how programmers interact with ai assistants

Nadia Polikarpova

UPenn Seminar, November 2023

UC San Diego
COMPUTER SCIENCE & ENGINEERING

the new era of programming

Github Copilot

Chat GPT

Amazon CodeWhisperer

and more...

this talk

I.
how do
programmers
use existing
tools?

II.
how can
we make
the tools
more usable?

this talk

I.

grounded copilot:

grounded theory
of AI-assisted programming

other studies

of existing tools

our work

II.

leap:

validating AI-generated code
with live programming

other designs

for new tools

how do programmers use existing tools?

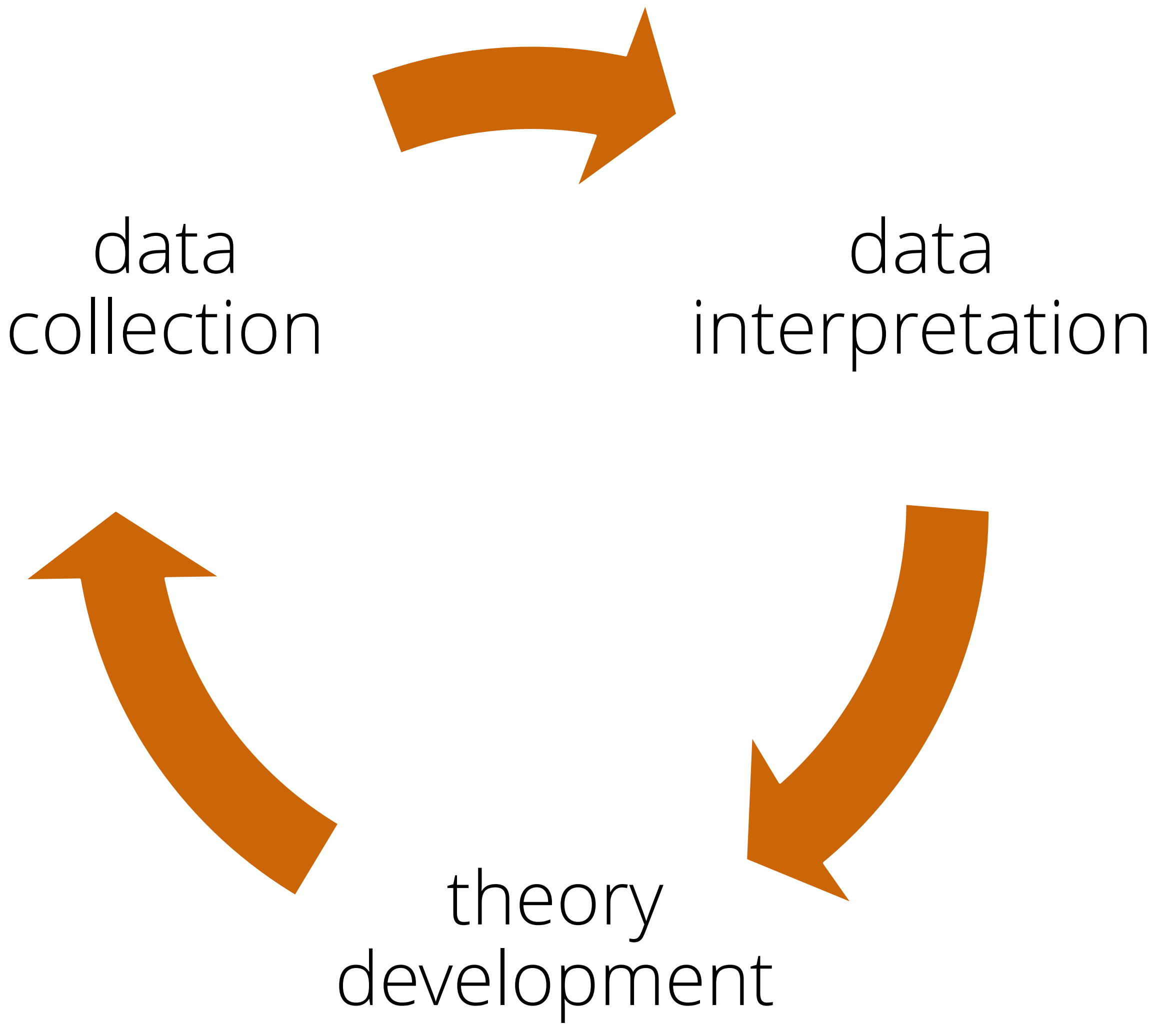
grounded copilot:

