Exploring and Validating AI-Generated Programs Through Concrete Values

Kasra Ferdowsi
The Usability of LLM Code Generation

GitHub Copilot

OpenAI ChatGPT

```python
// Fast inverse square root
def f_inv_sqrt(x: f32) -> f32 {
    let x2 = x * 0.5;
    let mut i = x.to_bits();
    i = 0x5f3759df - (i >> 1);
    let y = f32::from_bits(i);
    y * (1.5 - (x2 * y * y))
}
```
The Usability of LLM Code Generation

It’s Like GPT-3: It’s Fun, Fast, and It’s Only Going to Get Better

OpenAI’s new tool can create code that’s ahead of what professional coders can imagine. But worry not: It could also riddle the internet.

AI Is Generating Security Risks Faster Than Companies Can Keep Up

Rapid growth of generative AI-based software is challenging business technology leaders to keep potential cybersecurity issues in check.

Dumber at Basic Math

AI chatbots have stoked fears that they could spin out of control, but they also suffer from a type of deterioration called ‘drift’.
The Usability of LLM Code Generation
In Summary…

Validating AI-generated programs is becoming a part of our lives,
So programmers and end users alike need affordances for doing so!
Overview

LEAP:
Live Exploration of AI-Generated Code

ColDeco:
An End User Spreadsheet Inspection Tool for AI-Generated Code

Programmers

End Users
Overview

LEAP:
Live Exploration of AI-Generated Code

1. The Cost of Validation
2. LEAP demo
3. User Study

ColDeco:
An End User Spreadsheet Inspection Tool for AI-Generated Code
Overview

LEAP:
Live Exploration of AI-Generated Code

1. The Cost of Validation
2. LEAP demo
3. User Study

ColDeco:
An End User Spreadsheet Inspection Tool for AI-Generated Code
The Cost of Validation

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.

[Barke et al. 2023, Liang et al. 2023, Mozannar et al. 2022, Vaithilingam et al. 2022]
The Cost of Validation

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

LEAP:
Live Exploration of AI-Generated Code

1. The Cost of Validation
2. LEAP demo
3. User Study

ColDeco:
An End User Spreadsheet Inspection Tool for AI-Generated Code
Overview

LEAP:
Live Exploration of AI-Generated Code

1. The Cost of Validation
2. LEAP demo
3. User Study

ColDeco:
An End User Spreadsheet Inspection Tool for AI-Generated Code
User Study

How does Live Programming affect...

1. Code Correctness
2. Over-/Under-Reliance on AI
3. Cognitive Load

Between Subjects study:

17 Participants

2 Conditions:

1. AI
2. AI + LP

LP

NO-LP
RQ1: Correctness

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

6 no-LP vs 0 LP 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)

Live programming reduces over-/under-reliance on AI, by lowering the cost of validation.
Live programming significantly reduced the cognitive load of exploration for tasks amenable to validation by execution.
In Summary...

Live Programming is *not a panacea*. But!

It’s really powerful for reducing the *cost of validating* AI-generated programs.
Overview

LEAP:
Live Exploration of AI-Generated Code

1. The Cost of Validation
2. LEAP demo
3. User Study

ColDeco:
An End User Spreadsheet Inspection Tool for AI-Generated Code
Overview

LEAP:
Live Exploration of AI-Generated Code

1. The Cost of Validation
2. LEAP demo
3. User Study

ColDeco:
An End User Spreadsheet Inspection Tool for AI-Generated Code

1. End User Programming
2. ColDeco Example
3. Implementation
4. User Study
Overview

LEAP: Live Exploration of AI-Generated Code

1. The Cost of Validation
2. LEAP demo
3. User Study

ColDeco: An End User Spreadsheet Inspection Tool for AI-Generated Code

1. End User Programming
2. ColDeco Example
3. Implementation
4. User Study
End User Programming

Live Programming for free

End Users can’t always read the code

Participants described code examples as "overwhelming" (P8) and they "didn’t know where to start" (P3, P14). As a whole, participants wrestled with the perceived complexity of coding by adopting ad-hoc strategies to understand individual expressions and lines of code. [Lau et al. 2021]
Can we leverage familiar spreadsheet concepts for end user validation of AI-generated code?

