LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Media Informatics Group

IUIs for Software Development

Thomas Weber, Sven Mayer

Software Development Tools with AI

- Automatic bug detection
- Automatic testing
- Code optimization
- Documentation generation
- Automatic vulnerability detection
- Adaptive Dev Tools

Software Development Tools with AI

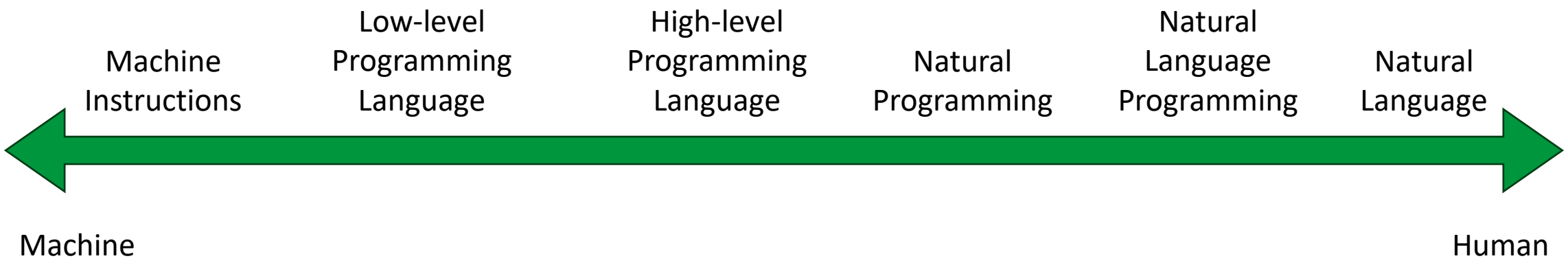
- Automatic bug detection
- Automatic testing
- Code optimization
- Documentation generation
- Automatic vulnerability detection
- Adaptive Dev Tools

How often do you use these tools?

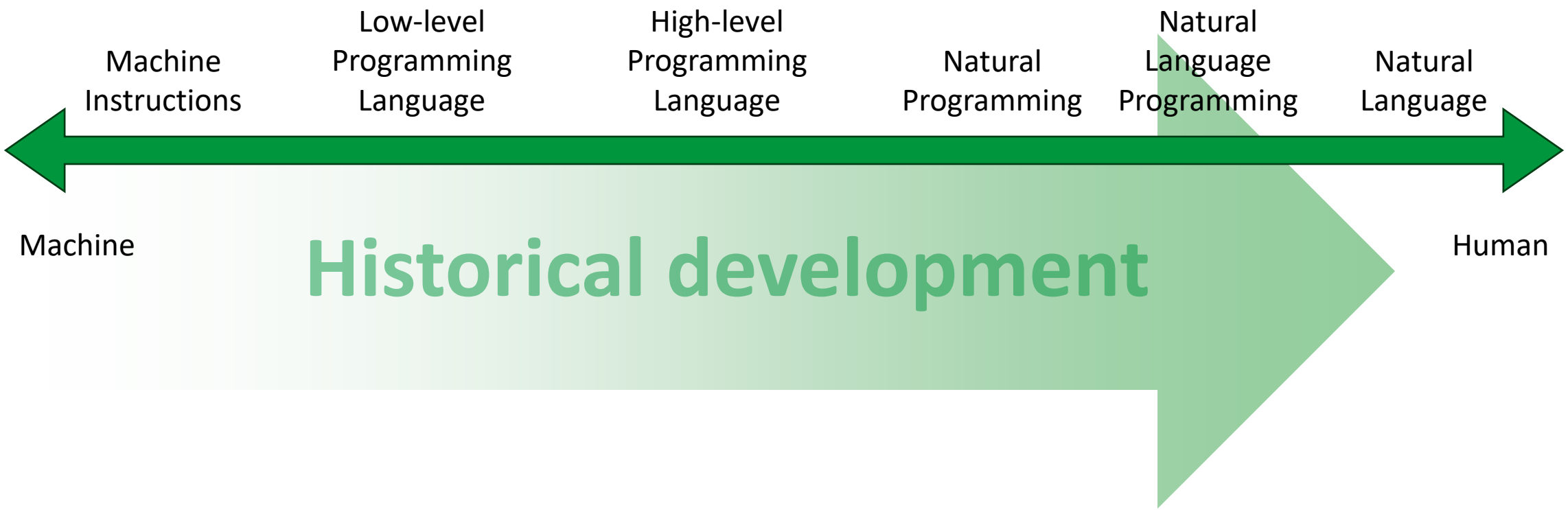
How often do you perform the tasks that these tools support?

Programming with AI

Human-Machine-Interaction for Programming



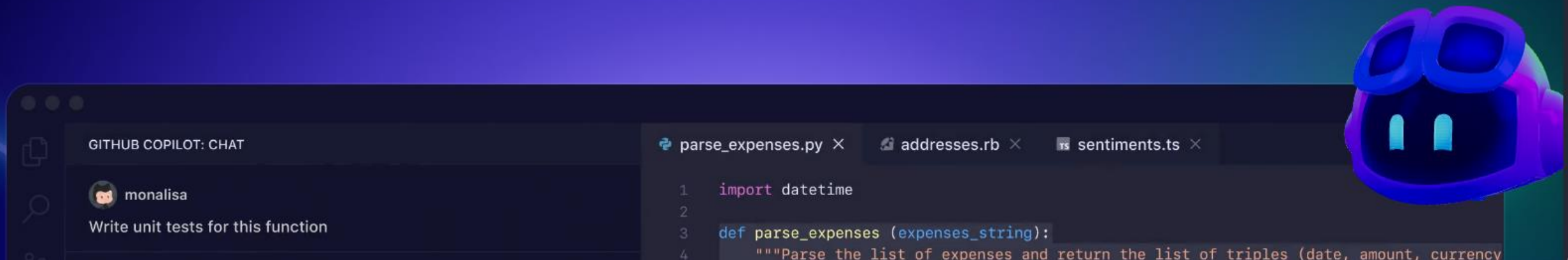
Human-Machine-Interaction for Programming