grounded theory
of AI-assisted programming

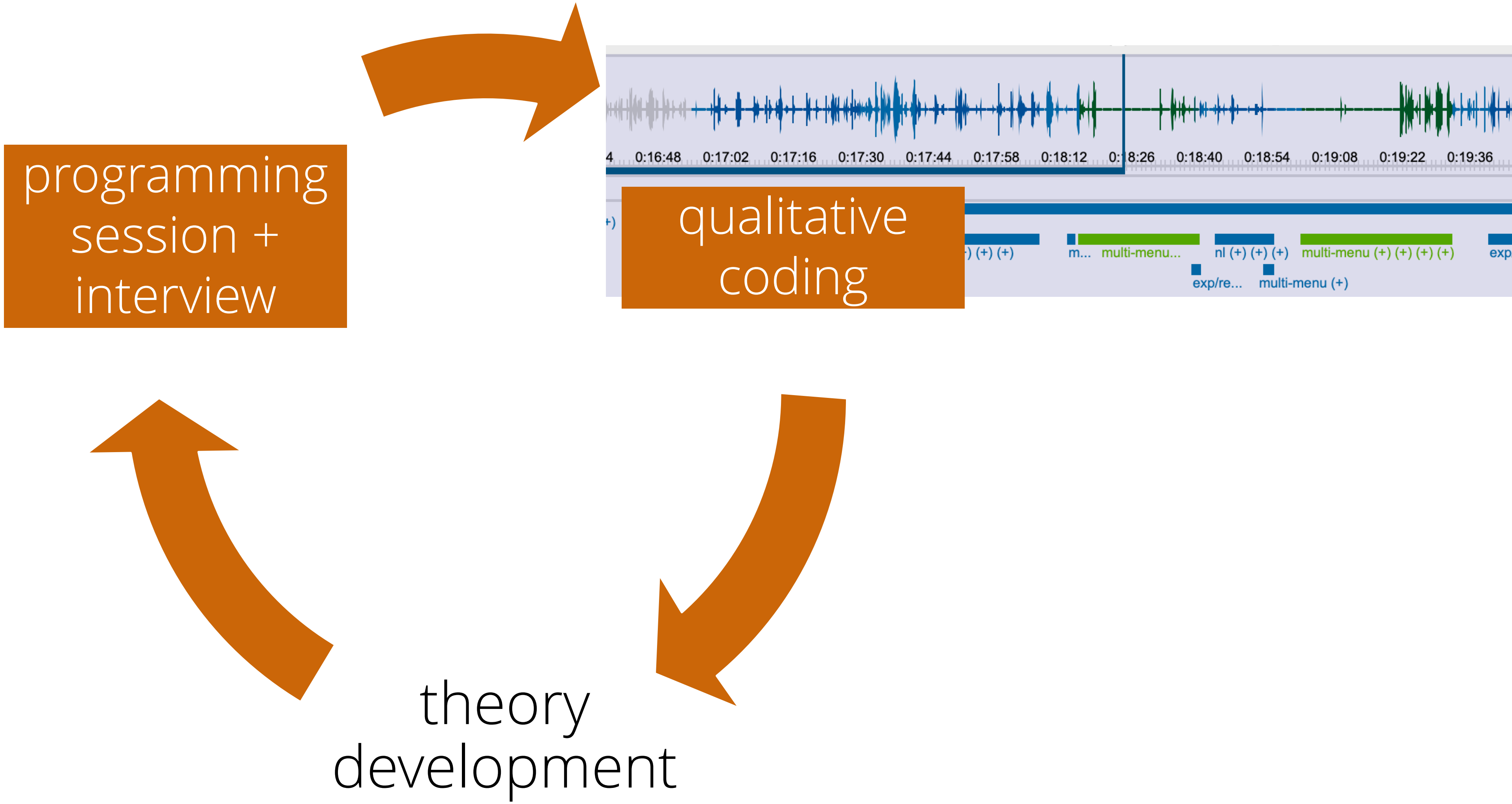
[Barke et al, OOPSLA'23]

distinguished paper

grounded theory



grounded theory



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

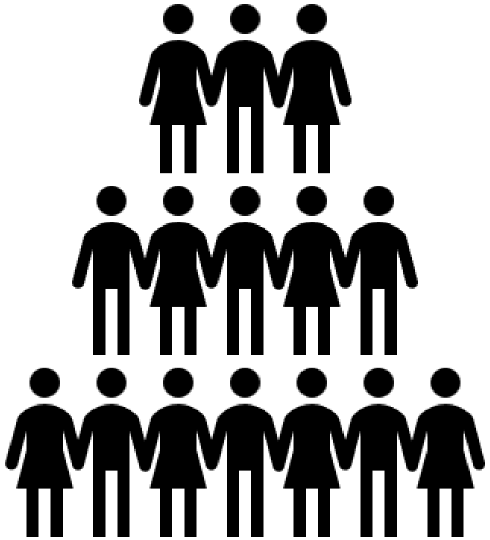
Rust + Python

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

programming, fast and slow

acceleration **VS** **exploration**

autocomplete++

programmer has a plan
copilot helps them get there faster

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

```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 = 
```

pauses
(unintentional prompting)

acceleration: example

```
# 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 VS exploration

acceleration	exploration
autocomplete++	StackOverflow++
programmer has a plan copilot helps them get there faster	programmer is lost copilot suggests potential solutions

exploration: example

programmer:
unfamiliar with matplotlib

```
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 • Uncommitt
```

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)

