Measuring The Runtime Performance Of C++ Code Written By Humans Using GitHub Copilot

Daniel Erhabor

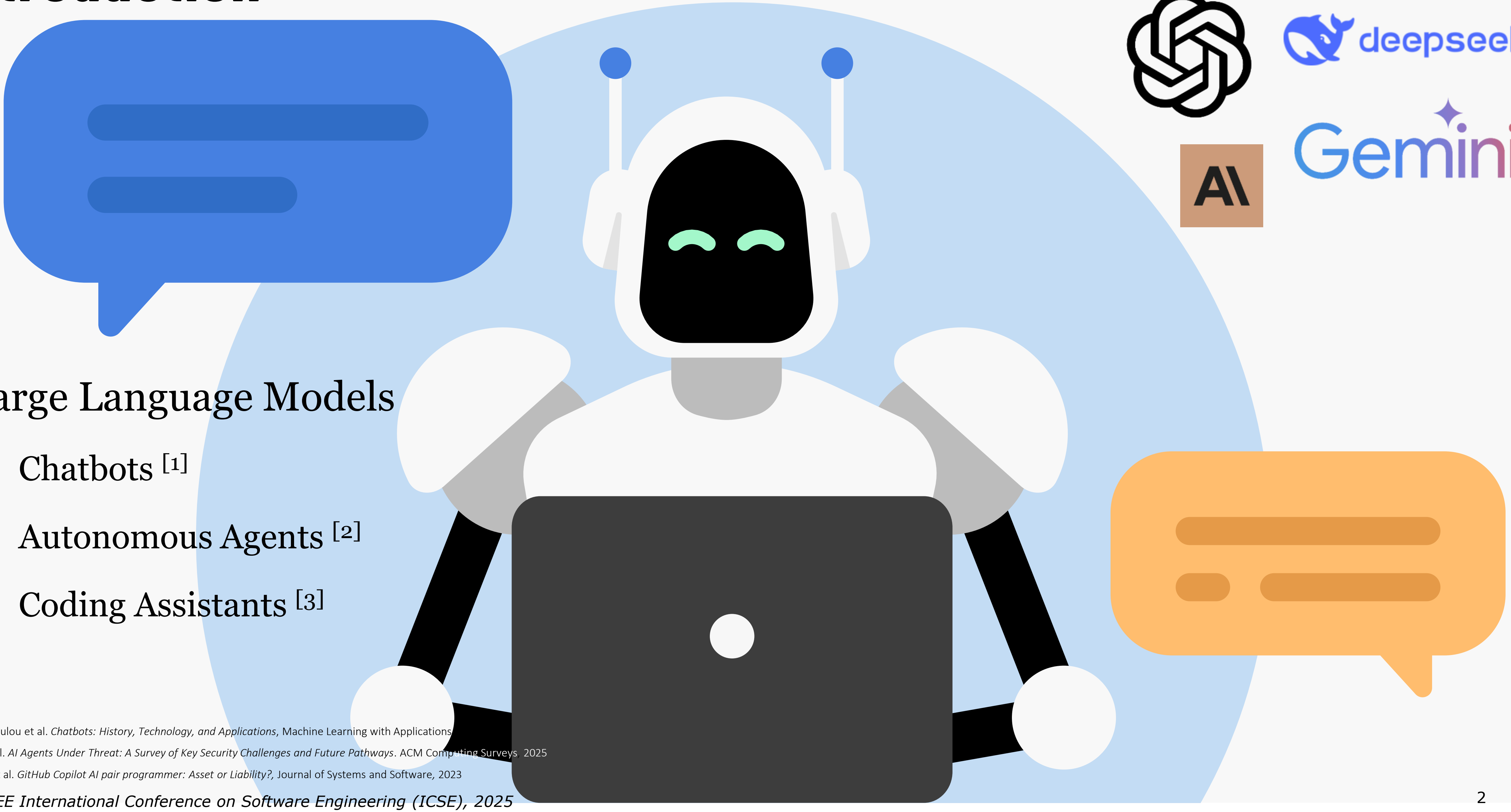
Sreeharsha Udayashankar

Meiyappan Nagappan

Samer Al-Kiswany

UNIVERSITY OF WATERLOO

Introduction



- Large Language Models
  - Chatbots [1]
  - Autonomous Agents [2]
  - Coding Assistants [3]

[1] E. Adamopoulou et al. *Chatbots: History, Technology, and Applications*, Machine Learning with Applications