- Helper Columns
- Filtering
  - Intermediate variables
  - Program Slicing!
Overview

LEAP:
Live Exploration of AI-Generated Code

1. The Cost of Validation
2. LEAP demo
3. User Study

ColDeco:
An End User Spreadsheet Inspection Tool for AI-Generated Code

1. End User Programming
2. ColDeco Example
3. Implementation
4. User Study
Create a column “Abbreviation” concatenating the first character of each part of the name

```python
df['Abbreviation'] = \\
    df['First Name'].str[0] + \\
    df['Middle Name'].str[0] + \\
    df['Last Name'].str[0]
```

* [Liu and Sarkar et al. 2023]
**ColDeco**

<table>
<thead>
<tr>
<th></th>
<th>DoB</th>
<th>text concatenation</th>
<th>1st letter of Last N</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11/5/1995</td>
<td>CM</td>
<td>F</td>
<td>CMF</td>
</tr>
<tr>
<td>4</td>
<td>6/21/1971</td>
<td>BH</td>
<td>B</td>
<td>BHB</td>
</tr>
<tr>
<td>5</td>
<td>9/11/1992</td>
<td>DB</td>
<td>M</td>
<td>DBM</td>
</tr>
<tr>
<td>6</td>
<td>5/24/1973</td>
<td>OJ</td>
<td>A</td>
<td>OJA</td>
</tr>
<tr>
<td>7</td>
<td>2/24/1997</td>
<td>AM</td>
<td>T</td>
<td>AMT</td>
</tr>
<tr>
<td>8</td>
<td>3/19/1986</td>
<td>AL</td>
<td>J</td>
<td>ALJ</td>
</tr>
<tr>
<td>9</td>
<td>6/3/1968</td>
<td>EMPTY</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>12/9/1966</td>
<td>AJ</td>
<td>S</td>
<td>AJS</td>
</tr>
</tbody>
</table>

**Helper columns**

**“Decomposed” Description**
ColDeco

<table>
<thead>
<tr>
<th></th>
<th>DoB</th>
<th>1st letter of First Na</th>
<th>1st letter of Middl</th>
<th>text concate</th>
<th>1st</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>11/5/1995</td>
<td>C</td>
<td>M</td>
<td>CM</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>6/21/1971</td>
<td>B</td>
<td>H</td>
<td>BH</td>
<td>B</td>
</tr>
<tr>
<td>5</td>
<td>9/11/1992</td>
<td>D</td>
<td>B</td>
<td>DB</td>
<td>M</td>
</tr>
<tr>
<td>6</td>
<td>5/24/1973</td>
<td>O</td>
<td>J</td>
<td>OJ</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>2/24/1997</td>
<td>A</td>
<td>M</td>
<td>AM</td>
<td>T</td>
</tr>
<tr>
<td>8</td>
<td>3/19/1986</td>
<td>A</td>
<td>L</td>
<td>AL</td>
<td>J</td>
</tr>
<tr>
<td>9</td>
<td>6/3/1968</td>
<td>W</td>
<td>EMPTY</td>
<td>EMPTY</td>
<td>S</td>
</tr>
<tr>
<td>10</td>
<td>12/9/1966</td>
<td>A</td>
<td>J</td>
<td>AJ</td>
<td>S</td>
</tr>
<tr>
<td>11</td>
<td>1/12/1989</td>
<td>A</td>
<td>A</td>
<td>AA</td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>12/6/1973</td>
<td>L</td>
<td>C</td>
<td>LC</td>
<td>W</td>
</tr>
</tbody>
</table>

One summary row per behavior of the code

Only referenced columns shown

Abbreviation: ('text concatenation' + '1st letter of Last Name')
1st letter of Last Name: (the first character from 'Last Name')

'1st letter of First Name' + '1st letter of Middle Name')
1st letter of Middle Name: (the first character from 'Middle Name')
1st letter of First Name: (the first character from 'First Name')
Overview

**LEAP:**
Live Exploration of AI-Generated Code

1. The Cost of Validation
2. LEAP demo
3. User Study

**ColDeco:**
An End User Spreadsheet Inspection Tool for AI-Generated Code

1. End User Programming
2. ColDeco Example
3. Implementation
4. User Study
Implementing Helper Columns

Goal:

Given a pandas programs of the form `df[<name>] = <expr>`, extract intermediate sub-exprs representing row-wise operations

Solution*:

1. Identify subexpressions that can be written as `Series` representing a column,
2. Assign them to new columns in the `Dataframe`, and
3. Replace the original subexpression with a column reference.

