how programmers interact with ai assistants

Nadia Polikarpova
DL4Code @ ICLR’23
with Shraddha Barke, Kasra Ferdowsifard, Lisa Huang, Michael B. James, and Sorin Lerner
the new era of programming

Github Copilot  
Chat GPT  
Amazon CodeWhisperer

and more...
the new era of programming

what do programmers use them for?

how do they express intent?

how do they validate suggestions?

how do they cope with errors?

Github Copilot
Chat GPT
Amazon CodeWhisperer
and more...
this talk

grounded copilot

grounded theory of AI-assisted programming

[OOPSLA'23]

leap

helping programmers validate AI-generated code

[under review]
this talk

grounded

method
copilot
theory
recommenda
grounded theory
ions of AI-assisted programming
this talk

grounded copilot

grounded theory of AI-assisted programming

1. method
grounded theory

data collection

data interpretation

theory development
grounded theory

programming session + interview

qualitative coding

theory development
tasks

chat server
business logic of a chat app
Python/Rust

chat client
networking + custom crypto API
Python/Rust

benford’s law
familiar algorithm + matplotlib
Rust + Python

string rewriting
competition task, easy to test
Python/Rust/Haskell/Java
participants

n = 20

occupation:
15 academia / 5 industry

language proficiency:
occasional / regular / professional

prior Copilot experience:
9 no / 11 yes
this talk

grounded copilot

grounded theory of AI-assisted programming

1. method
2. theory
3. recommendations
this talk

grounded copilot

grounded theory of AI-assisted programming

2. theory
programming, fast and slow

**acceleration**

*autocomplete++*

programmer has a plan
copilot helps them get there faster

**VS**

**exploration**

*StackOverflow++*

programmer is lost
copilot suggests potential solutions
programming, fast and slow

**acceleration**

autocomplete++

programmer has a plan
copilot helps them get there faster
acceleration: example

programmer: broke down the task, has a good idea for this function

```
# rules are formatted like:
# AB => C

def parse_input(filename):
    with open(filename) as f:
        template, rules = f.read().split("\n\n")
        for rule in rules:
            rule_parts = |
```
acceleration: example

```python
# rules are formatted like:
# AB => C

def parse_input(filename):
    with open(filename) as f:
        template, rules = f.read().split("\n\n")
        for rule in rules:
            rule_parts = rule.split(" => ")
```

**programmer:** “pattern-matches” suggestion against expectations; quickly accepts, without leaving flow

**copilot:** auto-completes current logical unit (line of code)
programming, fast and slow

**acceleration**
- autocomplete++
  - programmer has a plan
  - copilot helps them get there faster

**VS**

**exploration**
- StackOverflow++
  - programmer is lost
  - copilot suggests potential solutions
exploration: example

programmer: unfamiliar with matplotlib

intentionally prompts with a comment; invokes side panel
exploration: example

programmer: carefully examines suggestions; compares to gauge confidence in API usage

copilot suggests multiple alternatives

might cherry-pick parts from different suggestions

validates code by executing or consulting documentation
acceleration vs exploration

Interaction modes based on participant's expertise

- Occasional (n=9)
  - Acceleration: 2 min
  - Exploration: 14 min

- Regular (n=2)
  - Acceleration: 1 min
  - Exploration: 15 min

- Professional (n=11)
  - Acceleration: 8 min
  - Exploration: 8 min

Interaction modes based on participant's Prior Copilot Usage (PCU)

- PCU (n=9)
  - Acceleration: 8 min
  - Exploration: 8 min

- No-PCU (n=11)
  - Acceleration: 6 min
  - Exploration: 10 min
acceleration                vs                 exploration

unintentional

“pattern matching”

unit of focus
(sub-expression / statement)