GitHub Copilot

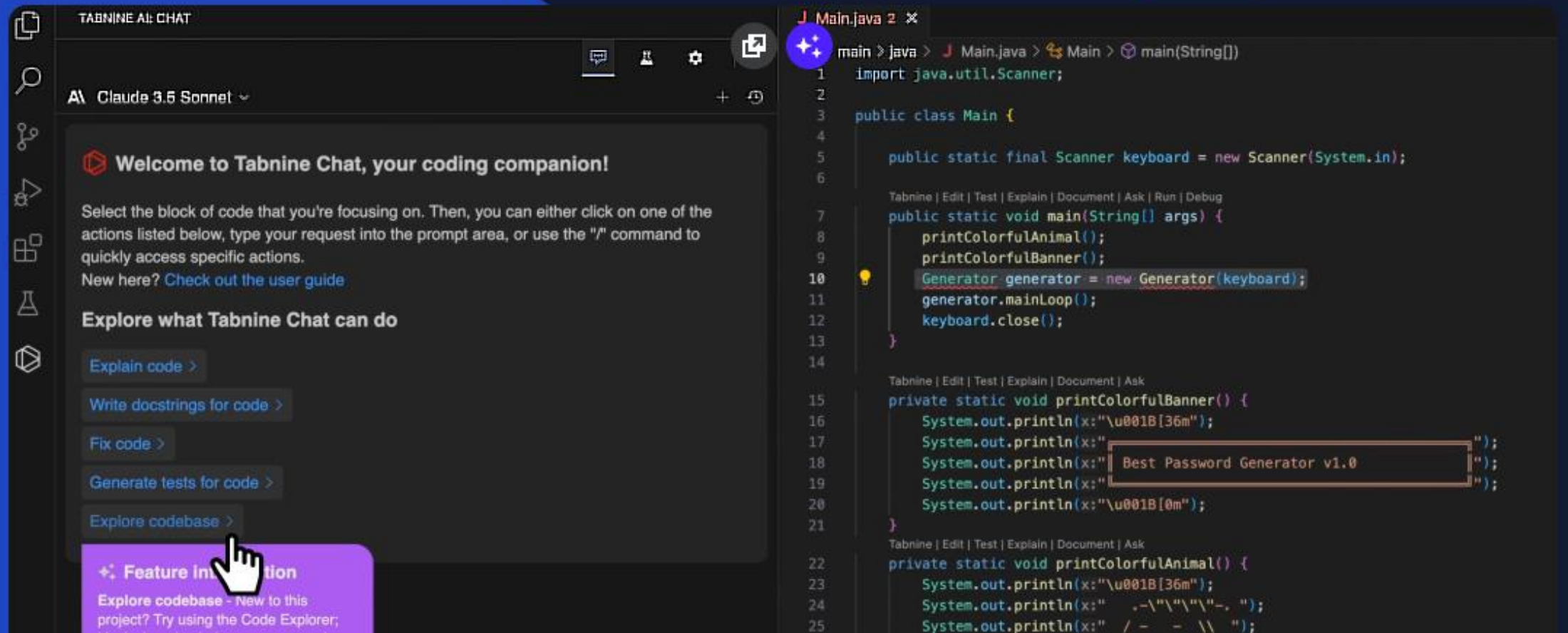
The world's most widely adopted AI developer tool

Get started with GitHub Copilot >



// See Tabnine in action //

Plan > Create > Document > Test > Review > Explain > Maintain >



AI Generators | Code Insights | Code Enhancers | Misc.

{ Code Generator }

Generate Code Online with AI

Write a function in JavaScript that detects face features

Create
Standard

Select a flavor

- Minimal
- Standard
- Documented
- With Tests

Your generated code will be shown here.

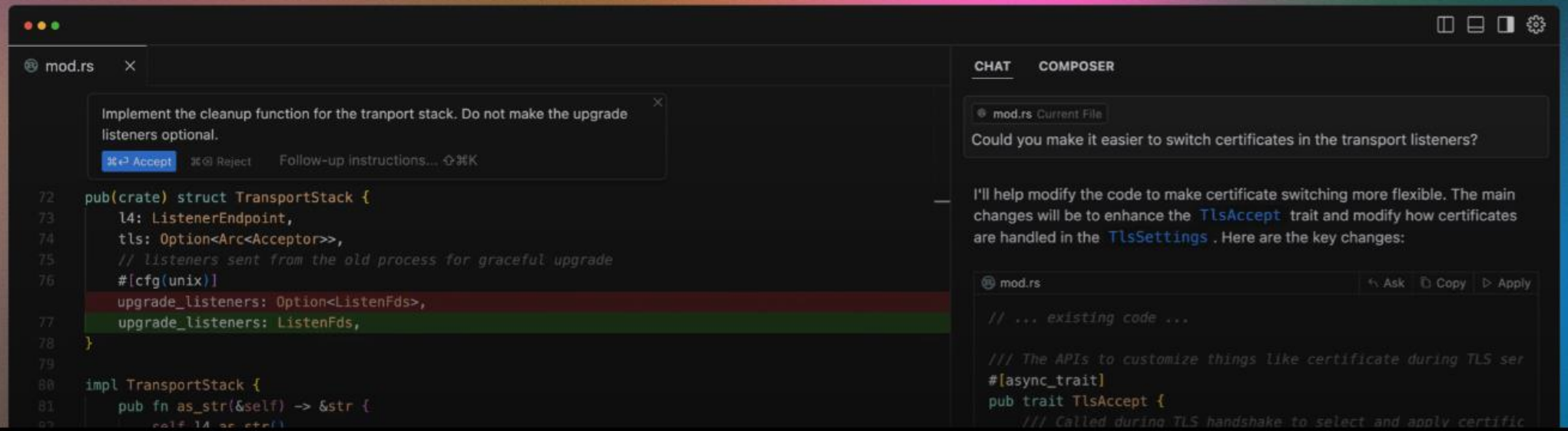
Recently Used › Code Generator

https://codepal.ai

The AI Code Editor

Built to make you extraordinarily productive,
Cursor is the best way to code with AI.

DOWNLOAD FOR WINDOWS WATCH DEMO



Built by Cognition

Devin is a collaborative AI teammate

Built to help ambitious engineering teams achieve more.

Scroll down ↓

GITHUB NEXT

GitHub Spark

Can we enable anyone to create or adapt software for themselves, using AI and a fully-managed runtime?

WHAT'S IT FOR?

Building and sharing personalized micro apps ("sparks")

SHARE

STAGE

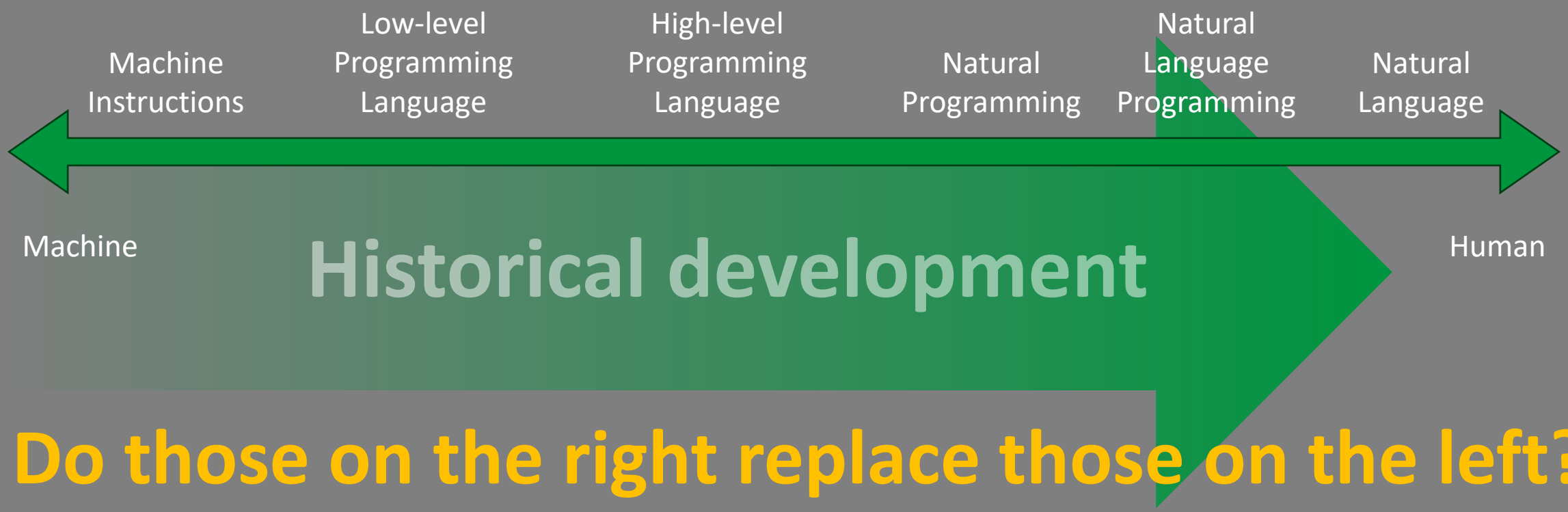
TECHNICAL PREVIEW

WHO MADE IT?