[2] Z. Deng et al. *AI Agents Under Threat: A Survey of Key Security Challenges and Future Pathways*. ACM Computing Surveys, 2025

[3] A. Dakhel et al. *GitHub Copilot AI pair programmer: Asset or Liability?*, Journal of Systems and Software, 2023

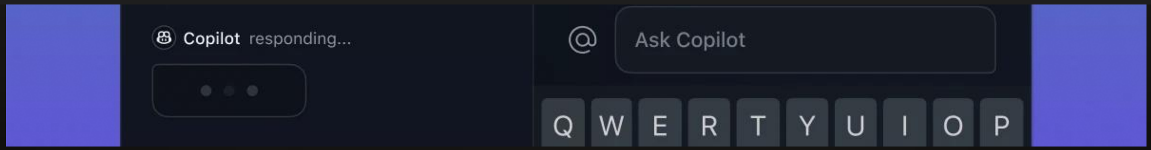
GitHub CoPilot

- Coding Assistant
  - Powered by OpenAI Codex (GPT-3)*
  - Visual Studio extension

JS test.js 1

JS test.js > calculateDaysBetweenDates

```javascript
function calculateDaysBetweenDates(begin, end) {
    var beginDate = new Date(begin);
    var endDate = new Date(end);
    var days = Math.round((endDate - beginDate) / (1000 * 60 * 60 * 24));
    return days;
}
```



* Now powered by choice of model between GPT-4o / Claude 3.5

GitHub CoPilot – Previous Studies

Usability / Functionality

Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models

Priyan Vaithilingam, pvaithilingam@g.harvard.edu, Harvard University, USA
Tianyi Zhang, tianyi@purdue.edu, Purdue University, USA
Elena L. Glassman, glassman@seas.harvard.edu, Harvard University, USA

ABSTRACT

Recent advances in Large Language Models (LLM) have made automatic code generation possible for real-world programming tasks in general-purpose programming languages such as Python. However, there are few human studies on the usability of these tools and how they fit the programming workflow. In this work, we conducted a within-subjects user study with 24 participants to understand how programmers use and perceive Copilot, a LLM-based code generation tool. We found that, while Copilot did not necessarily improve the task completion time or success rate, most participants preferred to use Copilot in daily programming tasks, since Copilot often provided a useful starting point and saved the effort of searching online. However, participants did face difficulties in understanding, editing, and debugging code snippets generated

A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges

Jenny T. Liang, Carnegie Mellon University, Pittsburgh, PA, USA, jtliang@cs.cmu.edu
Chenyang Yang, Carnegie Mellon University, Pittsburgh, PA, USA, cyang3@cs.cmu.edu
Brad A. Myers, Carnegie Mellon University, Pittsburgh, PA, USA, bam@cs.cmu.edu

Figure 1: An overview of the topics covered in our usability study of AI programming assistants.

On Programming Variability with Large Language Model-based Assistant

Mathieu Acher, University of Rennes, INSA, IRISA, Inria, IUF, Rennes, France, mathieu.acher@irisa.fr
José Galindo Duarte, University of Sevilla, Sevilla, Spain, jagalindo@us.es
Jean-Marc Jézéquel, University of Rennes, IRISA, Inria, IUF, Rennes, France, jean-marc.jezequel@irisa.fr

ABSTRACT

Programming variability is central to the design and implementation of software systems that can adapt to a variety of contexts and requirements, providing increased flexibility and customization. Managing the complexity that arises from having multiple features, variations, and possible configurations is known to be highly challenging for software developers. In this paper, we explore how large language model (LLM)-based assistants can support the programming of variability. We report on new approaches made possible with LLM-based assistants, like: features and variations can be implemented as prompts; augmentation of variability out of LLM-based domain knowledge; seamless implementation of variability in different kinds of artefacts, programming languages, and frameworks, at different binding times (compile-time or run-time). We are sharing our data (prompts, sessions, generated code, etc.) to support the assessment of the effectiveness and robustness of LLMs for variability-related tasks.