unwilling to edit

prompting

validation

scope

mismatch
tolerance

intentional with comments / invoke side panel

explicit validation via examination / execution / documentation

entire function + multiple alternatives

willing to edit / debug / “rip apart” / cherry-pick
this talk

grounded copilot

grounded theory of AI-assisted programming

1. method
2. theory
3. recommendations
this talk

grounded copilot

grounded theory of AI-assisted programming

3. recommendations
acceleration vs exploration

unexpected suggestions  
break flow

main challenges
acceleration vs exploration

unexpected suggestions break flow

main challenges

suggestions hard to validate & debug

multiple suggestions hard to distinguish
acceleration vs exploration

unexpected suggestions break flow

1. mode awareness

suggestions hard to validate & debug
multiple suggestions hard to distinguish

2. simplify validation

3. better support for comparing alternative suggestions
this talk

grounded copilot

grounded theory of AI-assisted programming

leap

helping programmers validate AI-generated code
this talk

idea: simplify validation using live programming (continuous display of runtime values)

live exploration of ai-generated programs

helping programmers validate AI-generated code
this talk

1. demo
2. study
3. findings

leap

helping programmers validate AI-generated code
this talk

1. demo

leap

helping programmers validate AI-generated code
```python
import matplotlib.pyplot as plt
import pandas as pd
import io

def create_plot(df):
    # create a box plot where each box refers to a group
    
    print(df)
    plt.boxplot([df[df['group'] == 'control']['time'],
                 df[df['group'] == 'experiment']['time']])

csv = '''
ID,group,time,success
P1,control,18.6,1
P2,control,15.42,1
P3,control,25.55,0
P4,control,12.56,0
P5,control,8.67,1
P6,experiment,7.31,0
P7,experiment,9.66,0
P8,experiment,13.64,1
P9,experiment,14.92,1
P10,experiment,18.47,1
...'''

df = pd.read_csv(io.StringIO(csv))

plot = create_plot(df)
```

**Suggestion 1**

```
fig, ax = plt.subplots()
ax.boxplot([df[df.group == 'control']['time'],
            df[df.group == 'experiment']['time']],
           labels=['control', 'experiment'])
ax.set_title('time to success')
ax.set_xlabel('time (s)')
plt.show()
```

**Suggestion 2**

```
# the x-axis is the group
# the y-axis is the time
# the color of the box depends on the success
```

**Suggestion 3**

```
fig, ax = plt.subplots()
df.boxplot(column="time", by="group", ax=ax)
```

**Suggestion 4**

```
plt.boxplot([df[df['group'] == 'control']['time'],
             df[df['group'] == 'experiment']['time']])
```
this talk

1. demo
2. study
3. findings

leap
helping programmers validate AI-generated code
this talk

2. study

leap

helping programmers validate AI-generated code
experimental conditions

no-PB

AI suggestions + terminal

PB

AI suggestions + projection boxes
research questions

how does live programming affect...

1. code correctness
2. over- / under-reliance on AI
3. cognitive load
4. user impressions
tasks

API-heavy

pandas
- clean dataframe and compute stats using pandas

algorithmic

bigrams
- find most frequent bigram in a string

open prompt

box plot
- overlay scatter plot over boxplot using matplotlib

string rewriting
- parse rewrite rules and apply to string

fixed prompt

multiple correct suggestions

no correct suggestions
participants

n = 17

occupation:
15 academia / 2 industry

Python usage:
2 occasionally / 
8 regularly / 
7 almost every day
this talk

1. demo
2. study
3. findings

leap
helping programmers validate AI-generated code
this talk

3. findings

leap

helping programmers validate AI-generated code
rq1: correctness

leap helps validate suggestions (but does not help fix incorrect ones)
rq2: over-/under-reliance

6 no-PB vs 0 PB participants mid-judged correctness of their solution by lowering the cost of validation, leap reduces over-/under-reliance on AI
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)

6 no-PB vs 0 PB participants **mid-judged** correctness of their solution by lowering the cost of validation, leap reduces over-/under-reliance on AI
rq3: cognitive load

NASA TLX cognitive load metrics on Pandas

leap significantly reduced cognitive load of exploring AI suggestions on tasks amenable to validation by execution
rq3: user impressions

"Being able to preview, edit, and look at the projection boxes before accepting a snippet was very helpful when choosing between multiple suggestions." (P1)

users found leap more usable and useful
this talk

grounded
copilot

grounded theory
of AI-assisted programming

[OOPSLA'23]

leap

helping programmers
validate AI-generated code

[under review]