- Devon Rifkin
- Terkel Gjervig Nielsen
- Cole Bemis
- Alice Li

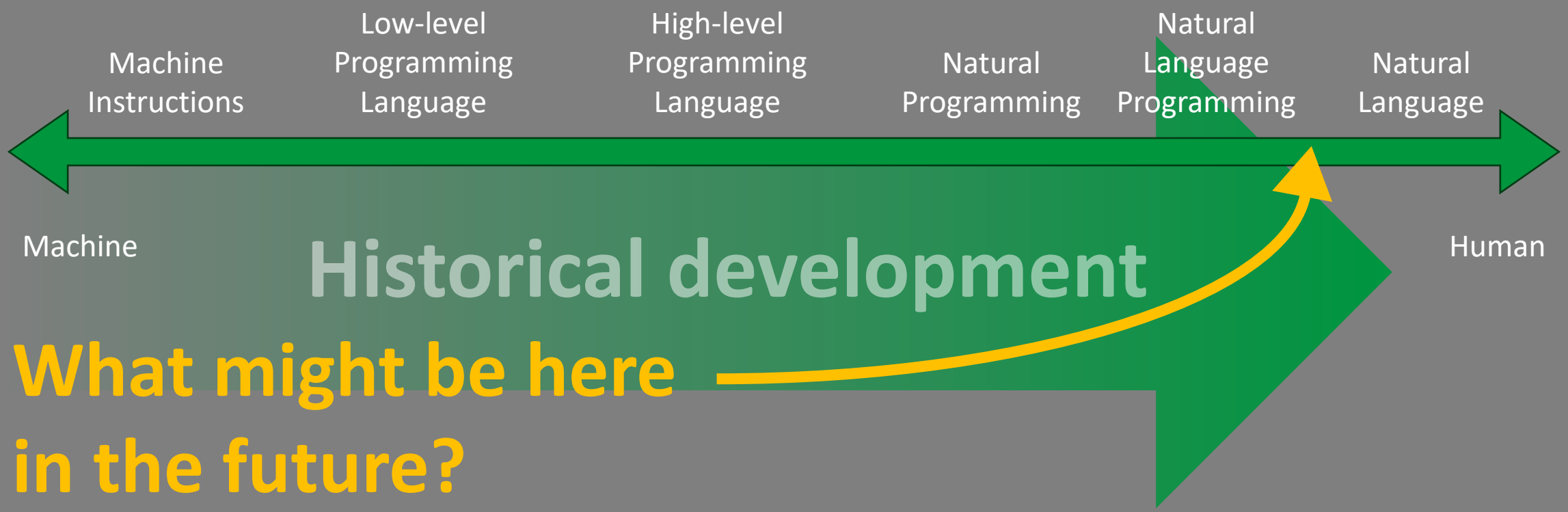
https://youtu.be/zB-izOTcQ6s?t=24

Human-Machine-Interaction for Programming



Do those on the right replace those on the left?