```python
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
```

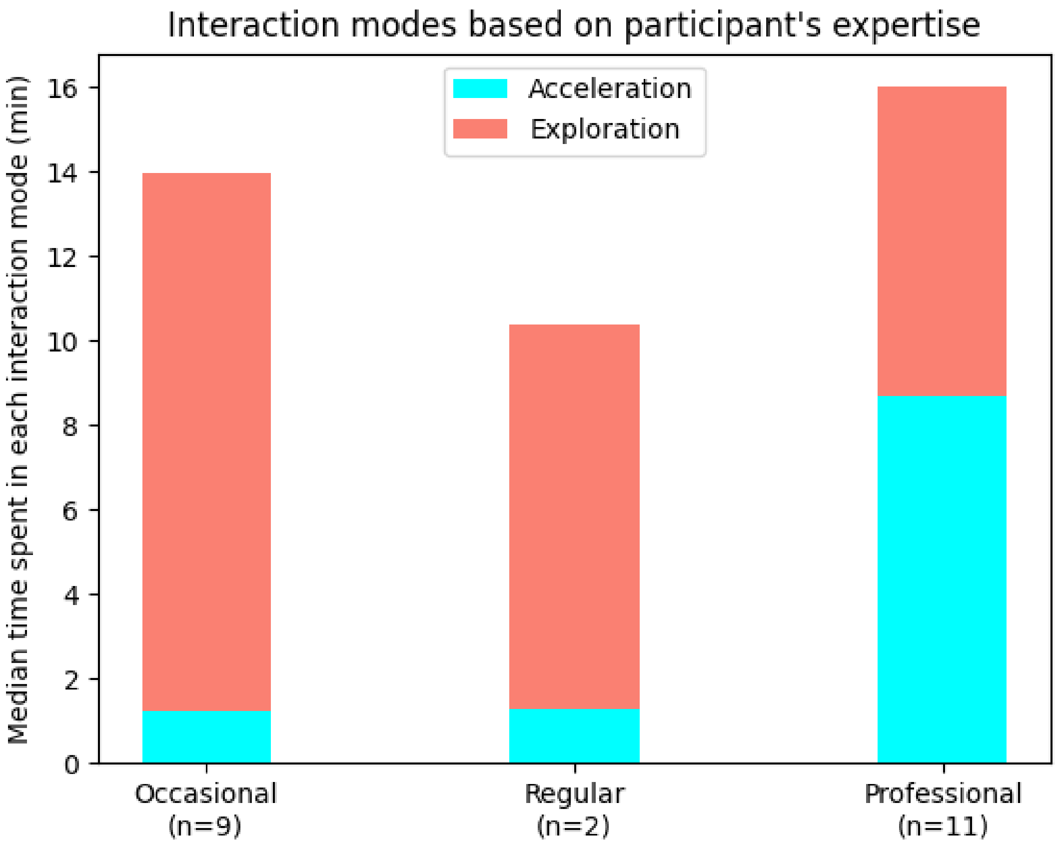
```
=======

