how programmers interact with ai assistants

Nadia Polikarpova

DL4Code @ ICLR'23

with Shraddha Barke, Kasra Ferdowsifard, Lisa Huang, Michael B. James, and Sorin Lerner

UC San Diego
COMPUTER SCIENCE & ENGINEERING

the new era of programming

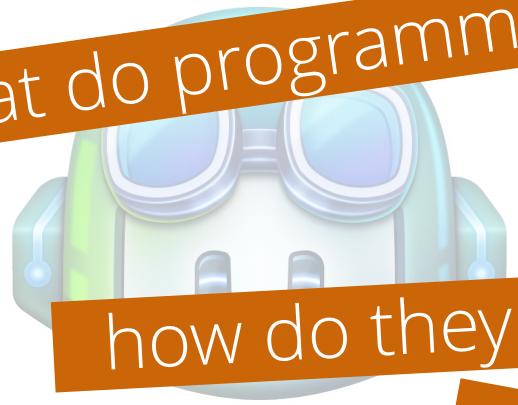
Github Copilot

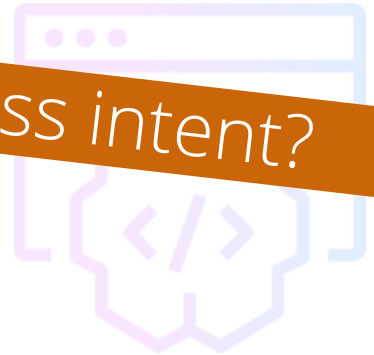
Chat GPT

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 G

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 copilot

grounded theory
of AI-assisted programming

1. method
2. theory