Human-Machine-Interaction for Programming



Leaderboard

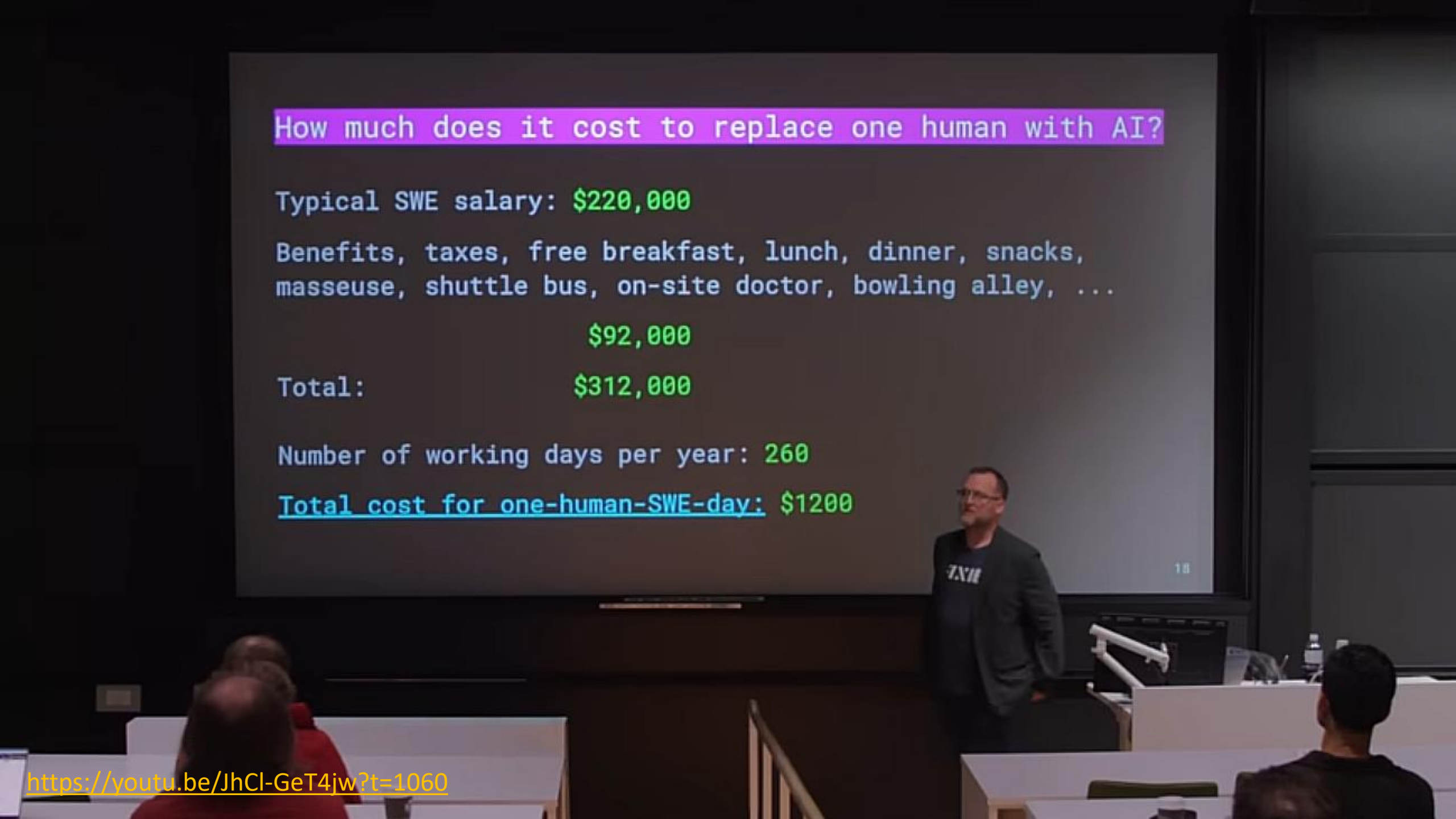
Lite | Verified | Full

Model	% Resolved	Org	Date	Logs	Trajs	Site
Amazon Q Developer Agent (v20241202-dev)	54.80	aws	2024-12-02	✓	✓	
devlo	54.20		2024-11-08	✓	✓	
✅ OpenHands + CodeAct v2.1 (claude-3-5-sonnet-20241022)	53.00		2024-10-29	✓	✓	
Engine Labs (2024-11-25)	51.80		2024-11-25	✓	✓	
Agentless-1.5 + GPT 4o (claude-3-5-sonnet-20241022)	50.80		2024-12-02	✓	✓	
Solver (2024-10-28)	50.00		2024-10-28	✓	✓	
Bytedance MarsCode Agent	50.00		2024-11-25	✓	✓	
nFactorial (2024-11-05)	49.20		2024-11-05	✓	✓	
Tools + Claude 3.5 Sonnet (2024-10-22)	49.00		2024-10-22	✓	✓	
✅ Composio SWE-Kit (2024-10-25)	48.60		2024-10-25	✓	✓	
✅ AppMap Navie v2	47.20		2024-11-06	✓	✓	
Emergent E1 (v2024-10-12)	46.60		2024-10-23	✓	✓	
AutoCodeRover-v2.0 (Claude-3.5-Sonnet-20241022)	46.20		2024-11-08	✓	✓	
Solver (2024-09-12)	45.40		2024-09-24	✓	✓	
Gru(2024-08-24)	45.20		2024-08-24	✓	✓	
Solver (2024-09-12)	43.60		2024-09-20	✓	✓	
nFactorial (2024-10-30)	41.60		2024-10-30	✓	✓	
Nebius AI Qwen 2.5 72B Generator + LLama 3.1 70B Critic	40.60		2024-11-13	✓	✓	
Tools + Claude 3.5 Haiku	40.60		2024-10-22	✓	✓	
Honeycomb	40.60		2024-08-20	✓	✓	
Composio SWEkit + Claude 3.5 Sonnet (2024-10-16)	40.60		2024-10-16	✓	✓	
EPAM AI/Run Developer Agent v20241029 + Anthopic Claude 3.5 Sonnet	39.60	epam	2024-10-29	✓	✓	
Amazon Q Developer Agent (v20240719-dev)	38.80	aws		✓	✓	

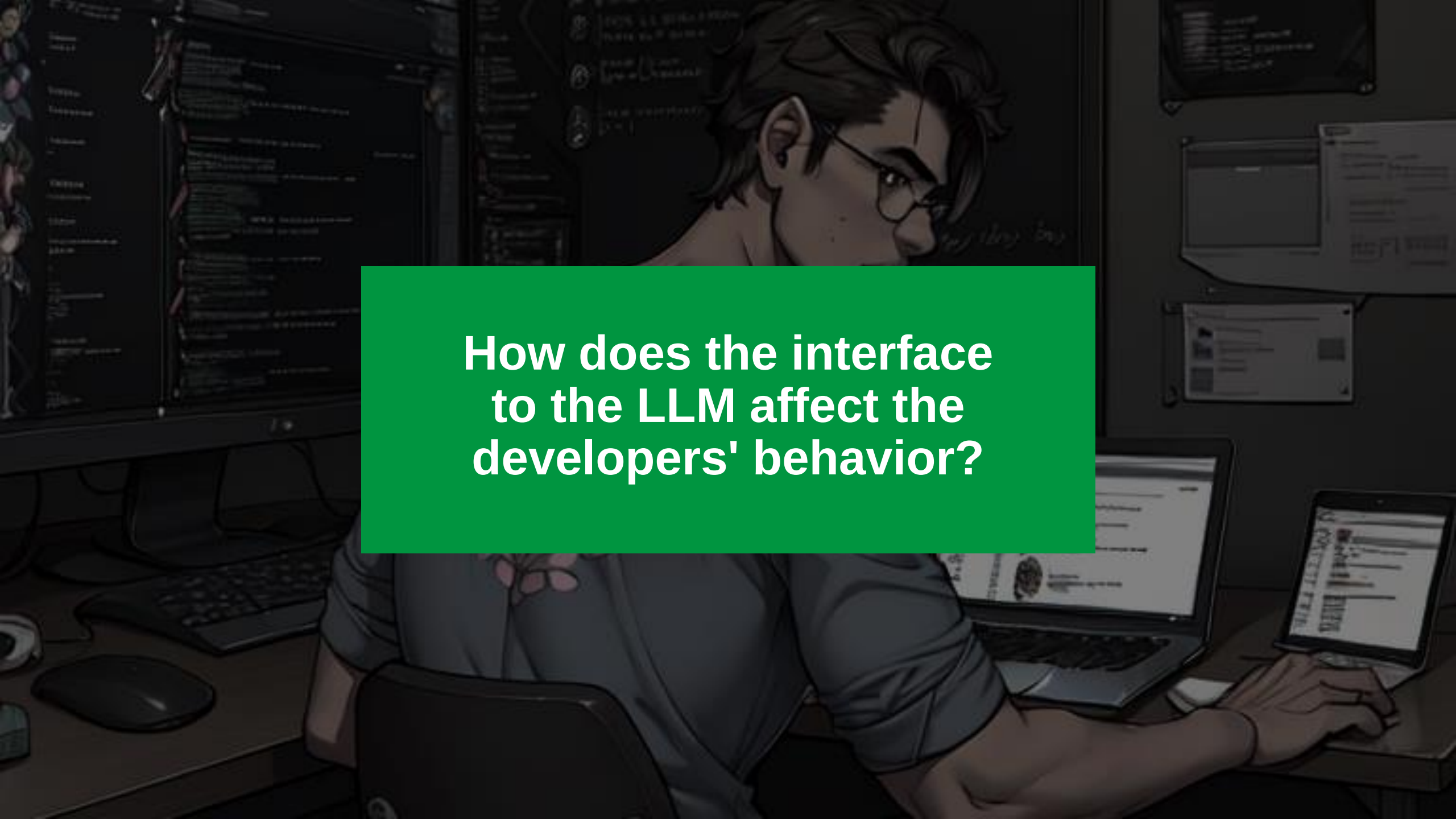
SWE-bench **Lite** is a subset of SWE-bench that's been curated to make evaluation less costly and more accessible [Post].

