
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`, 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

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

Add Test Case:

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected output</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1, 2, 3]</td>
<td>3</td>
</tr>
</tbody>
</table>

Add Test Group:

Enter test group name: *

Evaluate Test Suite:

Your function returns None as soon as it encounters a number that is greater than the key. It doesn’t continue to check the rest of the list if the first number is greater than the key. It doesn’t check the rest of the numbers in the list.

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

### Learning Objectives

<table>
<thead>
<tr>
<th>LO1: Comprehensive hypothesis construction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize hypothesis set</td>
</tr>
<tr>
<td>Modify hypothesis set</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LO2: Accurate hypothesis construction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select a hypothesis</td>
</tr>
<tr>
<td>Verify hypothesis</td>
</tr>
</tbody>
</table>

### Debug. Process Model

1. Initialize hypothesis set
2. Modify hypothesis set
3. Select a hypothesis
4. Verify hypothesis
5. Write test suite
6. Add test case
7. Select test case hint
8. Test category hint
9. Buggy codes
10. Write test suite
11. Add test case

### HypoCompass

1. Student flow
2. LLM generation

### Students’ primary tasks

1. Make test suite more complete
2. Correctly map explanations to bugs

#### 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, 30, 30], 20) == 1)
assert(num_smaller([10, 10, 20, 30, 30], 20) == 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 the additional test suite which tests the code, select which one you think it is.

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

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

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_list` properly. The code fails to explicitly assign an empty list to `new_list`, so when concatenating elements to `new_list` using the `+=` operator, a `TypeError` occurs because `new_list` 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_list` 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

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

```
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.”

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

```
def first_num_greater_than(numbers_list, key):
    for i in range(len(numbers_list)):
        if numbers_list[i] > key:
            return num
    return None
```
Task formation in HypoCompass (bug fixing)

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

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"}

```
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.”

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

numbers_list[i] <= key → numbers_list[i] > key
That’s all for today!!