Live Exploration of AI-Generated Programs

Kasra Ferdowsi

with Ruanqianqian (Lisa) Huang*, Michael B. James, Nadia Polikarpova, and Sorin Lerner
Overview

1. Motivation & Background:
   a. Grounded Copilot
   b. Live Programming

2. LEAP: Live Exploration of AI-Generated Code

3. User Study

4. Findings
   a. Validating suggestions
   b. Over-/Under-reliance
   c. Cognitive Load
   d. Impressions

Slides based on Polikarpova 2023.
Overview

1. Motivation & Background:
   a. Grounded Copilot
   b. Live Programming

2. LEAP: Live Exploration of AI-Generated Code

3. User Study

4. Findings
   a. Validating suggestions
   b. Over-/Under-reliance
   c. Cognitive Load
   d. Impressions

Slides based on Polikarpova 2023.
LLMs for Code Generation*

GitHub
Copilot

OpenAI
ChatGPT

* For experienced programmers.
1. **Grounded Copilot: How Programmers Interact with Code-Generating Models.**
   Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023.

2. **Understanding the Usability of AI Programming Assistants.**
   Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2023.

3. **Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming.**

4. **Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models.**
   Priyan Vaithilingam, Tianyi Zhang, and Elena Glassman. 2022.
Background

In summary, programmers using AI-generated code...

- Spend significant time validating code suggestions,
- Have trouble evaluating the correctness of generated code,
- Choose validation strategies based on time cost, and so
- Both under- and over-rely on AI code suggestions.
"User interactions with Copilot can be classified into two modes—acceleration and exploration—akin to the two systems of thought in dual-process theories of cognition"*

* Grounded Copilot: How Programmers Interact with Code-Generating Models
Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023.
## Grounded Copilot

<table>
<thead>
<tr>
<th>Acceleration</th>
<th>vs.</th>
<th>Exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td>unintentional</td>
<td></td>
<td>intentional with comments / invoke side panel</td>
</tr>
<tr>
<td>&quot;pattern matching&quot;</td>
<td></td>
<td>explicit validation via elimination / execution / documentation</td>
</tr>
<tr>
<td>unit of focus (sub-expression / statement)</td>
<td></td>
<td>entire function + multiple alternatives</td>
</tr>
<tr>
<td>unwilling to edit</td>
<td></td>
<td>willing to edit / debug / &quot;rip apart&quot; / cherry-pick</td>
</tr>
</tbody>
</table>

**Promoting**

**Validation**

**Scope**

**Mismatch**

**Tolerance**
## Grounded Copilot

<table>
<thead>
<tr>
<th>Acceleration</th>
<th>vs.</th>
<th>Exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td>unintentional</td>
<td>Prompting</td>
<td>intentional with comments / invoke side panel</td>
</tr>
<tr>
<td>&quot;pattern matching&quot;</td>
<td>Validation</td>
<td>explicit validation via elimination / execution / documentation</td>
</tr>
<tr>
<td>unit of focus (sub-expression / statement)</td>
<td>Scope</td>
<td>entire function + multiple alternatives</td>
</tr>
<tr>
<td>unwilling to edit</td>
<td>Mismatch</td>
<td>willing to edit / debug / &quot;rip apart&quot; / cherry-pick</td>
</tr>
<tr>
<td></td>
<td>Tolerance</td>
<td></td>
</tr>
</tbody>
</table>
Live Programming

Demo!
Research Question:

Does *Live Programming* offer a good interaction for *exploring* AI-generated code?
Overview

1. Motivation & Background:
   a. Grounded Copilot
   b. Live Programming

2. LEAP: Live Exploration of AI-Generated Code

3. User Study

4. Findings
   a. Validating suggestions
   b. Over-/Under-reliance
   c. Cognitive Load
   d. Impressions

Slides based on Polikarpova 2023.
Overview

1. Motivation & Background:
   a. Grounded Copilot
   b. Live Programming

2. LEAP: Live Exploration of AI-Generated Code

3. User Study

4. Findings
   a. Validating suggestions
   b. Over-/Under-reliance
   c. Cognitive Load
   d. Impressions

Slides based on Polikarpova 2023.
Copilot's Side Panel

```python
import matplotlib.pyplot as plt
import pandas as pd

# Get the data, assuming we have the following
# column names: "group", "time", "success"

df = import_data('my_data.csv')

# create a box plot where each box refers to a group
```