3. recommendations

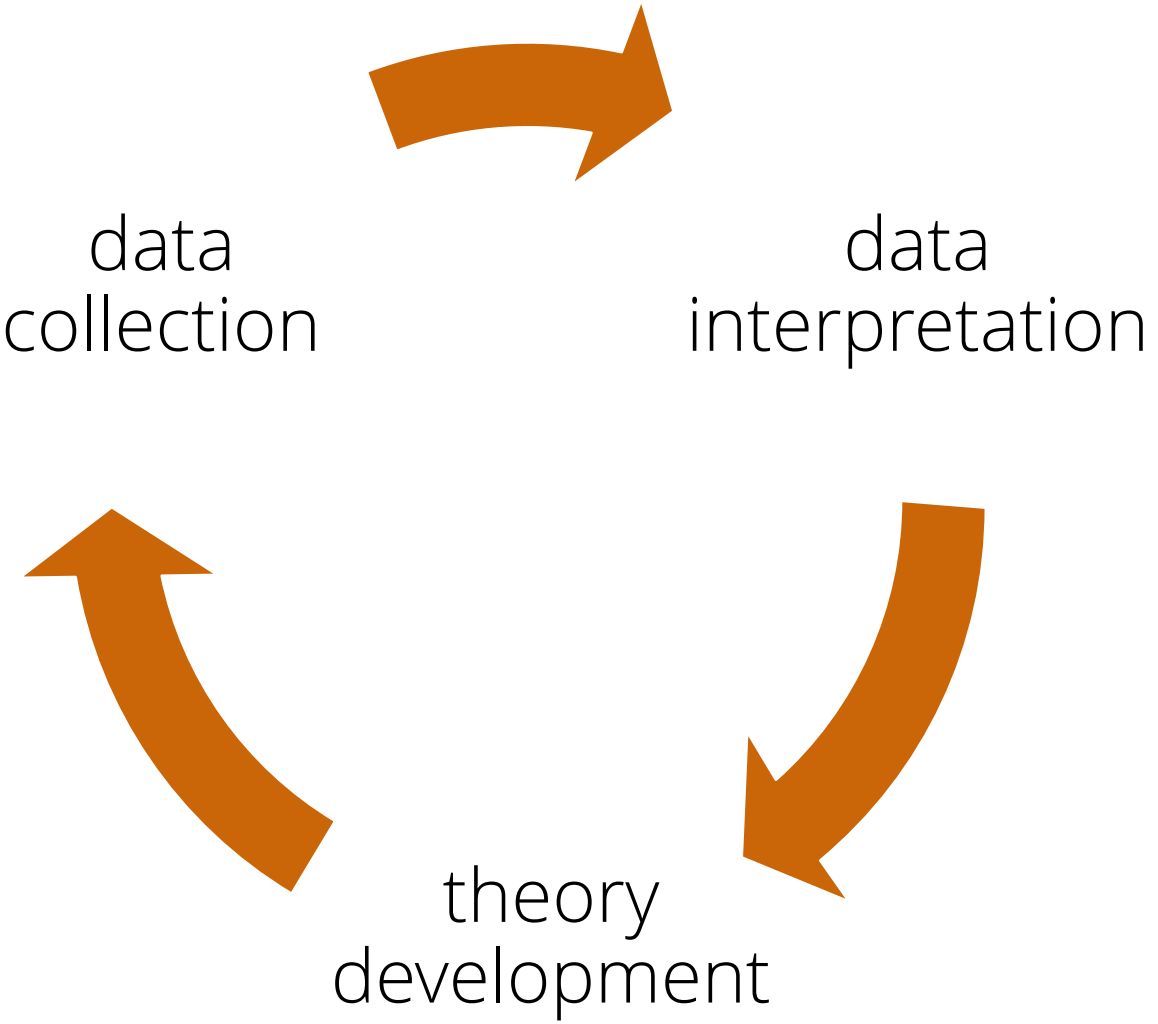
this talk

grounded copilot

grounded theory
of AI-assisted programming

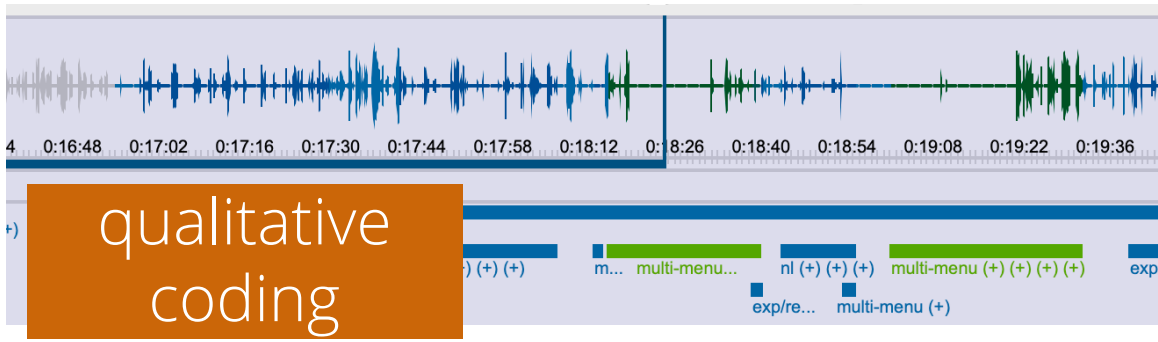
1. method

grounded theory



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

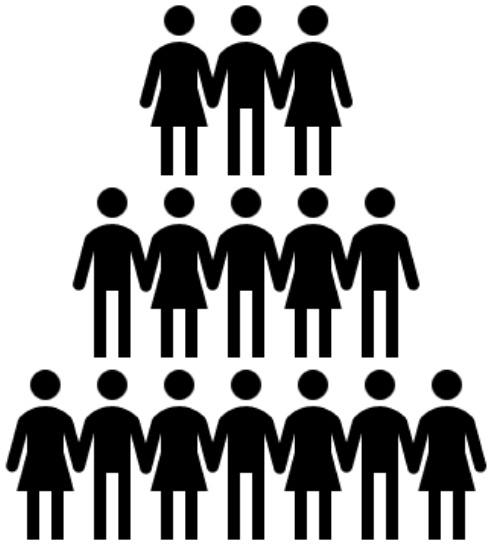
Rust + Python

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 =
```

pauses
(unintentional prompting)

acceleration: example

programmer: "pattern-matches" suggestion against expectations; quickly accepts, without leaving flow

