



The State of Agentic Coding: How Close Have AI Agents Come to Human Developers?



Yutaro Kashiwa

Associate professor

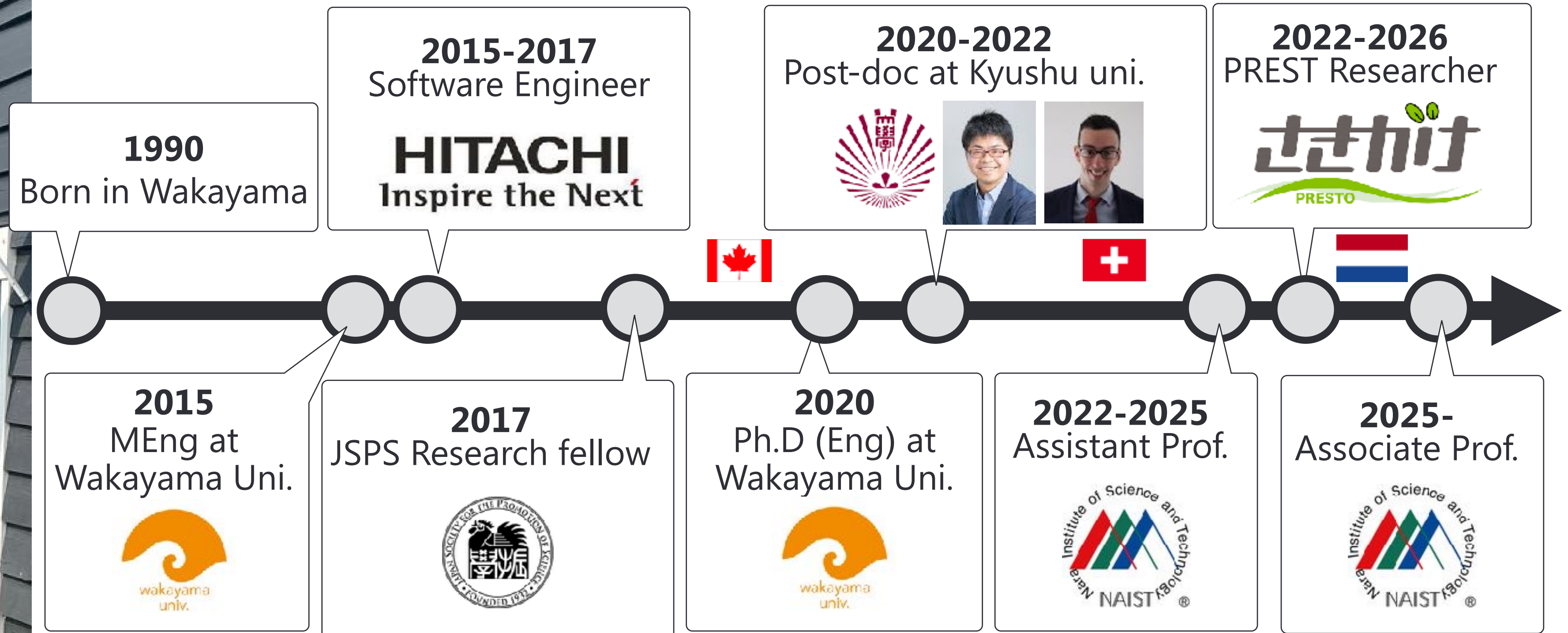
Nara Institute of Science and Technology (NAIST)

Self-introduction



Yutaro Kashiwa
(柏 祐太郎)