KEYWORDS

variability, programming, software product lines, generative AI, large language model

Security

Lost at C: A User Study on the Security Implications of Large Language Model Code Assistants

Gustavo Sandoval*, Hammond Pearce*, Teo Nys, Ramesh Karri, Siddharth Garg, Brendan Dolan-Gavitt
New York University

Large Language Models for Code: Security Hardening and Adversarial Testing

Jingxuan He, ETH Zurich, Switzerland, jingxuan.he@inf.ethz.ch
Martin Vechev, ETH Zurich, Switzerland, martin.vechev@inf.ethz.ch

Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions

research highlights

DOI:10.1145/3610721

By Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri

Runtime Performance

- Page-load Latency vs Customer Bounce Rate [1]

Few additional *seconds*

90% bounce rate increase

- Latency vs Sales revenue [2]

100 *milliseconds*

Millions of $

- Large-scale systems track 99th percentile latency in µs [3]

[1 Google. Mobile Page Speed and Industry Benchmarks, 2017.

[2] Gigaspaces, Amazon Found Every 100ms of Latency Cost them 1% in Sales, 2023.

[3] Udayashankar S et al. Draconis: Network Accelerated Scheduling for Microsecond Scale Workloads, ACM SIGOPS European Conference on Computer Systems (EuroSys), 2024

UNIVERSITY OF WATERLOO

Our Contributions

- First study focused on runtime performance
  - Systems / Infrastructure Engineering

32 participants

Two C++ problems

+/- Copilot

- Sample RQ: *Is there a runtime performance difference in C++ code when using Copilot?*
  - Spoiler Alert!: On average, copilot-aided solutions were **15-27% slower** than their unaided counterparts.

UNIVERSITY OF WATERLOO

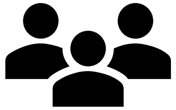
Outline

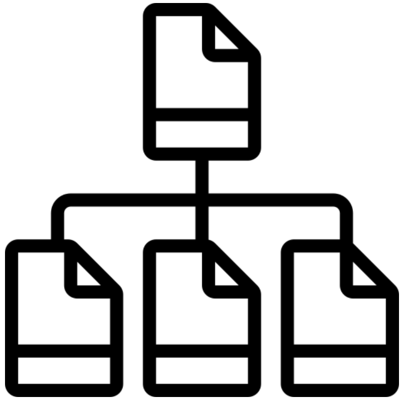
- Introduction
- Methodology
- Evaluation ~ RQs
- Takeaways

Methodology

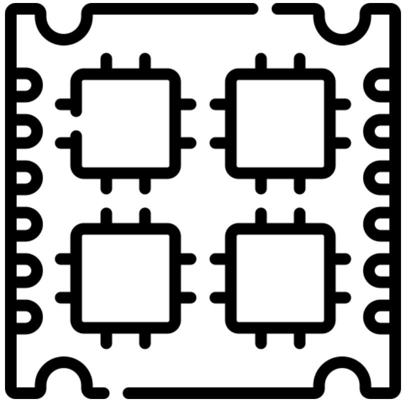
32 Participants

- Participants asked to solve two problems
  - One with and another without Copilot assistance





Problem A: File I/O



Problem B: Multi threading

Must be quickly solvable!

Must have multiple solutions with only performance differences!