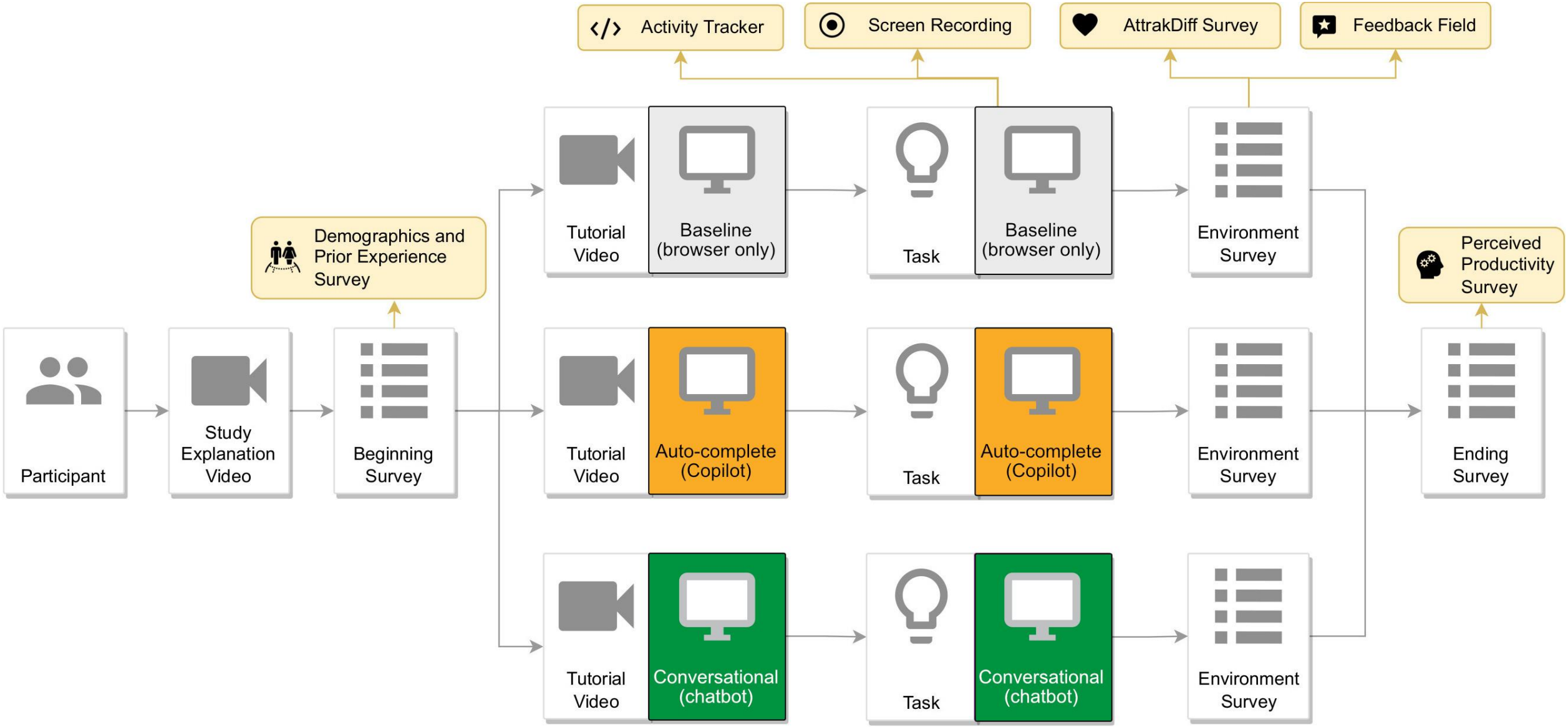
SWE-bench **Verified** is a human annotator filtered subset that has been deemed to have a ceiling of 100% resolution rate [Post].

- The **% Resolved** metric refers to the percentage of SWE-bench instances (**2294** for test, **500** for verified, **300** for lite) that were *resolved* by the model.
- ✅ **Checked** indicates that we, the SWE-bench team, received access to the system and were able to reproduce the patch generations.