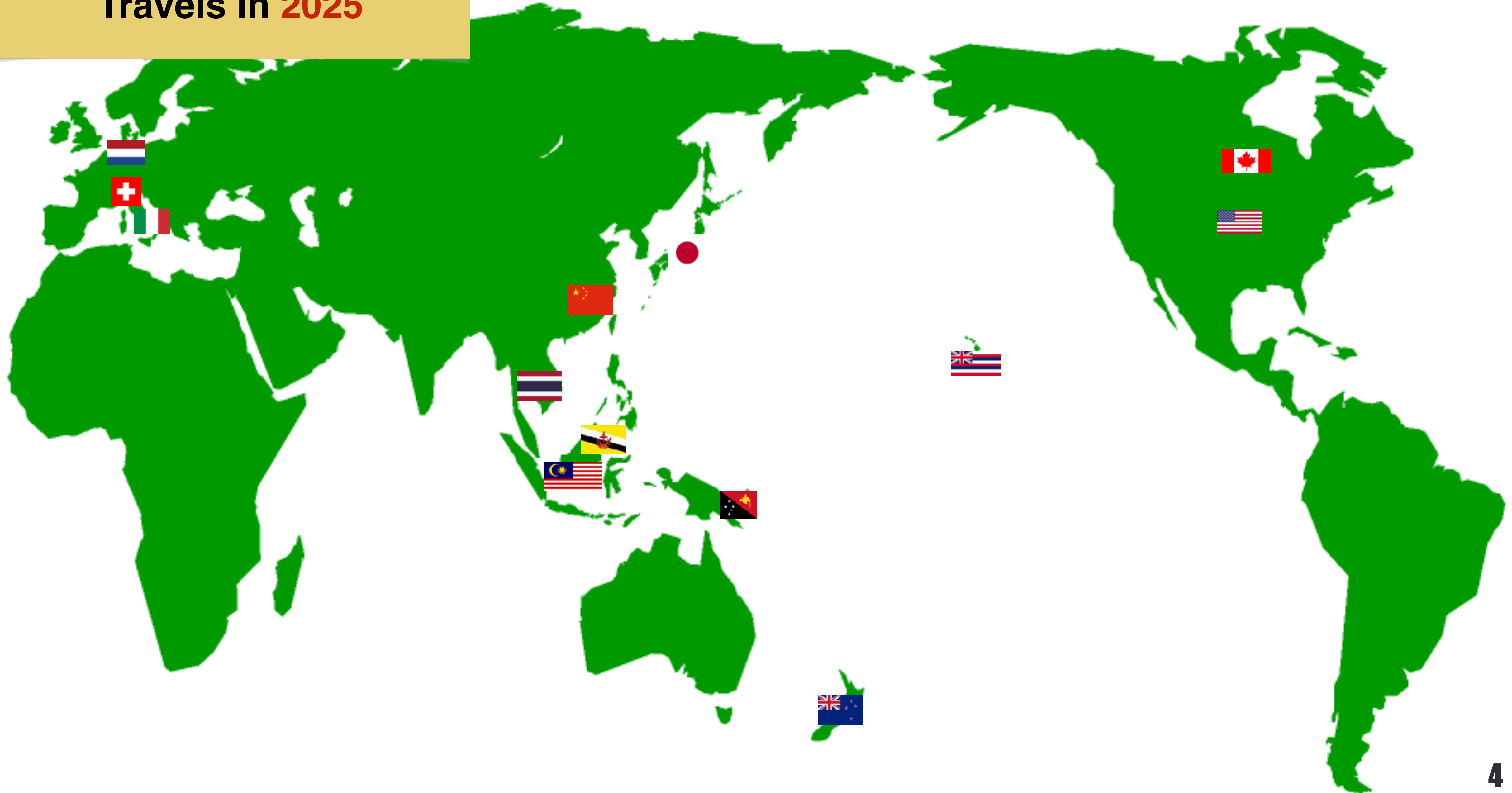
- Research career



- Hobby: Soccer, Bowling, Fishing, Tennis, **Traveling**



Travels in 2025



Travels in **my life**



In Total

17%
World



34
Countries

Out of 195 UN Countries



11 hours

Nara



Tokyo



Osaka



Kyoto



West Japan has more attractive places

Promoting World-Class Research

NAIST ITC

Administrative staff

374

Postgraduates

1,043

Location

Ikoma (Kansai Science City),
Nara, Japan

Campus

Suburban,
139,967 m²

Mascot

None

Website

www.naist.jp



WIKIPEDIA
The Free Encyclopedia

40% are international students

All classes are in English

Software Design and Analysis Lab.