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")
```

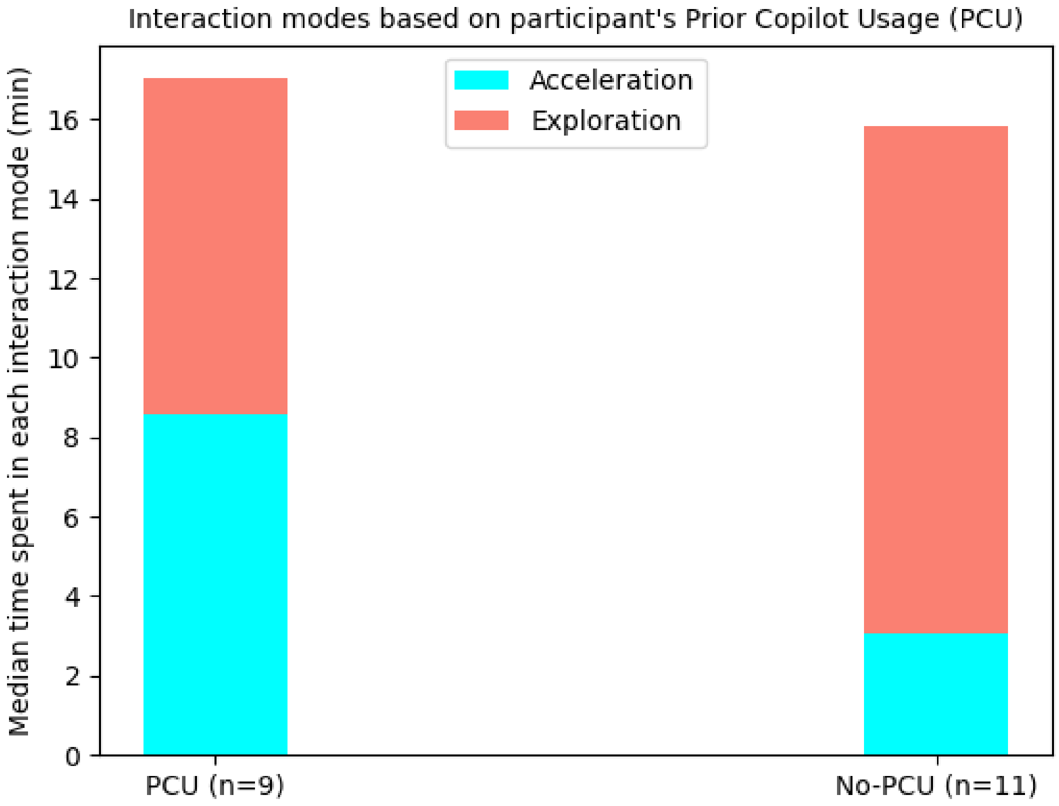
might cherry-pick parts from different suggestions

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

how do programmers use existing tools?

I.

grounded copilot:
grounded theory
of AI-assisted programming

other studies
of existing tools

how do programmers use existing tools?

other studies
of existing tools

[Ziegler et al, MAPS'22]

[Vaithilingam et al, CHI EA'22]

[Mozannar et al, arXiv'22]

[Peng et al, arXiv'23]

[Liang et al, arXiv'23]

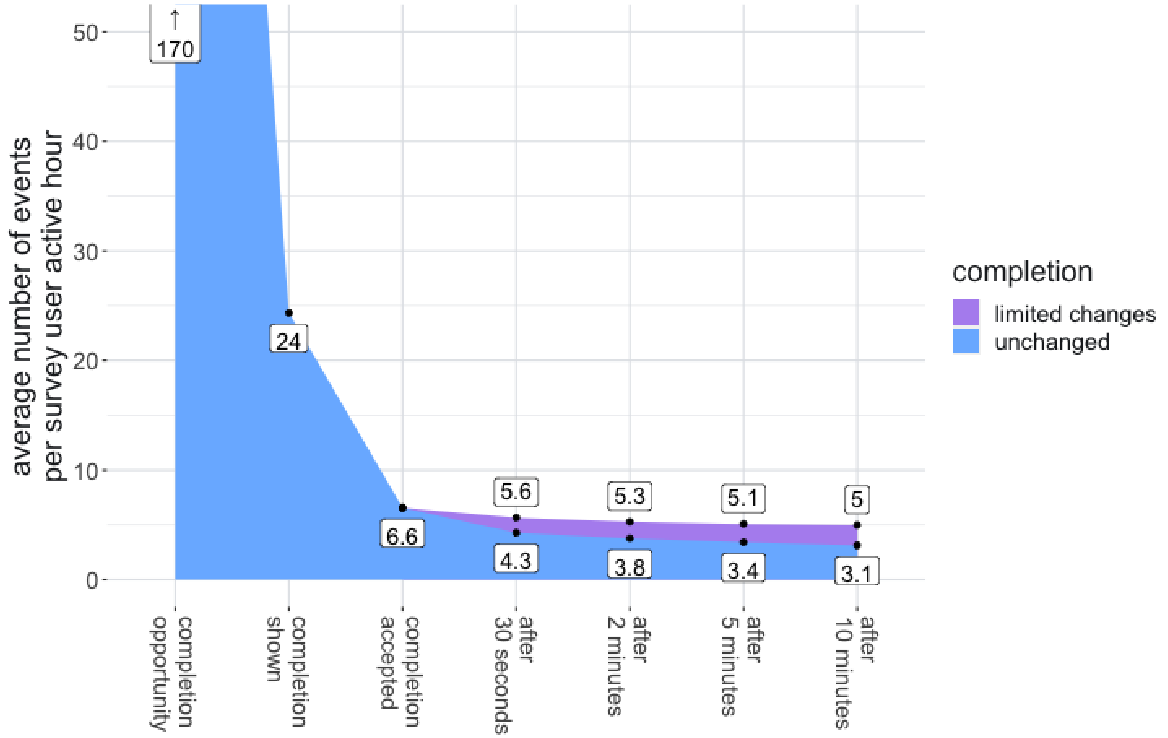
productivity

[Ziegler et al, MAPS'22]

- analysis of 2531 survey responses + telemetry from Copilot
- measure perceived productivity

results:

- programmers perceive themselves more productive
- correlated with acceptance rate
- average acceptance rate ~30%



productivity (objective)

[Vaithilingam et al, CHI EA'22]

- 24 participants (mostly students)
- 3 programming tasks (easy to hard)
- within subjects
- Copilot vs IntelliSense

results:

- no improvement in task completion rate or time
- but most participants preferred Copilot

[Peng et al, arXiv'23]

- 95 developers recruited through UpWork
- task: HTTP server in JavaScript
- between subjects
- Copilot vs regular IDE

results:

- completion time improved by 55.8%
- rate also improved but not significantly

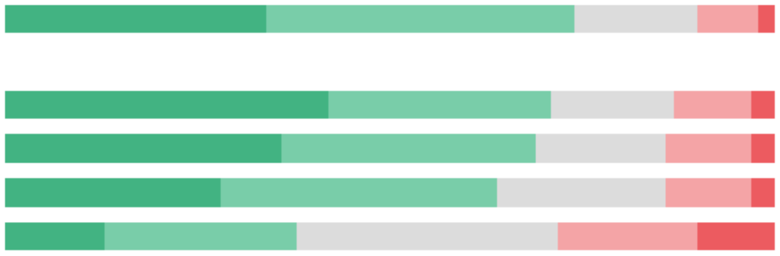
usage patterns

[Liang et al, arXiv'23]