```
# 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(" => ")
```

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

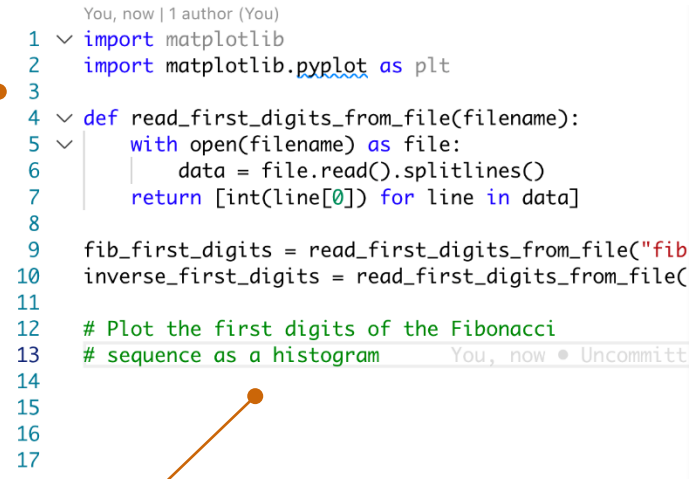
exploration

StackOverflow++

programmer is lost
copilot suggests potential solutions

exploration: example

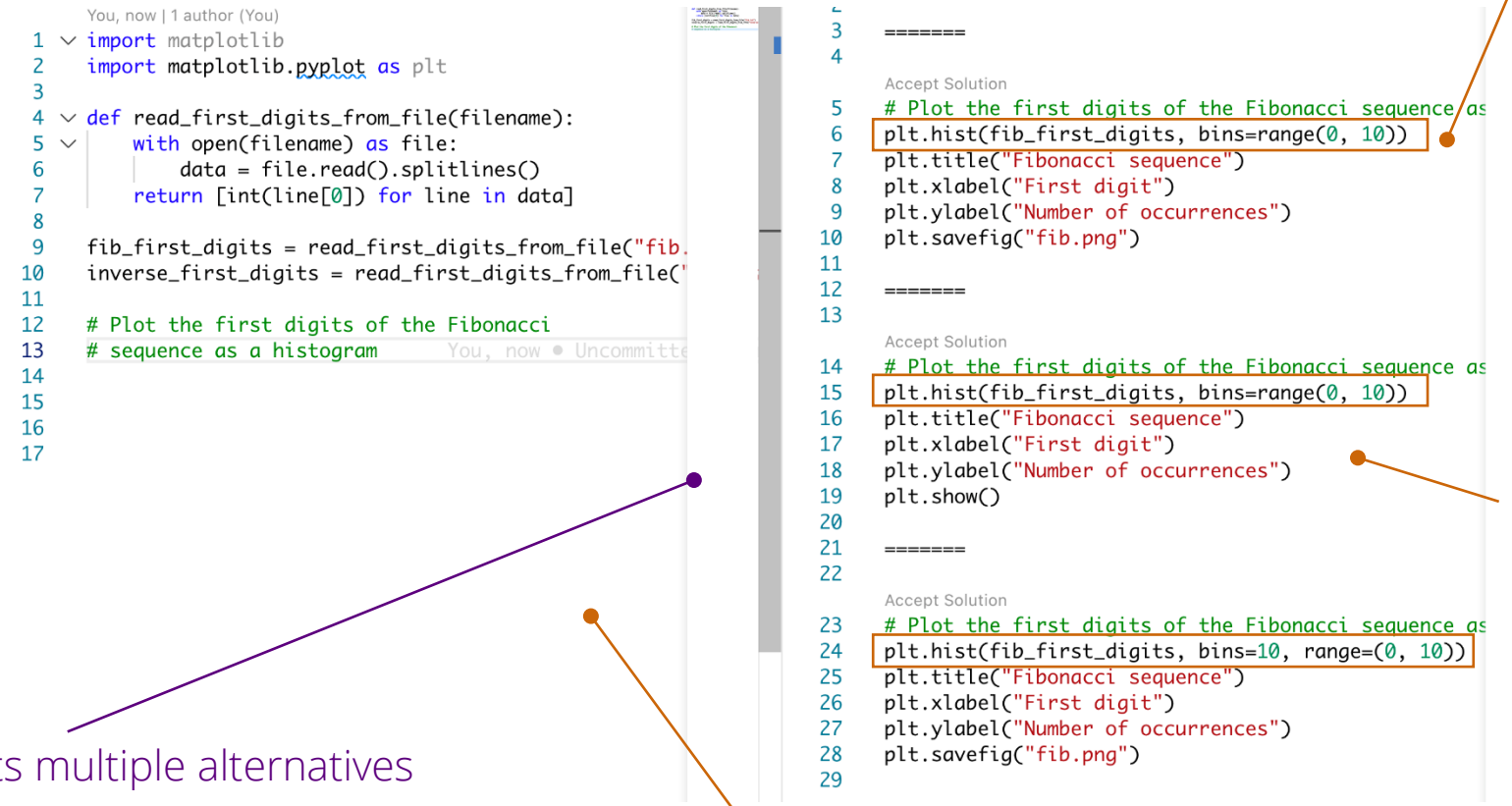
programmer:
unfamiliar with matplotlib



intentionally prompts
with a comment;
invokes side panel

exploration: example

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



```
You, now | 1 author (You)
import matplotlib
import matplotlib.pyplot as plt

def read_first_digits_from_file(filename):
    with open(filename) as file:
        data = file.read().splitlines()
    return [int(line[0]) for line in data]

fib_first_digits = read_first_digits_from_file("fib.
inverse_first_digits = read_first_digits_from_file("