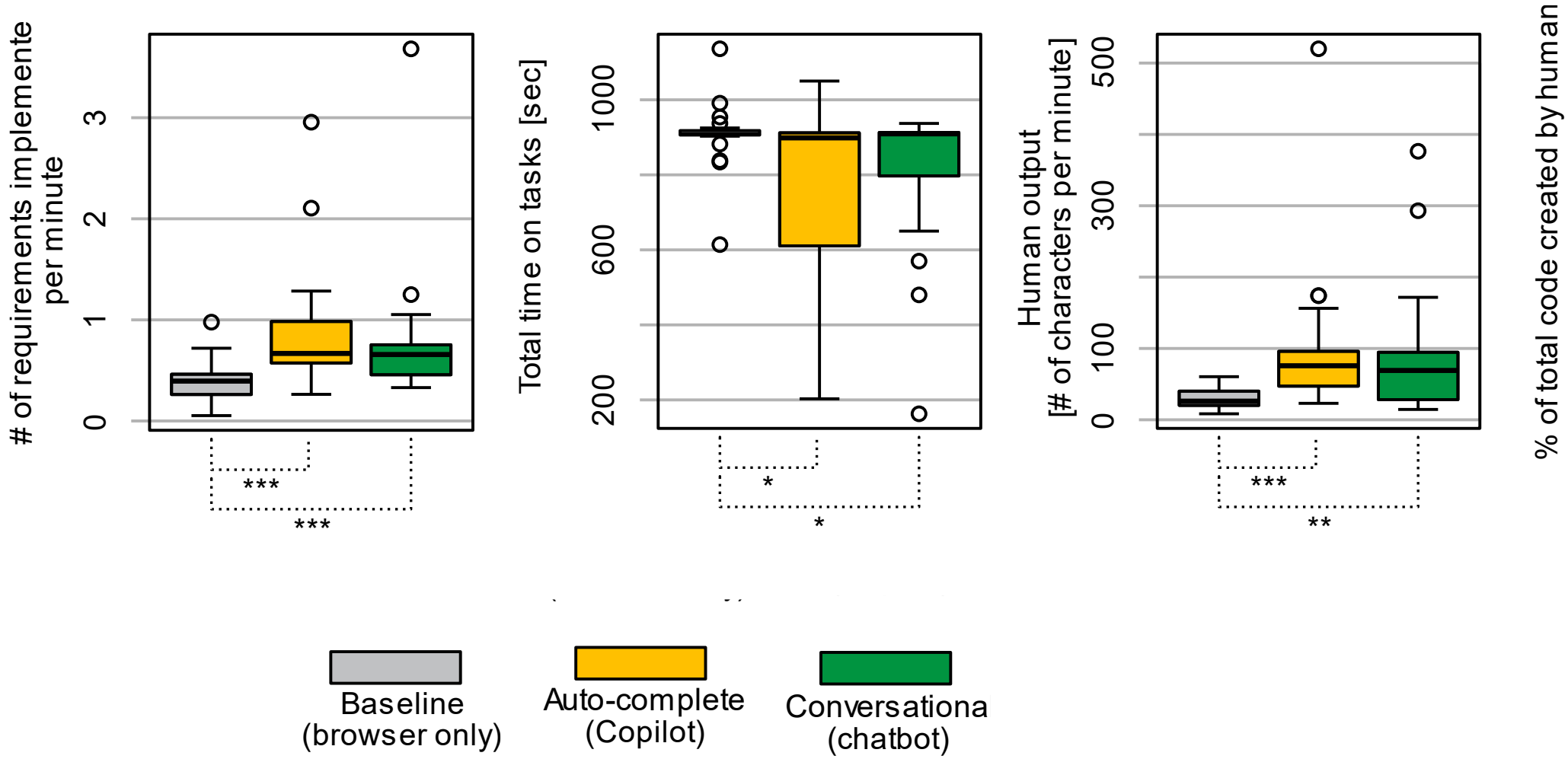


https://youtu.be/JhCl-GeT4jw?t=1060

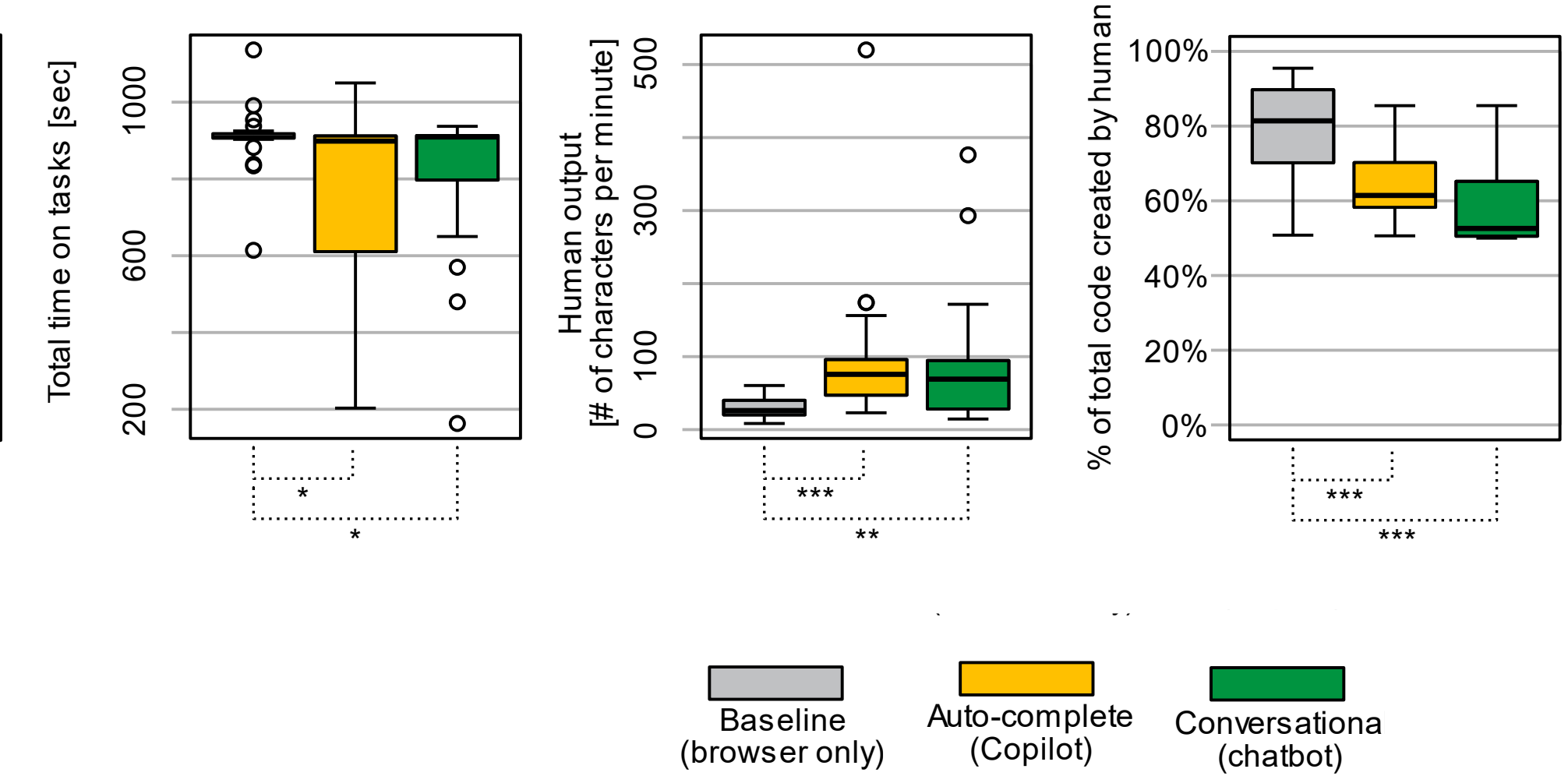

How does the interface to the LLM affect the developers' behavior?


Activity Tracker
Screen Recording
AttrakDiff Survey
Feedback Field
Demographics and Prior Experience Survey
Perceived Productivity Survey
Participant
Study Explanation Video
Beginning Survey
Tutorial Video
Baseline (browser only)
Task
Baseline (browser only)
Environment Survey
Tutorial Video
Auto-complete (Copilot)
Task
Auto-complete (Copilot)
Environment Survey
Ending Survey
Tutorial Video
Conversational (chatbot)
Task
Conversational (chatbot)
Environment Survey

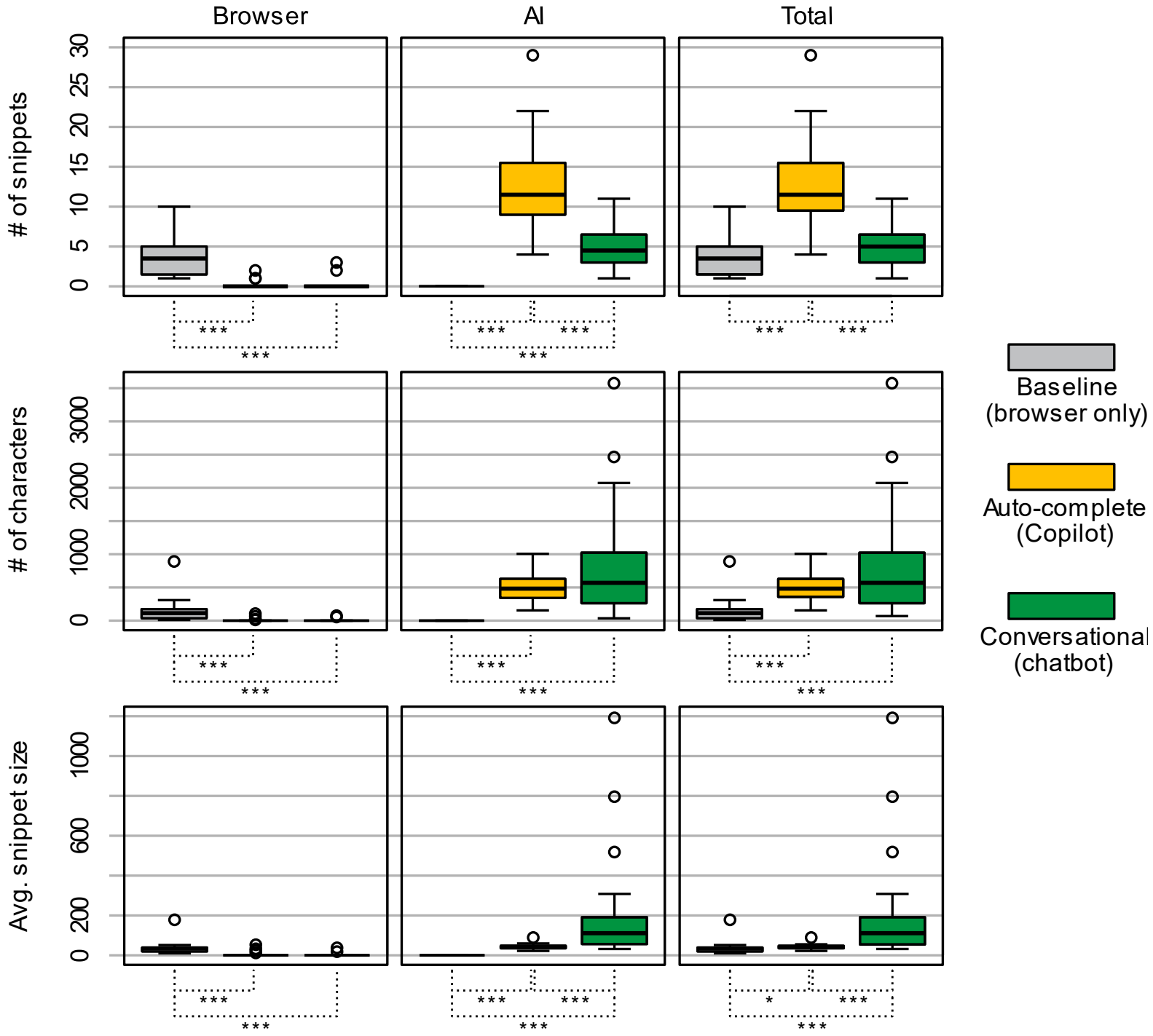
Results


# of requirements implemente per minute
0
1
2
3
***
***
Total time on tasks [sec]
200
600
1000
*
*
Human output [# of characters per minute]
0
100
300
500
***
**
% of total code created by human
1
Baseline (browser only)
Auto-complete (Copilot)
Conversationa (chatbot)