* Basically, A-Normal Form conversion for Dataframe programs.
Implementing Helper Columns

\[
\text{df['Abbreviation']} = \text{df['First Name'].str[0]} + \text{df['Last Name'].str[0]}
\]

\[
\text{Series} + \text{Series}
\]

\[
\text{df['$fresh1']} = \text{df['First Name'].str[0]}
\]
\[
\text{df['$fresh2']} = \text{df['Last Name'].str[0]}
\]
\[
\text{df['Abbreviation']} = \text{df['$fresh1']} + \text{df['$fresh2']}
\]
Implementing Helper Columns

df["Popular"] = df.apply(lambda x:
   "Yes" if x["votes"] > 10000 and x["vote_avg"] >= 8 else "No",
   axis=1)

df["$fresh1"] = df.apply(lambda x: x["votes"] > 10000, axis=1)

df["$fresh2"] = df.apply(lambda x: x["vote_avg"] >= 8, axis=1)

df["Popular"] = df.apply(lambda x: "Yes" if x["$fresh1"] and x["$fresh2"] else "No", axis=1)
Implementing Summary Rows

Dataflow analysis, program tracing, etc.?

\[
x := 5
\]
\[
y := x + 2
\]
\[
x := x + 1
\]
\[
y := x + 10
\]

Table Filtering: Predicates over the values!

<table>
<thead>
<tr>
<th>vote_avg</th>
<th>votes</th>
<th>votes &gt; 10k</th>
<th>Popular</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7954</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>8.4</td>
<td>18132</td>
<td>True</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>vote_avg</th>
<th>votes</th>
<th>votes &gt; 10k</th>
<th>Popular</th>
</tr>
</thead>
<tbody>
<tr>
<td>{positive}</td>
<td>{positive}</td>
<td>{isFalse}</td>
<td>{Enum[No]}</td>
</tr>
<tr>
<td>{positive}</td>
<td>{positive}</td>
<td>{isTrue}</td>
<td>{Enum[Yes]}</td>
</tr>
</tbody>
</table>
Implementing Summary Rows

<table>
<thead>
<tr>
<th>vote_avg</th>
<th>votes</th>
<th>Popular</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7954</td>
<td>No</td>
</tr>
<tr>
<td>8.4</td>
<td>18132</td>
<td>Yes</td>
</tr>
</tbody>
</table>
# Implementing Summary Rows

1. Expand *all* helper columns.

<table>
<thead>
<tr>
<th>vote_avg</th>
<th>votes</th>
<th>vote_avg &gt;= 8</th>
<th>votes &gt; 10000</th>
<th>Popular</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7954</td>
<td>True</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>8.4</td>
<td>18132</td>
<td>True</td>
<td>True</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Implementing Summary Rows

1. Expand *all* helper columns.
2. *Tag* the values in each column using a predetermined set of predicates:
   - a. \{positive, zero, negative\}
   - b. \{isTrue, isFalse\}
   - c. \{empty, nonEmpty\}
   - d. Enumeration Value (distinct string values)

<table>
<thead>
<tr>
<th>vote_avg</th>
<th>votes</th>
<th>vote_avg &gt;= 8</th>
<th>votes &gt; 10000</th>
<th>Popular</th>
</tr>
</thead>
<tbody>
<tr>
<td>{positive}</td>
<td>{pos...}</td>
<td>{isTrue}</td>
<td>{isFalse}</td>
<td>...</td>
</tr>
<tr>
<td>{positive}</td>
<td>{pos...}</td>
<td>{isTrue}</td>
<td>{isTrue}</td>
<td>{enum[Yes]}</td>
</tr>
</tbody>
</table>
Implementing Summary Rows

1. Expand all helper columns.
2. Tag the values in each column using a predetermined set of predicates:
   a. \{positive, zero, negative\}
   b. \{isTrue, isFalse\}
   c. \{empty, nonEmpty\}
   d. Enumeration Value (distinct string values)
3. Partition the rows based on the vector of tags.

<table>
<thead>
<tr>
<th>vote_avg</th>
<th>votes</th>
<th>vote_avg &gt;= 8</th>
<th>votes &gt; 10000</th>
<th>Popular</th>
</tr>
</thead>
<tbody>
<tr>
<td>{positive}</td>
<td>{pos...}</td>
<td>{isTrue}</td>
<td>{isFalse}</td>
<td>...</td>
</tr>
<tr>
<td>{enum[No]}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{positive}</td>
<td>{pos...}</td>
<td>{isTrue}</td>
<td>{isTrue}</td>
<td>{enum[Yes]}</td>
</tr>
</tbody>
</table>
Overview