# Plot the first digits of the Fibonacci
# sequence as a histogram      You, now • Uncommitte
```

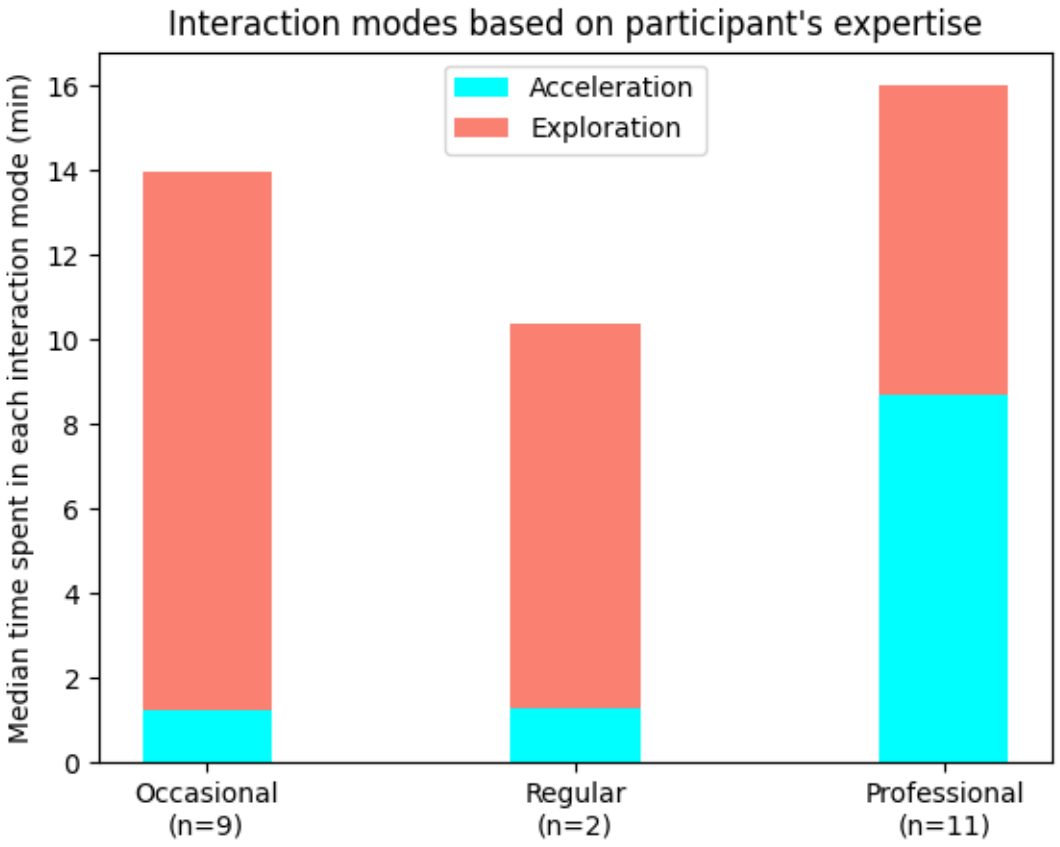
```
=======

Accept Solution
# Plot the first digits of the Fibonacci sequence as
plt.hist(fib_first_digits, bins=range(0, 10))
plt.title("Fibonacci sequence")
plt.xlabel("First digit")
plt.ylabel("Number of occurrences")
plt.savefig("fib.png")

=======

Accept Solution
# Plot the first digits of the Fibonacci sequence as
plt.hist(fib_first_digits, bins=range(0, 10))
plt.title("Fibonacci sequence")
plt.xlabel("First digit")
plt.ylabel("Number of occurrences")
plt.show()