- survey of 410 developers using Copilot / ChatGPT / CodeWhisperer /etc
- quantitative data to complement our findings
  - for example: prevalence of validation strategies related to their time cost

C. Methods of evaluating code output

S13	Quickly checking the generated code for specific keywords or logic structures	*74%*	*10%*
S14	Compilers, type checkers, in-IDE syntax checkers, or linters	*71%*	*14%*
S15	Executing the generated code	*69%*	*14%*
S16	Examining details of the generated code's logic in depth	*64%*	*15%*
S17	Consulting API documentation	*38%*	*28%*

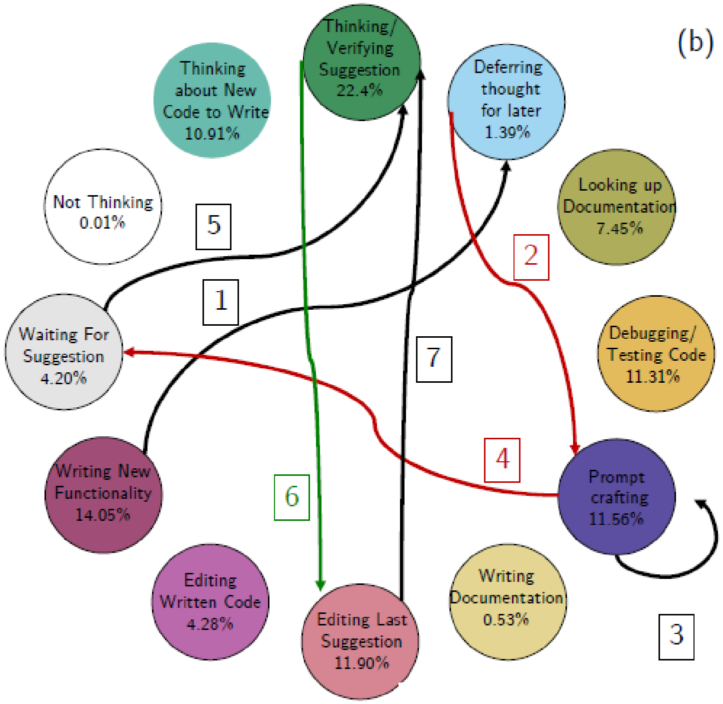


- extensive list of requested features

usage patterns

[Mozannar et al, arXiv'22]

- observed 21 programmers using Copilot
- developed the CUPS taxonomy of user states
  - refinement of our two modes
- collected stats on prevalence of states and transitions
  - users spend the most time (22.4%) validating suggestions
  - users often validate after "accepting" (e.g. to see syntax highlighting)



this talk

I.
how do
programmers
use existing
tools?

II.
how can
we make
the tools
more usable?

how can we make the tools more usable?

1. help with validation
2. eliminate distractions
3. give user more control
4. navigating solution spaces

how can we make the tools more usable?

1. help with validation

leap:

validating AI-generated code
with live programming

the validation challenge

"In the context of Copilot, there is a shift from writing code to understanding code"