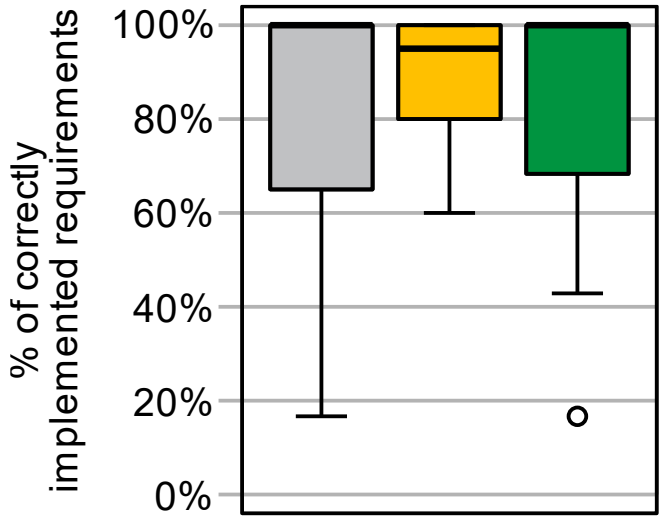
Results

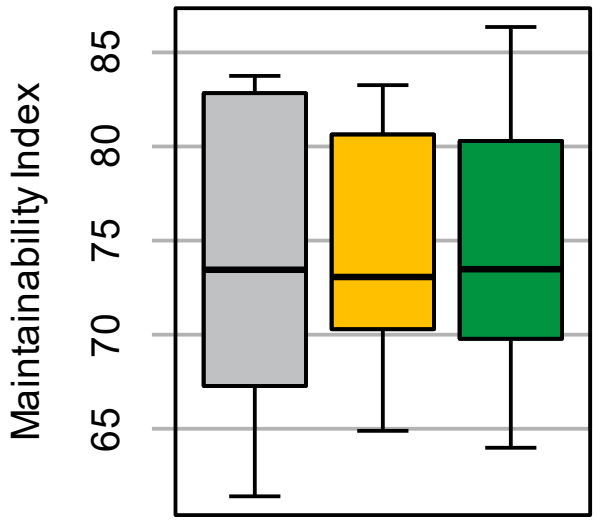

Total time on tasks [sec]
200
600
1000
*
*
Human output [# of characters per minute]
0
100
300
500
***
**
% of total code created by human
0%
20%
40%
60%
80%
100%
***
***
Baseline (browser only)
Auto-complete (Copilot)
Conversationa (chatbot)

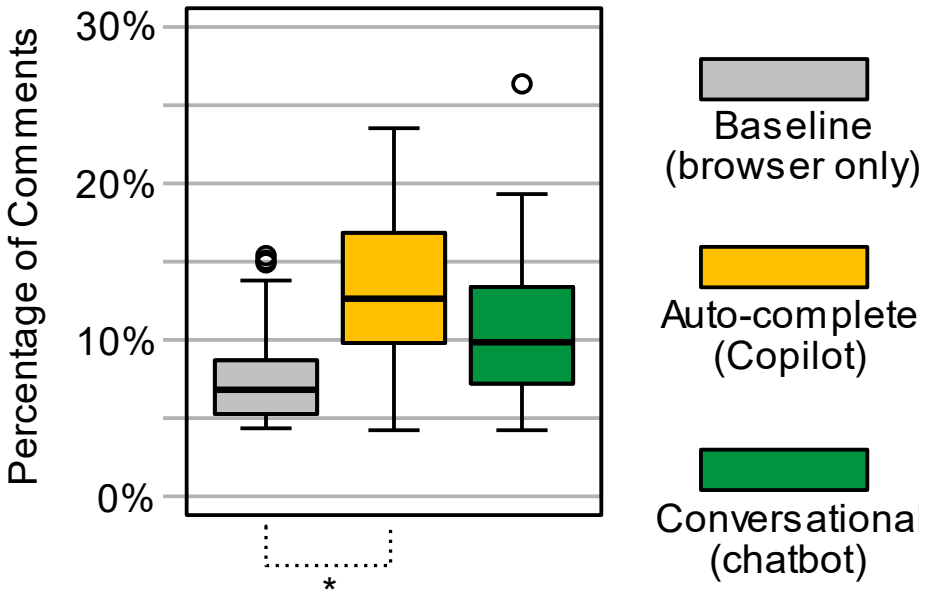
Results


Browser
AI
Total
# of snippets
0
5
10
15
20
25
30
# of characters
0
1000
2000
3000
Avg. snippet size
0
200
600
1000
***
*
Baseline
(browser only)
Auto-complete
(Copilot)
Conversational
(chatbot)

Results


% of correctly implemented requirements
100%
80%
60%
40%
20%
0%


Maintainability Index
85
80
75
70
65


Percentage of Comments
30%
20%
10%
0%
*

Baseline (browser only)
Auto-complete (Copilot)
Conversationa (chatbot)

Consequences

- Human responsibilities shift
  - Coding → Steering, review
  - New/change in skills necessary
- Coding quality averages out (with current models)
- Level of abstraction increases
- Code for AI not for humans

AI Coders Are among Us: Rethinking Programming Language Grammar towards Efficient Code Generation

Zhensu Sun
Singapore Management University
Singapore
zssun@smu.edu.sg

Xiaoning Du*
Monash University
Australia
xiaoning.du@monash.edu

Zhou Yang
Singapore Management University
Singapore
zyang@smu.edu.sg

Li Li
Beihang University
China
lilicoding@ieee.org

David Lo
Singapore Management University
Singapore
davidlo@smu.edu.sg

Abstract

Artificial Intelligence (AI) models have emerged as another important audience for programming languages alongside humans and machines, as we enter the era of large language models (LLMs). LLMs can now perform well in coding competitions and even write programs like developers to solve various tasks, including mathematical problems. However, the grammar and layout of current programs are designed to cater the needs of human developers – with many grammar tokens and formatting tokens being used to make the code easier for humans to read. While this is helpful, such a design adds unnecessary computational work for LLMs, as each token they either use or produce consumes computational resources.

To improve inference efficiency and reduce computational costs, we propose the concept of *AI-oriented grammar*. This aims to represent code in a way that better suits the working mechanism of AI models. Code written with AI-oriented grammar discards formats and uses a minimum number of tokens to convey code semantics effectively. To demonstrate the feasibility of this concept, we explore and implement the first AI-oriented grammar for Python, named Simple Python (SimPy). SimPy is crafted by revising the original Python grammar through a series of heuristic rules. Programs written in SimPy maintain identical Abstract Syntax Tree (AST) structures to those in standard Python. This allows for not only exe- models can maintain or even improve their performance when using SimPy instead of Python for these tasks. With these promising results, we call for further contributions to the development of AI-oriented program grammar within our community.

CCS Concepts

• **Computing methodologies → Artificial intelligence**; **Philosophical/theoretical foundations of artificial intelligence**; • **Theory of computation → Grammars and context-free languages**.

Keywords

Code Generation, Programming Language, Large Language Model

ACM Reference Format:
Zhensu Sun, Xiaoning Du, Zhou Yang, Li Li, and David Lo. 2024. AI Coders Are among Us: Rethinking Programming Language Grammar towards Efficient Code Generation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), September 16–20, 2024, Vienna, Austria.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3650212.3680347

1 Introduction

High-level programming languages like Python, Java, and C++ are

AI Coders Are among Us: Rethinking Programming Language Grammar towards Efficient Code Generation

Zhensu Sun
Singapore Management University

Xiaoning Du*
Monash University

Zhou Yang
Singapore Management University

However, the grammar and layout of current programs are designed to cater the needs of human developers – with many grammar tokens and formatting tokens being used to make the code easier for humans to read
(...)
To improve inference efficiency and reduce computational costs, we propose the concept of **AI-oriented grammar.** This aims to represent code in a way that **better suits the working mechanism of AI models**

resources.