**Synthesizing 10/10 solutions (Duplicates hidden)**

Accept Solution

----------

**Suggestion 1**

```
# and the y axis is the time it took to complete the task
df.boxplot(column='time', by='group')
plt.show()
```

This is not necessarily the most efficient way to do it

Accept Solution

----------

**Suggestion 2**

```
fig, ax = plt.subplots()
ax.boxplot(df['time'], labels=df['group'])
ax.set_ylabel('Time (s)')
ax.set_title('Time taken to complete a task')
plt.show()
```

You can find more information about box plots here.
Demo!
Overview

1. Motivation & Background:
   a. Grounded Copilot
   b. Live Programming

2. LEAP: Live Exploration of AI-Generated Code

3. User Study

4. Findings
   a. Validating suggestions
   b. Over-/Under-reliance
   c. Cognitive Load
   d. Impressions

Slides based on Polikarpova 2023.
Overview

1. Motivation & Background:
   a. Grounded Copilot
   b. Live Programming

2. LEAP: Live Exploration of AI-Generated Code

3. User Study

4. Findings
   a. Validating suggestions
   b. Over-/Under-reliance
   c. Cognitive Load
   d. Impressions

Slides based on Polikarpova 2023.
Research Questions

How does Live Programming effect...

1. Code Correctness
2. Over-/Under-Reliance on AI
3. Cognitive Load
4. Users' Impressions
Experimental Conditions

no-PB

AI suggestions
+
Manually Invoked Terminal Output

PB

AI Suggestions
+
Projection Boxes
Tasks

**Pandas**
clean dataframe and compute stats using pandas

**Box Plot**
overlay scatter plot over boxplot using matplotlib

**Bigrams**
find the most frequent bigram in a string

**String Rewriting**
parse rewrite rules and apply to a string
Participants

n = 17

Occupation:
15 academia
2 industry

Python Usage:
2 occasionally
8 regularly
7 almost every day
Overview

1. Motivation & Background:
   a. Grounded Copilot
   b. Live Programming

2. LEAP: Live Exploration of AI-Generated Code

3. User Study

4. Findings
   a. Validating suggestions
   b. Over-/Under-reliance
   c. Cognitive Load
   d. Impressions

Slides based on Polikarpova 2023.
RQ1: Correctness

LEAP helps validate suggestions! (But does not help fix incorrect ones)
RQ2: Over-/Under-reliance

6 no-PB vs 0 PB participants **mis-judged** correctness of their solutions.
RQ2: Over-/Under-reliance

"it was easy to understand the behavior of a code suggestion because the little boxes on the side allowed for you to preview the results." (P3)

"it saved me the effort of writing multiple print statements." (P1)

LEAP reduces over-/under-reliance on AI, by lowering the cost of validation.
LEAP significantly reduced the cognitive load of exploring AI suggestions on tasks amenable to validation by execution.
LEAP was more **usable** and more **useful**.
Overview

1. Motivation & Background:
   a. Grounded Copilot
   b. Live Programming

2. LEAP: Live Exploration of AI-Generated Code

3. User Study

4. Findings
   a. Validating suggestions
   b. Over-/Under-reliance
   c. Cognitive Load
   d. Impressions

Slides based on Polikarpova 2023.
Summary

1. Background:
   a. Live Programming, for
   b. Exploration of AI-Generated Code

2. LEAP: Projection Boxes + Copilot-like interface

3. User Study w/ 17 participants

4. Found that LEAP...
   a. Helps with validating suggestions,
   b. Reduces Over-/Under-reliance,
   c. Improves Cognitive Load, and
   d. Leaves a positive impressions on participants.