=======

Accept Solution
# Plot the first digits of the Fibonacci sequence as
plt.hist(fib_first_digits, bins=10, range=(0, 10))
plt.title("Fibonacci sequence")
plt.xlabel("First digit")
plt.ylabel("Number of occurrences")
plt.savefig("fib.png")
```

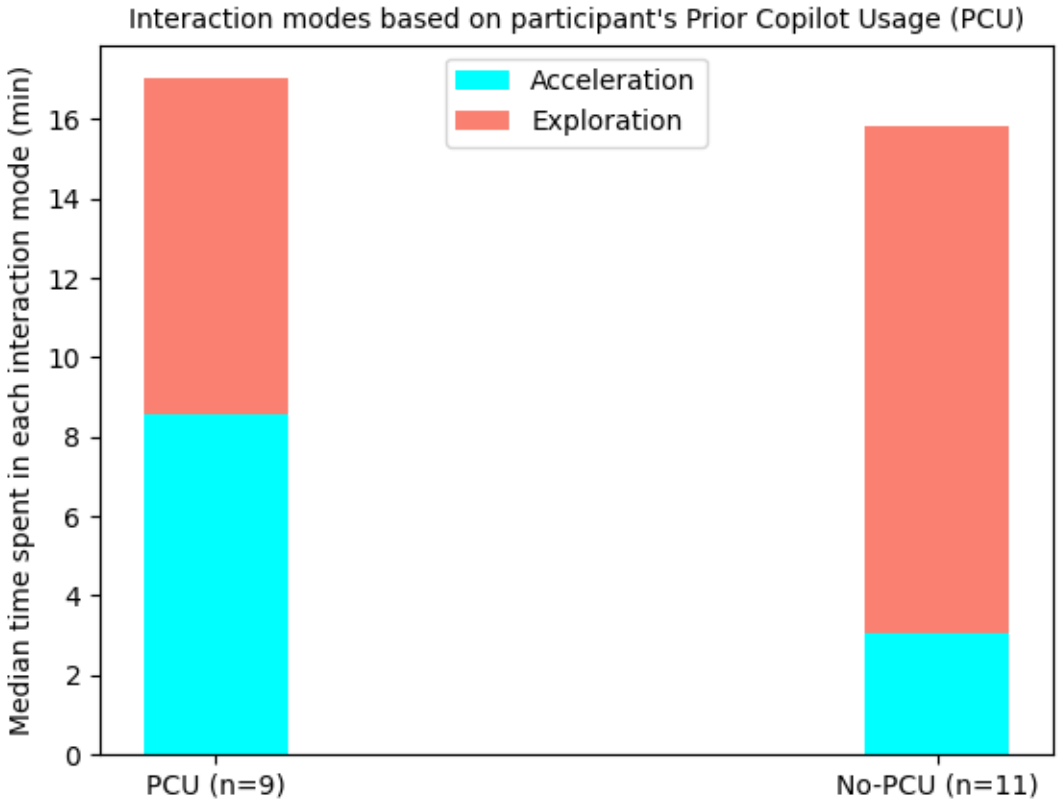
might cherry-pick parts from different suggestions

copilot suggests multiple alternatives

validates code by executing or consulting documentation

acceleration vs exploration





acceleration	vs	exploration
unintentional	prompting	intentional with comments / invoke side panel
“pattern matching”	validation	explicit validation via examination / execution / documentation
unit of focus (sub-expression / statement)	scope	entire function + multiple alternatives
unwilling to edit	mismatch tolerance	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

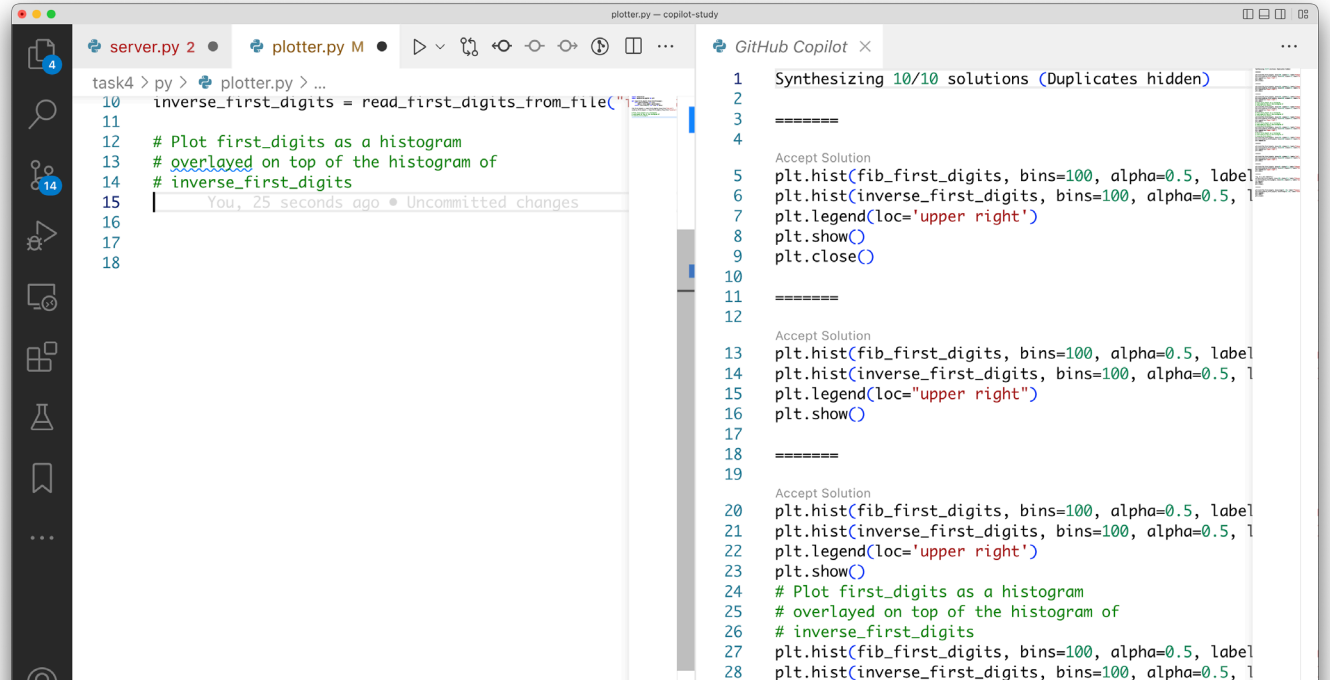
acceleration vs exploration

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)