Taking Flight with Copilot, ACM Queue, Dec 22

- validation is hard
  - [Vaithilingam et al] observed 8 cases of over-reliance: bugs due to skipped validation
- validation is a bottleneck
  - single most prevalent activity according to [Mozannar et al]
- prevalence of a validation strategy depends on its cost [Liang et al]

to help with validation, we need to lower its cost

leap

lowers the cost of validation by execution
using live programming

demo

user study

no-LP

AI suggestions
+
terminal

LP

AI suggestions
+
live programming

research questions

how does live programming affect…

1. over- / under-reliance on AI
2. validation strategies
3. cognitive load

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

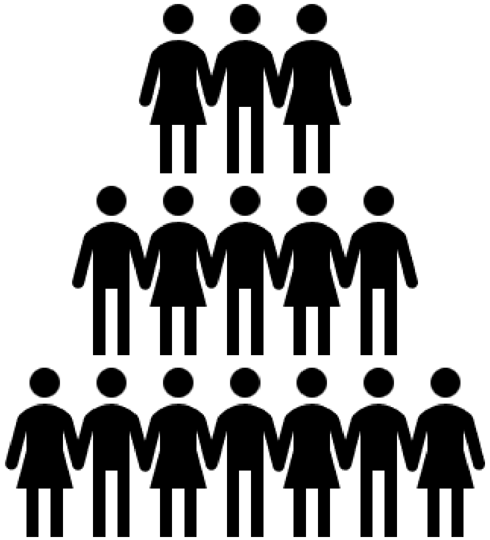
box plot

overlay scatter plot over boxplot using matplotlib

string rewriting

parse rewrite rules and apply to string

participants

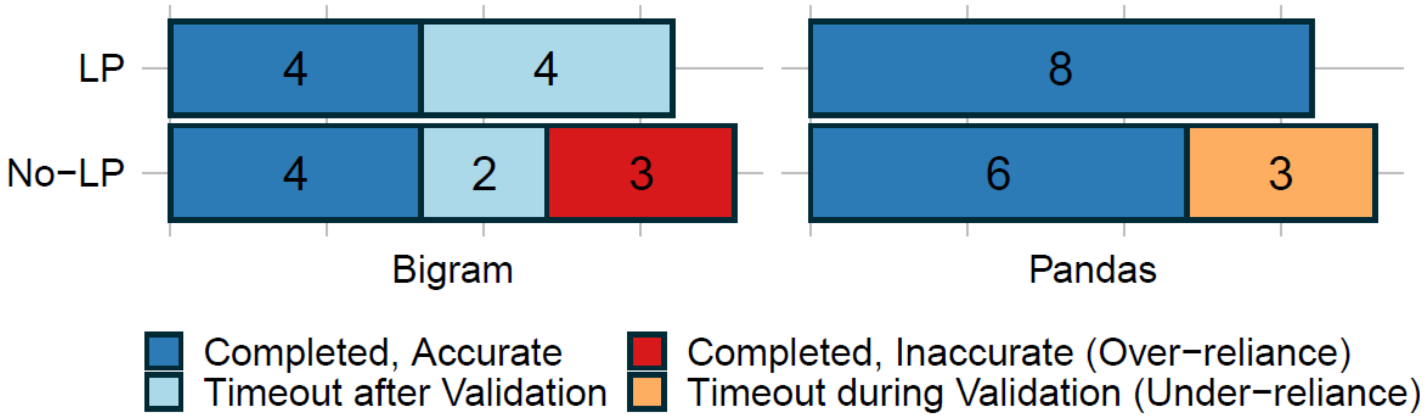


n = 17

occupation:
15 academia / 2 industry

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

rq1: 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

rq1: 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)

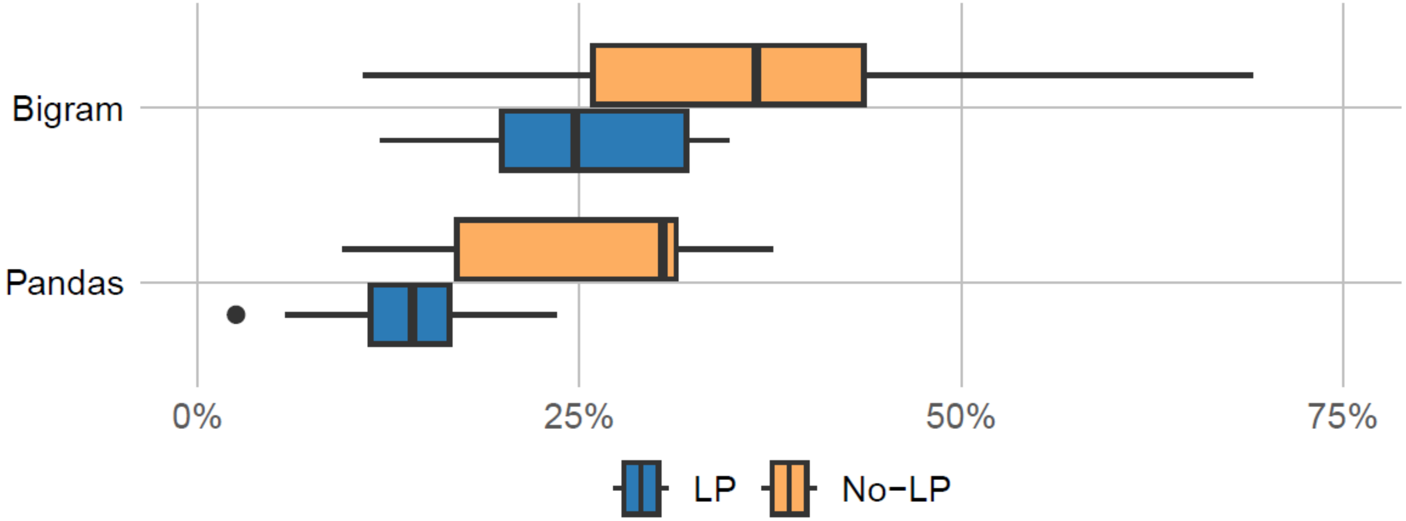
"it **saved me the effort** of writing multiple print statements." (P1)

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

rq2: validation strategies

percentage of time spent in Suggestion Panel

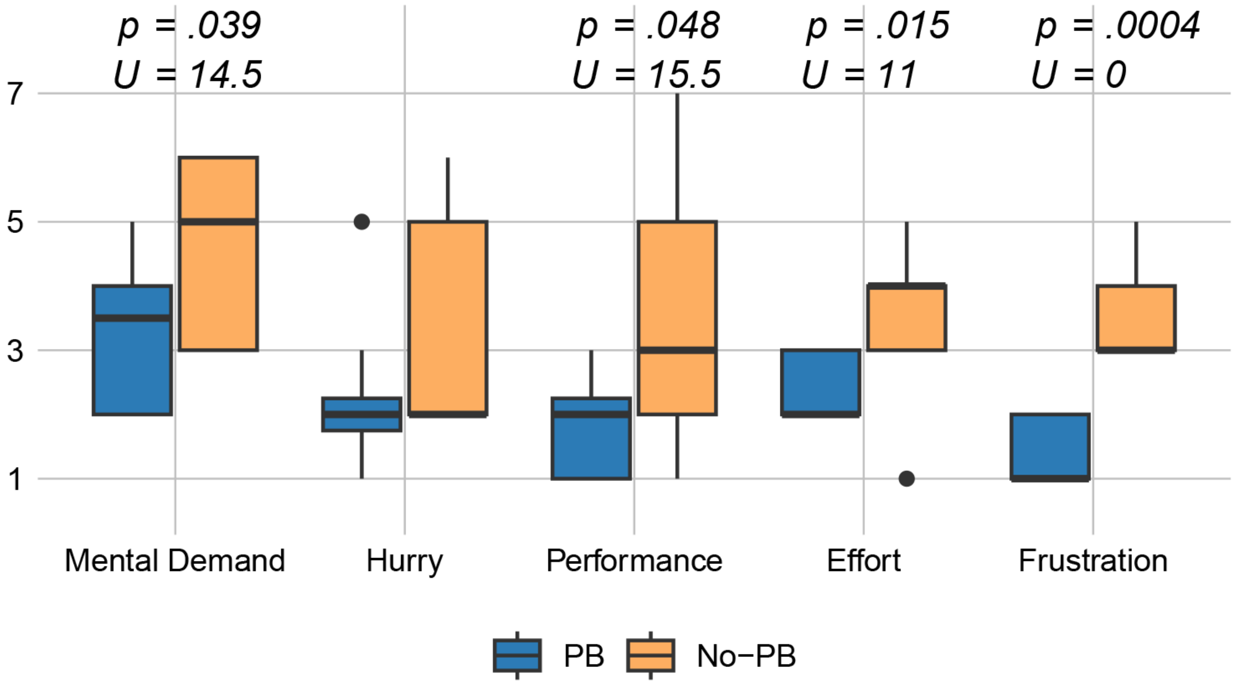