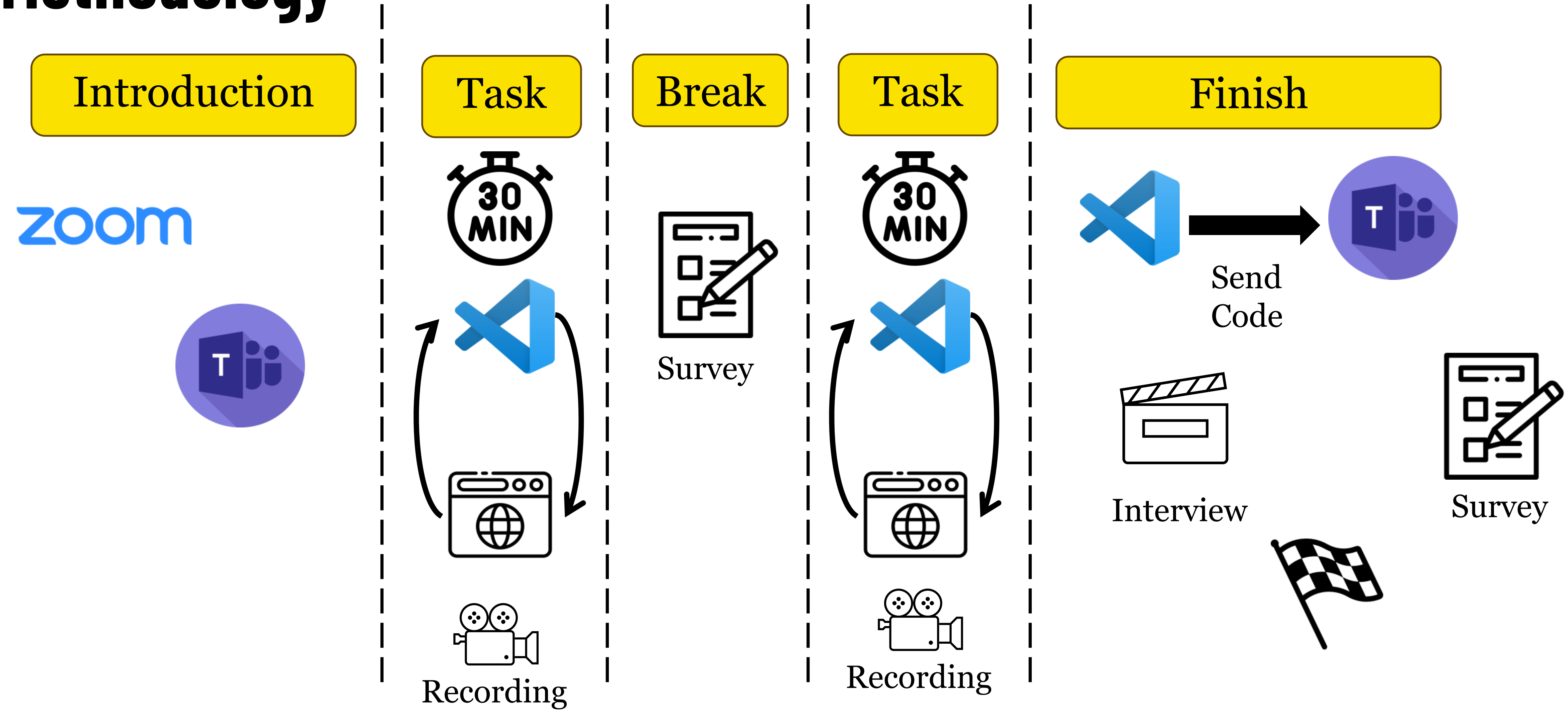
Methodology

32 Participants

- Participants asked to solve two problems
  - One with and another without Copilot assistance
- Code Stubs

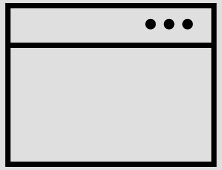
```
.....

void primaryFunction() {
    // YOUR CODE GOES HERE
}
.....

int main() {
    initialization();
    primaryFunction();
    sanityCheck();
    return 0;
}
```

Methodology

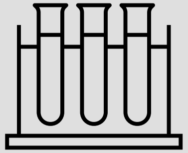


RESEARCH QUESTIONS

RQ 0: *Does Copilot influence program correctness?*



Code Analysis



Test Cases

Copilot leads developers to produce correct code in most cases!

Test Failure ✕ Compilation Failure ✕

#	partID	problem	mode	compiled	passed
1	P32 ✕	B	C	TRUE	FALSE
2	P30	B	NC	TRUE	FALSE
3	P23	A	NC	TRUE	NULL
4	P15 ✕	A	NC	FALSE	NULL
5	P15	B	C	FALSE	NULL
6	P7	A	NC	FALSE	NULL
7	P6	B	NC	TRUE	FALSE
8	P3	A	NC	TRUE	FALSE

TABLE OF INVALID RUNS

Without Copilot
A: 4/16
B: 2/16

With Copilot
A: 0/16
B: 2/16

RESEARCH QUESTIONS

RQ 1: *Is there a runtime performance difference in C++ code when using Copilot?*

Runtime Performance

Copilot-aided solutions possess worse runtime performance than unaided ones!

- 32 runs on 8-core Intel Xeon D-1548 @ 2 GHz
- Wilcoxon Rank Sum Test

Problem	*Mode*	*Valid Runs*	*Mean*	*Median*	*Min*	*Max*
A	C	16 x 32	34.86 s	34.85 s	33.82 s	36.02 s
A	NC	12 x 32	26.02 s	34.47 s	4.045 s	35.84 s
B	C	14 x 32	1898 ms	945.4 ms	612.1 ms	7356 ms
B	NC	14 x 32	1628 ms	943.9 ms	494.9 ms	6761 ms

TABLE OF RUNTIME PERFORMANCE

Problem A: Mean runtime with Copilot is **27% slower.**

Problem B: Mean runtime with Copilot is **15% slower.**

Copilot-aided solutions possess worse runtime performance than unaided ones!