```

A

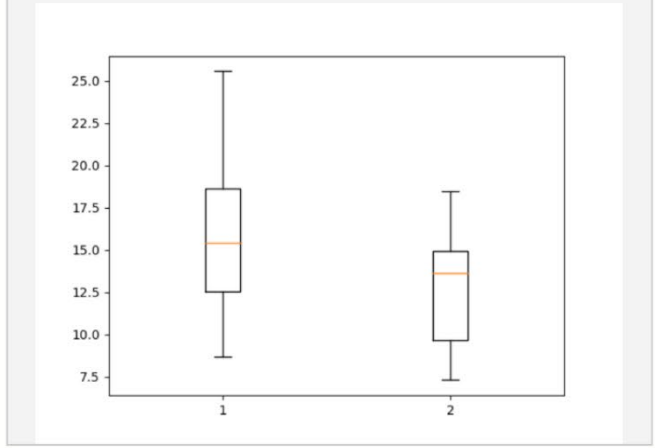
D

df

	ID	group	time	success
0	P1	control	18.60	1
1	P2	control	15.42	1
2	P3	control	25.55	0
3	P4	control	12.56	0
4	P5	control	8.67	1
5	P6	experiment	7.31	0
6	P7	experiment	9.66	0
7	P8	experiment	13.64	1
8	P9	experiment	14.92	1
9	P10	experiment	18.47	1

E

Plot



'\nID,group,time,success\nP1,control,18.6,1\nP2,control,15.42,1\nP3,control,25.55,0\

df

	ID	group	time	success
0	P1	control	18.60	1
1	P2	control	15.42	1
2	P3	control	25.55	0
3	P4	control	12.56	0
4	P5	control	8.67	1

C

B

Suggestion 1

Preview

```python
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_ylabel('time (s)')
plt.show()
```