To improve inference efficiency and reduce computational costs, we propose the concept of *AI-oriented grammar*. This aims to represent code in a way that better suits the working mechanism of AI models. Code written with AI-oriented grammar discards formats and uses a minimum number of tokens to convey code semantics effectively. To demonstrate the feasibility of this concept, we explore and implement the first AI-oriented grammar for Python, named Simple Python (SimPy). SimPy is crafted by revising the original Python grammar through a series of heuristic rules. Programs written in SimPy maintain identical Abstract Syntax Tree (AST) structures to those in standard Python. This allows for not only exe-

Code Generation, Programming Language, Large Language Model

ACM Reference Format:
Zhensu Sun, Xiaoning Du, Zhou Yang, Li Li, and David Lo. 2024. AI Coders Are among Us: Rethinking Programming Language Grammar towards Efficient Code Generation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), September 16–20, 2024, Vienna, Austria.* ACM, New York, NY, USA, 13 pages.
https://doi.org/10.1145/3650212.3680347

1 Introduction

High-level programming languages like Python, Java, and C++ are

AI Coders Are among Us: Rethinking Programming Language Grammar towards Efficient Code Generation

Zhensu Sun
Singapore Management University

Xiaoning Du*
Monash University

Zhou Yang
Singapore Management University

However, the grammar and layout of current programs are designed to cater the needs of human developers – with many grammar tokens and formatting tokens being used to make the code easier for humans to read
(…)
To improve inference efficiency and reduce computational costs, we propose the concept of **AI-oriented grammar.** This aims to represent code in a way that **better suits the working mechanism of AI models**

resources.

To improve inference efficiency and reduce computational costs, we propose the concept of *AI-oriented grammar*. This aims to represent code in a way that better suits the working mechanism of AI models. Code written with AI-oriented grammar discards formats and uses a minimum number of tokens to convey code semantics effectively. To demonstrate the feasibility of this concept, we explore and implement the first AI-oriented grammar for Python, named Simple Python (SimPy). SimPy is crafted by revising the original Python grammar through a series of heuristic rules. Programs written in SimPy maintain identical Abstract Syntax Tree (AST) structures to those in standard Python. This allows for not only exe-

Code Generation, Programming Language, Large Language Model

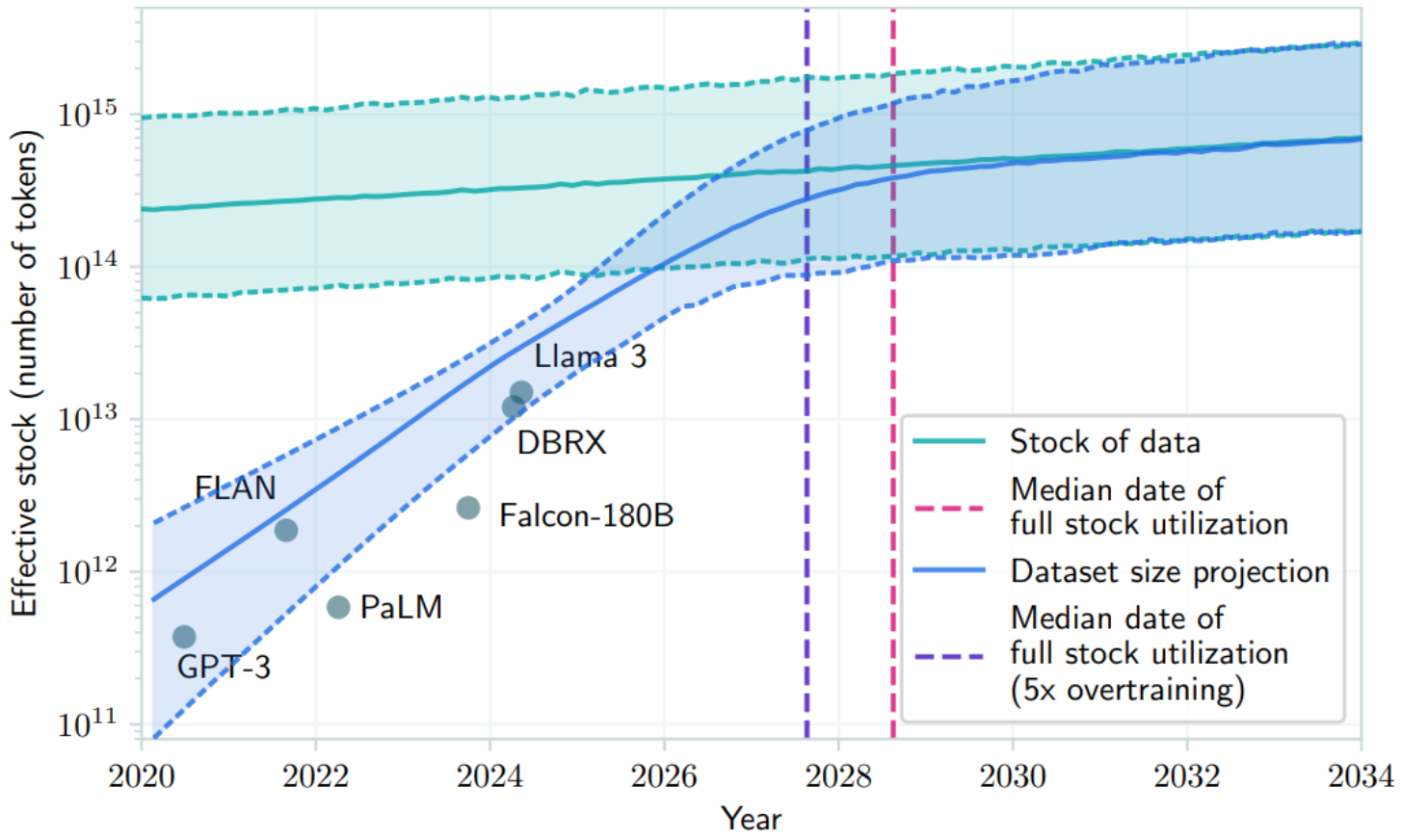
ACM Reference Format:
Zhensu Sun, Xiaoning Du, Zhou Yang, Li Li, and David Lo. 2024. AI Coders Are among Us: Rethinking Programming Language Grammar towards Efficient Code Generation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), September 16–20, 2024, Vienna, Austria.* ACM, New York, NY, USA, 13 pages.
https://doi.org/10.1145/3650212.3680347

1 Introduction

High-level programming languages like Python, Java, and C++ are

Challenges

Challenges – Training Data



Villalobos, P., Ho, A., Sevilla, J., Besiroglu, T., Heim, L., & Hobbhahn, M. Position: Will we run out of data? Limits of LLM scaling based on human-generated data. In *Forty-first International Conference on Machine Learning*.

Challenges

- Unclear requirements
- Unclear how well LLMs can determine code quality [1]
- What if the code does not work? → Debugging
- Lack of reflection, learning, and problem-solving
- Licensed code in the training data and output
- Security, Reliability, etc.

[1] Simões, I. R. D. S., & Venson, E. (2024). Evaluating Source Code Quality with Large Language Models: a comparative study. *arXiv preprint arXiv:2408.07082*.

Conclusion

Questions?

Contact

Thomas Weber
LMU Munich
thomas.weber@ifi.lmu.de