RESEARCH QUESTIONS

RQ 1: *Is there a runtime performance difference in C++ code when using Copilot?*

Runtime Performance

- 32 runs on 8-core Intel Xeon D-1548 @ 2 GHz
- Wilcoxon Rank Sum Test

Problem	*Mode*	*Valid Runs*	*Mean*	*Median*	*Min*	*Max*
A	C	16 x 32	34.86 s	34.85 s	33.82 s	36.02 s
A	NC	12 x 32	26.02 s	34.47 s	4.045 s	35.84 s
B	C	14 x 32	1898 ms	945.4 ms	612.1 ms	7356 ms
B	NC	14 x 32	1628 ms	943.9 ms	494.9 ms	6761 ms

TABLE OF RUNTIME PERFORMANCE

Problem A: Fastest solution without Copilot is **8x faster** than with.

Same participant had a solution closer to median runtime when using CoPilot for Problem B!

Copilot-aided solutions possess worse runtime performance than unaided ones!

RESEARCH QUESTIONS

RQ 1: *Is there a runtime performance difference in C++ code when using Copilot?*

Runtime Performance

Problem A:
Mean runtime with Copilot is **27% slower.**

Problem B:
Mean runtime with Copilot is **15% slower.**

Problem A: Fastest solution without Copilot is **8x faster** than with.

Same participant had a solution closer to median runtime when using CoPilot for Problem B!

Copilot tends to sway developers towards slower solutions!

RESEARCH QUESTIONS

RQ 2: *Does Copilot sway developers towards or away from solutions with faster runtime performance?*

P12 (B) when implementing an optimization:

"Copilot didn't understand me well; I just gave up and wrote it myself."

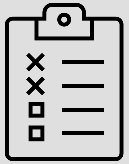
Video Analysis

Open Coding

Participant Survey

RESEARCH QUESTIONS

Bonus: *How do participant demographics affect these results?*

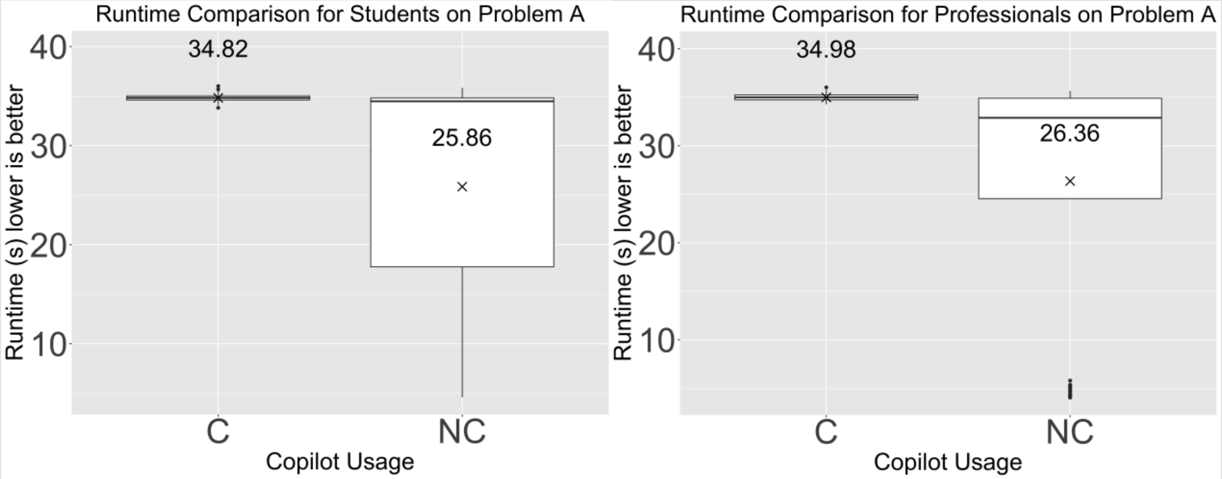


Participant Survey

Runtime Performance

Copilot-aided solutions are slower regardless of participant demographics!

Lower runtime is better.



Students **Professionals**

Problem A

Summary

- Coding assistants powered by LLMs are popular
  - Generated code needs to be carefully examined
- GitHub Copilot
  - Produces functionally correct code in most cases
  - Does not target better runtime performance and hinders developers trying to do so.

- Anonymized Participant Data / Scripts: Artifact Link

Artifacts Available acm
Artifacts Evaluated acm Functional
Artifacts Evaluated acm Reusable

UNIVERSITY OF WATERLOO

UNIVERSITY OF
WATERLOO