LEAP:
Live Exploration of AI-Generated Code

1. The Cost of Validation
2. LEAP demo
3. User Study

ColDeco:
An End User Spreadsheet Inspection Tool for AI-Generated Code

1. End User Programming
2. ColDeco Example
3. Implementation
4. User Study
User Study

User study with 24 participants, solving 4 tasks:

**Does ColDeco enable code validation by end users?**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Mid Study</th>
<th>Post Study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Cost</td>
<td>20</td>
<td>14</td>
</tr>
<tr>
<td>Event Duration</td>
<td>23</td>
<td>9</td>
</tr>
<tr>
<td>Show Popularity</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>Price Rounding</td>
<td>9</td>
<td>6</td>
</tr>
</tbody>
</table>

**What are users’ impressions of ColDeco’s features?**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Mid Study</th>
<th>Post Study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Descriptions are Useful</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>Summary Rows are Useful</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>Helper Columns are Useful</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Tree View is Useful</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

Correct: Strongly Agree, Agree, Neutral
Incorrect: Disagree, Strongly Disagree

40
User Study

Helper Columns afford transparency:

“show-your-work button” (P19)

It makes the code "less like a black box" (P23)

Helping them "pinpoint exactly which part of the prompt is not working well" (P15)

ColDeco for Collaboration:

*Explain* their work to someone else (P11, P15)

Help with *understanding* complex formulas (P6, P19)

Automatically *document* spreadsheets (P6, P15)
In Summary...

Using **familiar concepts** can enable end users to validate code suggestions.

PL techniques can offer **new affordances**, even if the user doesn’t see the program!
Overview

LEAP:
Live Exploration of AI-Generated Code

Live Programming for Programmers

ColDeco:
An End User Spreadsheet Inspection Tool for AI-Generated Code

PL Techniques for End Users
References


J. T. Liang, C. Yang, and B. A. Myers, “Understanding the Usability of AI Programming Assistants.” 2023


N. Polikarpova, “How Programmers Interact with AI Assistants,” 2023

L. Chen, M. Zaharia, and J. Zou, “How is ChatGPT’s behavior changing over time?,” 2023


Bonus Slides
Code Generation in the Wild

Excel FlashFill

An End User Tool:
- Input-Output Examples
- Output program not shown

“It’s a great concept, but it can also lead to lots of bad data. [...] Be very careful. [...]”

John Walkenbach
(Cited in [Mayer 2015])
GitHub Copilot

A Developer Tool:

- Code Context + Natural Language
- Only output program is shown

Programmers using AI-generated code...
1. Significant time validating code suggestions,
2. Trouble evaluating code correctness, and
3. Under- and over-rely on AI code suggestions.
## Grounded Copilot

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

- **Pattern matching**
- **Validation**
- **Scope**
- **Mismatch**
- **Tolerance**
Participants

n = 17

Occupation:
15 academia
2 industry

Python Usage:
2 occasionally
8 regularly
7 almost every day
LEAP was more *usable* and more *useful*. 
### User Impressions

<table>
<thead>
<tr>
<th>Feature</th>
<th>Mid Study</th>
<th>Post Study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Descriptions are Useful</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>Summary Rows are Useful</td>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>Helper Columns are Useful</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>Tree View is Useful</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td><strong>Strongly Agree</strong></td>
<td><strong>8</strong></td>
<td><strong>16</strong></td>
</tr>
<tr>
<td><strong>Agree</strong></td>
<td><strong>6</strong></td>
<td><strong>7</strong></td>
</tr>
<tr>
<td><strong>Neutral</strong></td>
<td><strong>4</strong></td>
<td><strong>2</strong></td>
</tr>
<tr>
<td><strong>Disagree</strong></td>
<td><strong>1</strong></td>
<td><strong>2</strong></td>
</tr>
<tr>
<td><strong>Strongly Disagree</strong></td>
<td><strong>1</strong></td>
<td><strong>1</strong></td>
</tr>
</tbody>
</table>

Users liked ColDeco.
User Impressions

Helper Columns -

Summary Rows -

Descriptions -

[Diagram showing the distribution of user impressions across helper columns and summary rows, with numbers indicating the count for each category: 12, 7, 5 for helper columns, 5, 8, 11 for summary rows, and 11, 7, 6 for descriptions.]
Usability of Summary Rows

"I don’t really understand it, so I wanted to look at the table myself." (P6)

“It brings the different outcomes and behaviors to the front of the screen very quickly.” (P16)

"I think I didn’t understand summary rows before this [...] Maybe I got used to it because it’s my fourth time using this program" (P14)

Summary Rows had a steeper learning curve