Suggestion 2

Preview

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

Suggestion 3

Preview

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

Suggestion 4

Preview

```python
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

algorithmic

multiple correct suggestions

pandas

clean dataframe and compute stats using pandas

no correct suggestions

bigrams

find most frequent bigram in a string

fixed prompt

open prompt

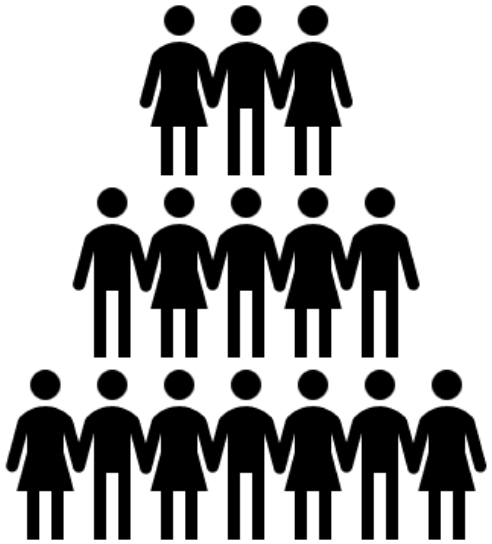
box plot

overlay scatter plot over boxplot using matplotlib

string rewriting

parse rewrite rules and apply to string

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

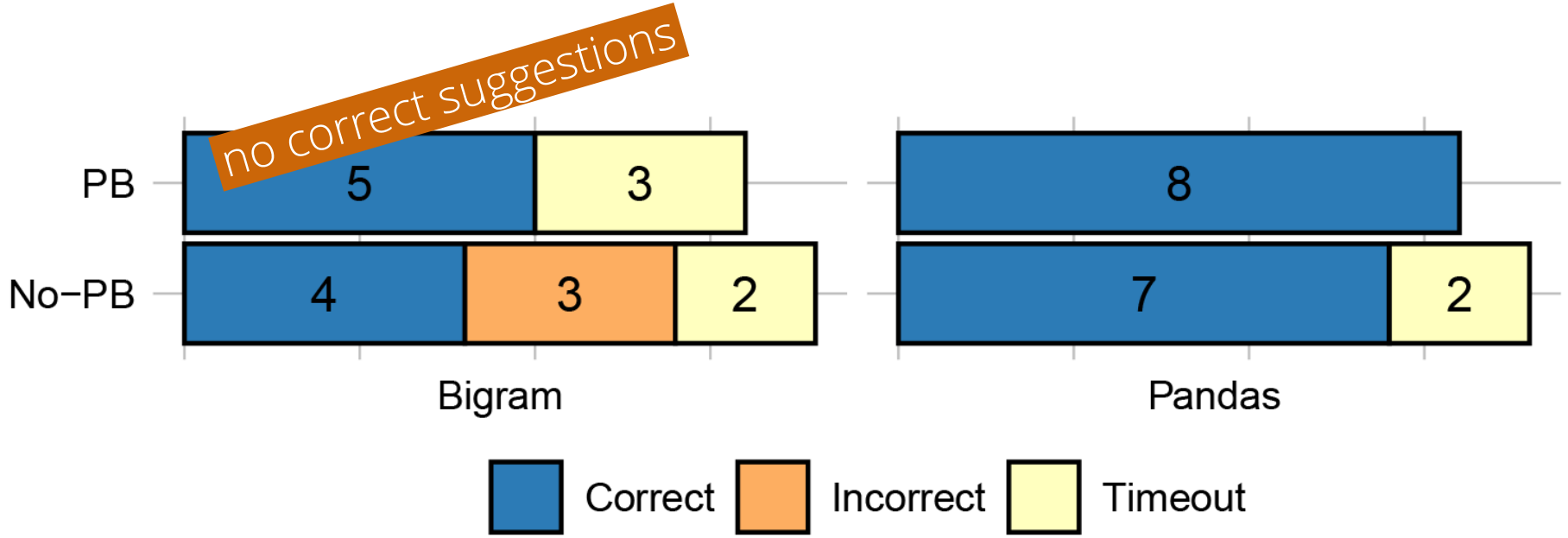
this talk

3. findings

leap

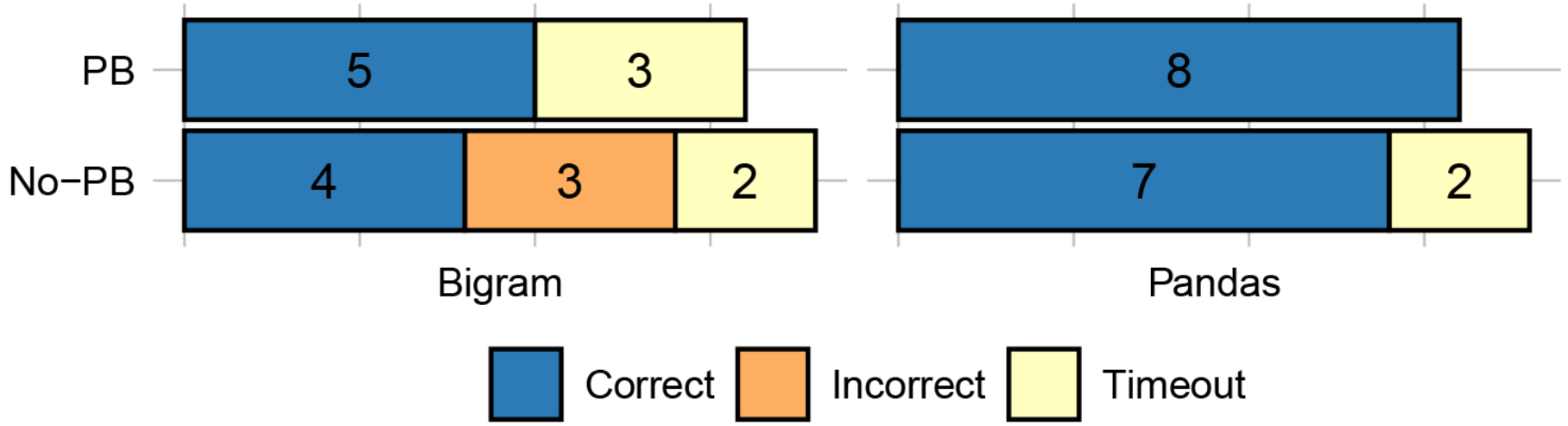
helping programmers
validate AI-generated code

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)

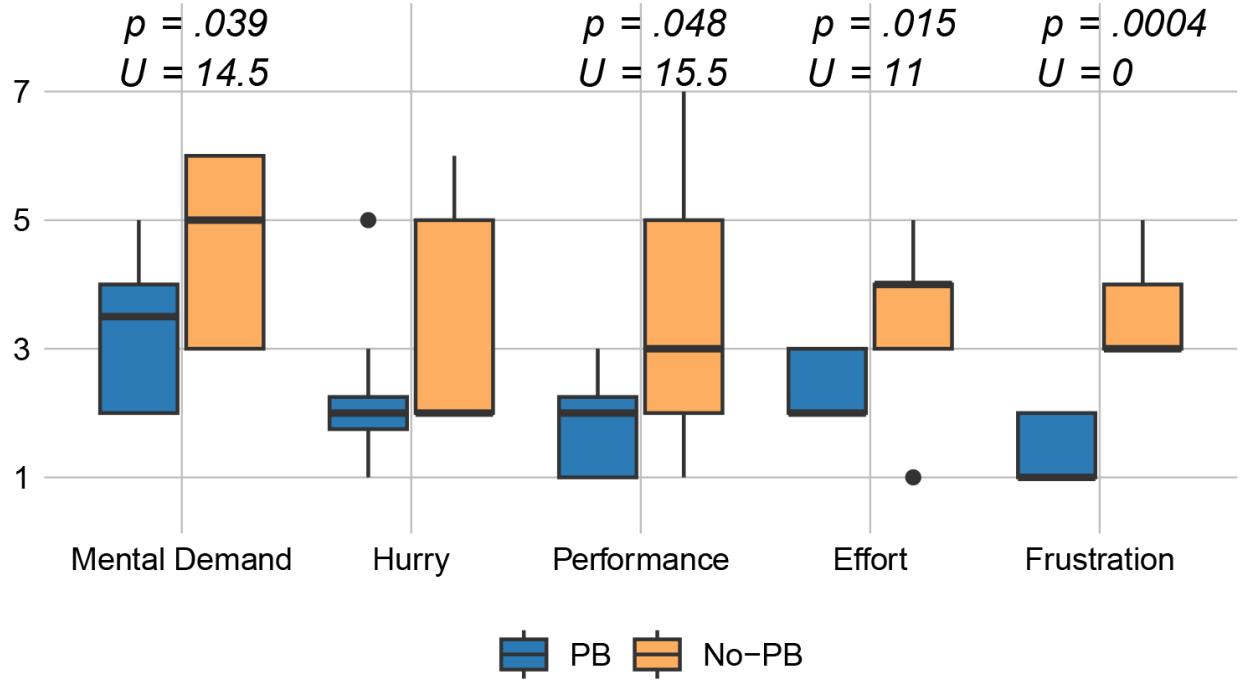
"it **saved me the effort** of writing multiple print statements." (P1)

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

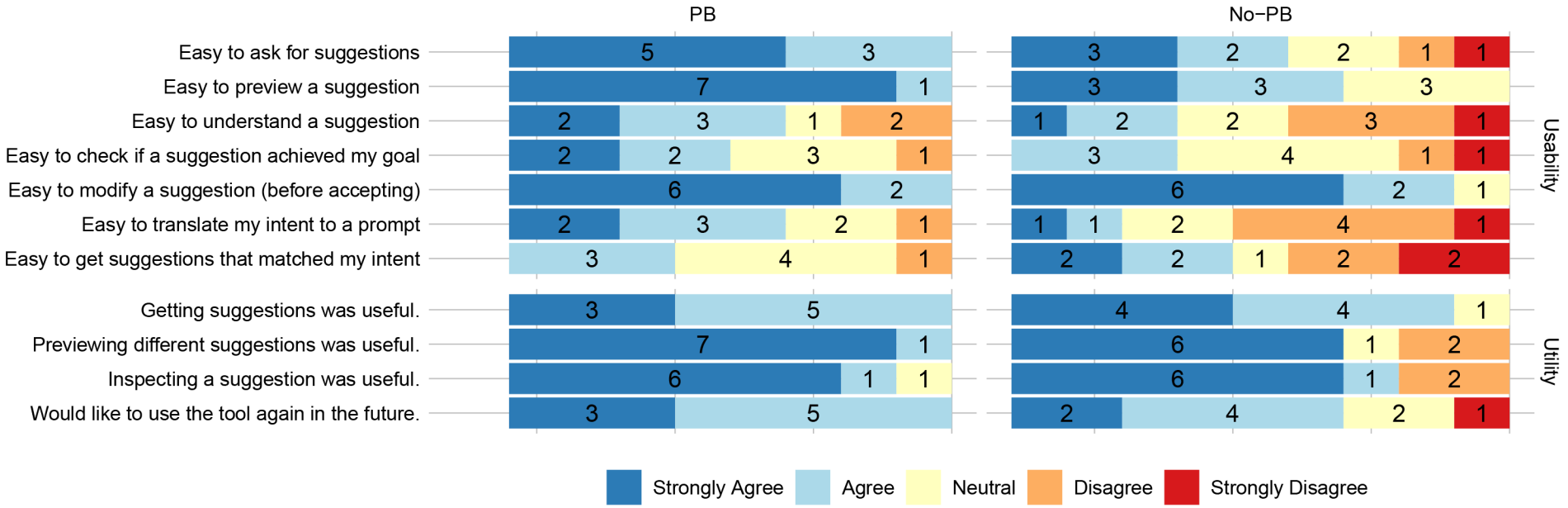
rq3: cognitive load

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]