"I didn't look too closely in the actual code,
I was *just looking at the runtime values* on the side." (P1)

leap participants spent less time reading code

rq3: cognitive load

NASA TLX cognitive load metrics on Pandas



leap significantly reduced cognitive load of AI-assisted programming on tasks amenable to validation by execution

how can we make the tools more usable?

1. help with validation

II.

leap:

validating AI-generated code
with live programming

other designs

for new tools

how can we make the tools more usable?

1. help with validation
   [Vasconcelos et al, NeurIPS'22]
   - highlight parts of the suggestion that will require editing
   - show that using LLM confidence scores doesn't work
   - train a separate model to predict this

how can we make the tools more usable?

2. eliminate distractions
   [Sun et al, ICSE'23]
   - train a lightweight model to predict *low-return* prompts
   - helps save 5-20% of computational cost

how can we make the tools more usable?

3. give user more control

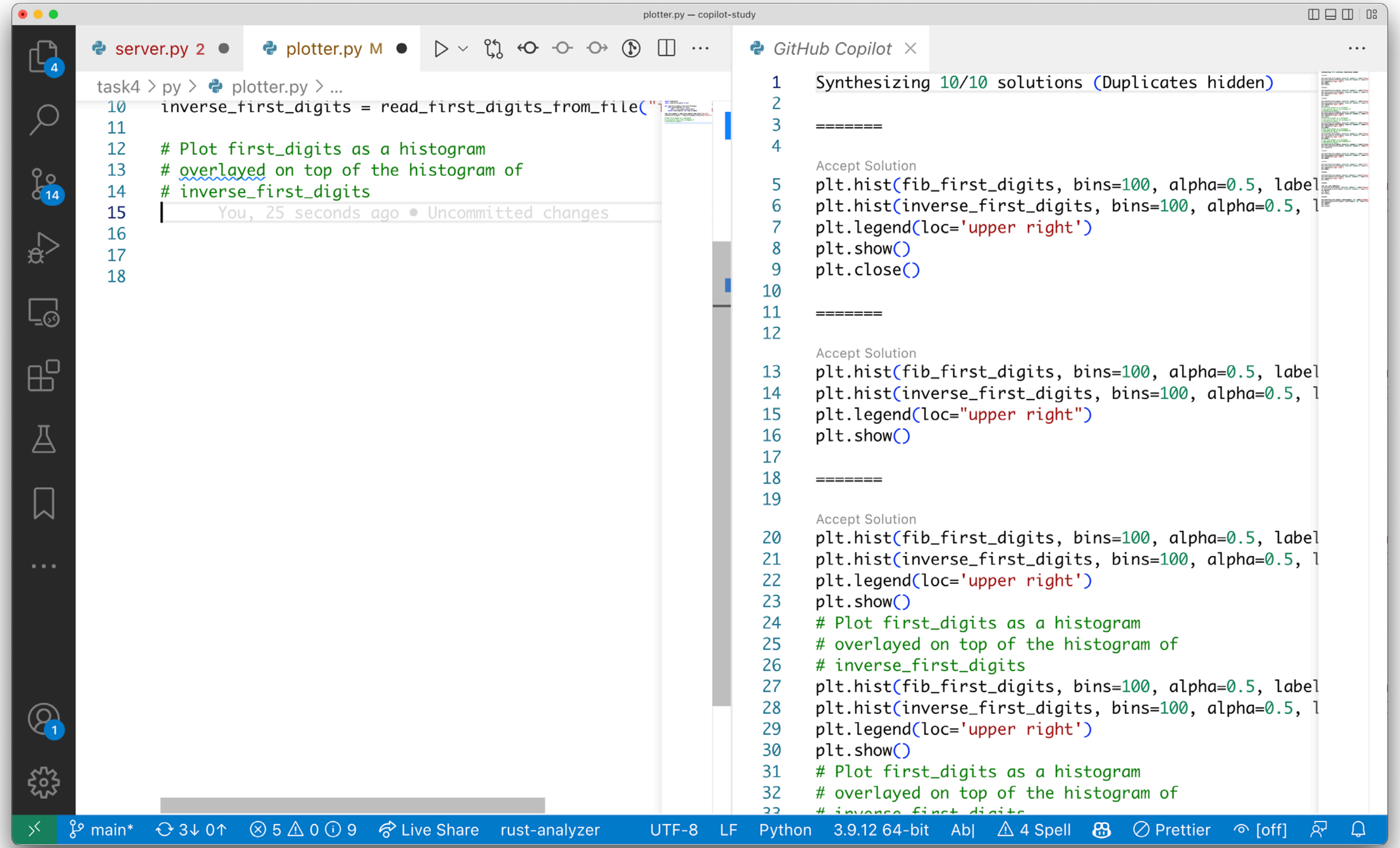
   [Ross et al, IUI’23]

   - conversational programming assistant
   - initiative with the user
   - user controls the context (via selection)

how can we make the tools more usable?

4. navigating solution spaces

navigating solution spaces

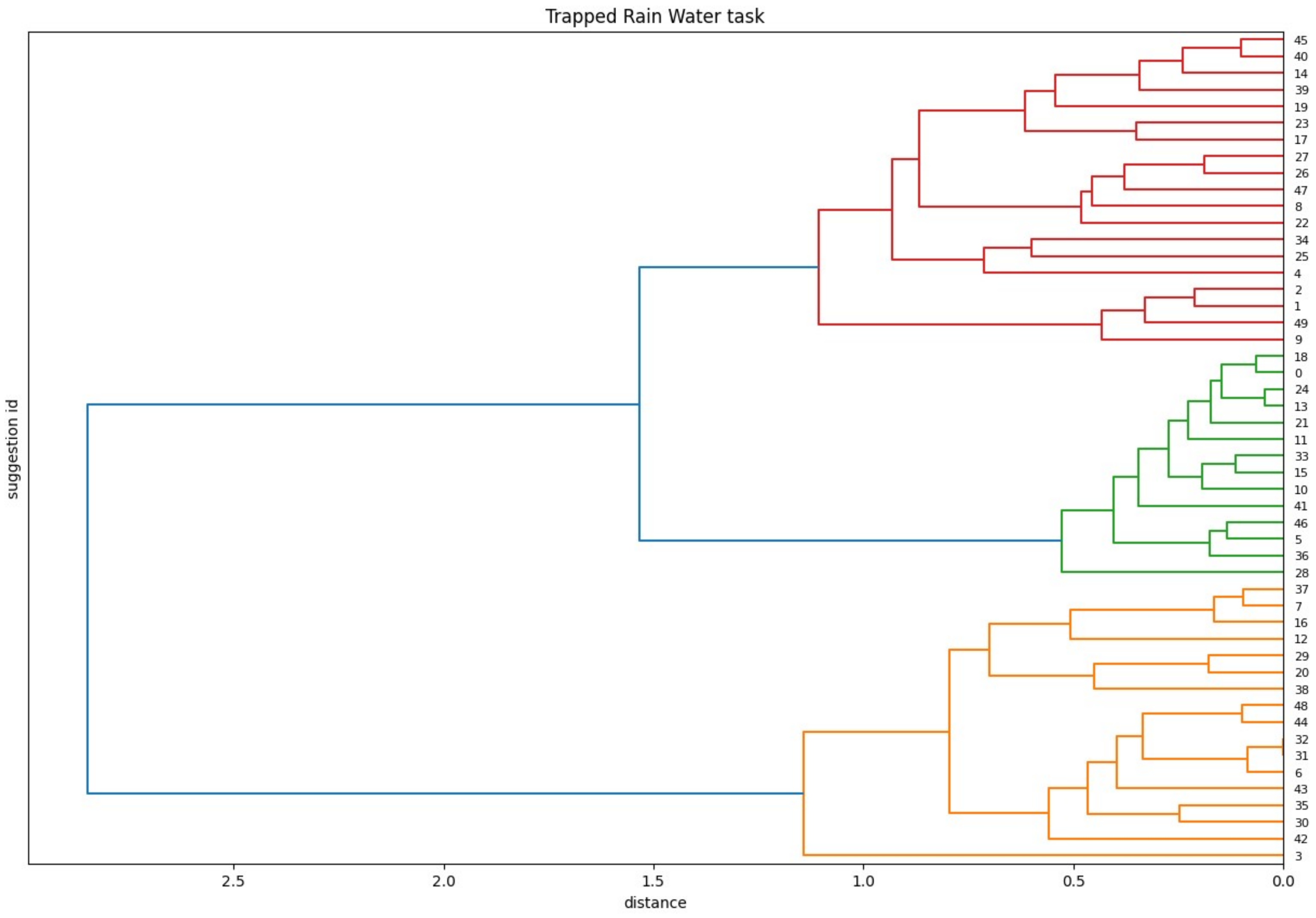


Copilot's multi-suggestion pane

hard to distinguish

hard to find common / rare solutions

our ongoing work




