how programmers interact with ai assistants

Nadia Polikarpova
UPenn Seminar, November 2023
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

- 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

occupation:
15 academia / 5 industry

language proficiency:
occasional / regular / professional

prior Copilot experience:
9 no / 11 yes

n = 20
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

```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

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

**copilot:** auto-completes current logical unit (line of code)

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

might cherry-pick parts from different suggestions

copilot suggests multiple alternatives

validates code by executing or consulting documentation
acceleration vs exploration

Interaction modes based on participant's expertise

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

Median time spent in each interaction mode (min)

- Occasional (n=9)
- Regular (n=2)
- Professional (n=11)

- PCU (n=9)
- No-PCU (n=11)
acceleration

vs

exploration

unintentional

prompting

intentional with comments / invoke side panel

“pattern matching”

validation

explicit validation via examination / execution / documentation

unit of focus

scope

entire function + multiple alternatives

(sub-expression / statement)

mismatch tolerance

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

unwilling to edit
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?

[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]

other studies of existing tools
productivity

- 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

- 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

- extensive list of requested features
usage patterns

• 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)
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

Al suggestions
+ terminal

LP

Al 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**

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

**Algorithmic**

- **bigrams**
  - find most frequent bigram in a string

**Multiple correct suggestions**

- **box plot**
  - overlay scatter plot over boxplot using matplotlib

**No correct suggestions**

- **string rewriting**
  - parse rewrite rules and apply to string

**Fixed prompt**

**Open prompt**
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
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