```
def max_water ( heights : List [ int ] ) → int :
    if not heights :
        return 0
...
    left , right = ( 0 , len ( heights ) - 1 )
    left_max , right_max = ( heights [ left ] , heights [ right ] )
...
    while left ... right :
...
        if ... left ... ⩽ ... right ... :
...
            total_water += left_max - heights [ left ]
            left += 1
        else :
...
            total_water += right_max - heights [ right ]
            right -= 1
    return total_water
```

```
def max_water ( heights : List [ int ] ) → int :
    ... = ...
    left , right ... = ( 0 , ... )
    left_max , right_max = ( 0 , ... )
    while left ... right :
        if heights [ left ] ... heights [ right ] :
            if heights [ left ] ... left_max :
                left_max = heights [ left ]
            else :
                water += left_max - heights [ left ]
            left += 1
        else :
            if heights [ right ] ... right_max :
                right_max = heights [ right ]
            else :
                water += right_max - heights [ right ]
            right -= 1
    return water
```

```
def max_water ( heights : List [ int ] ) → int :
...
    max_left = [ 0 ] * len ...
    max_right = [ 0 ] * len ...
...
    max_left [ 0 ] = heights [ 0 ]
    for i in range ( 1 , len ... ) :
        max_left [ i ] = max ( max_left [ i - 1 ] , heights [ i ] )
    max_right [ ... - 1 ] = heights [ ... - 1 ]
    for i in range ( len ... - 2 , - 1 , - 1 ) :
        max_right [ i ] = max ( max_right [ i + 1 ] , heights [ i ] )
...
    for i in range ( ... len ... ) :
        max_water += min ( max_left [ i ] , max_right [ i ] ) - heights [ i ]
    return max_water
```

how can we make the tools more usable?

1. help with validation
2. eliminate distractions
3. give user more control
4. navigating solution spaces

this talk

I.
how do
programmers
use existing
tools?

II.
how can
we make
the tools
more usable?

who did all the work

Michael James

Shraddha Barke

Kasra Ferdowsi

Lisa Huang

Emmanuel Anaya Gonzalez

Sorin Lerner