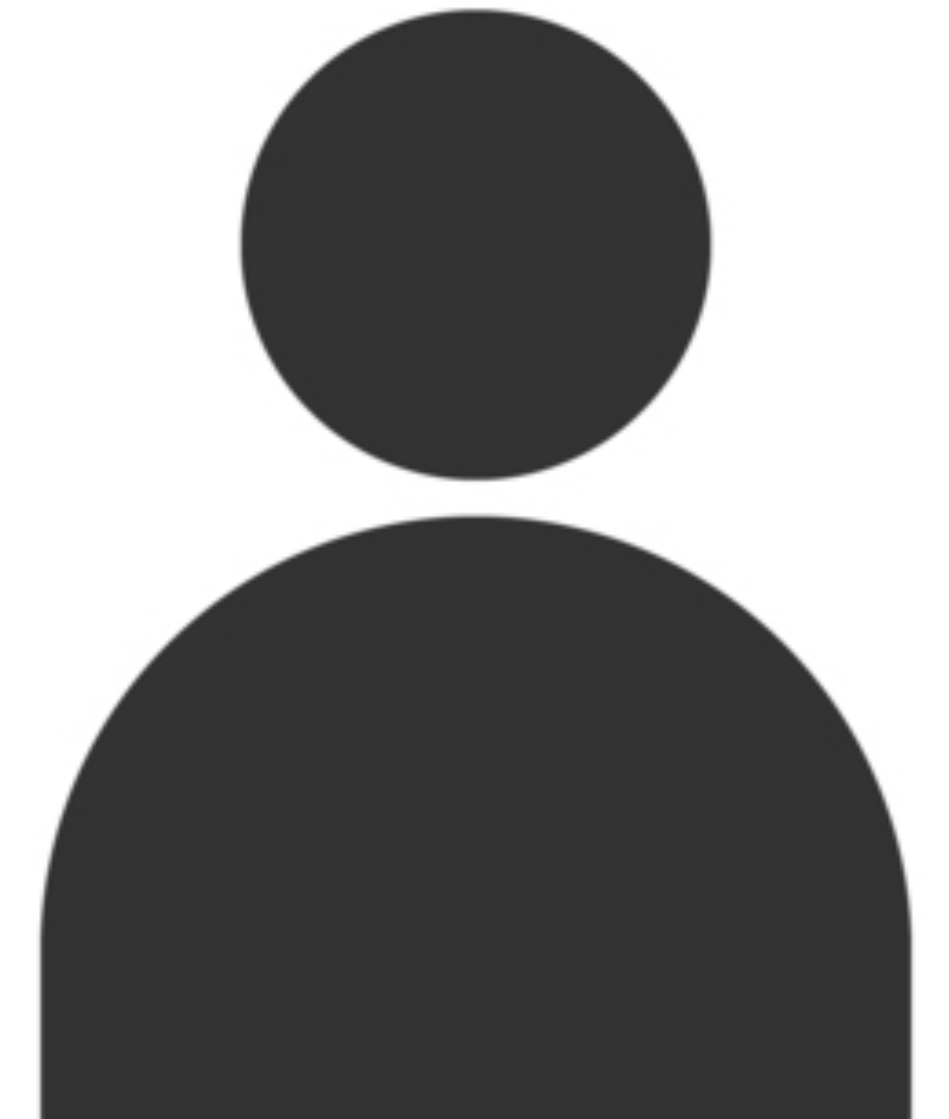
Professor
Hajimu IIDA



Associate Professor
Yutaro KASHIWA



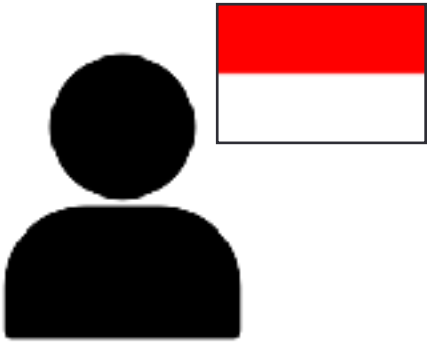
Assistant Professor
Brittany Anne REID



Assistant Professor
[Coming soon]

Students

Doctoral course



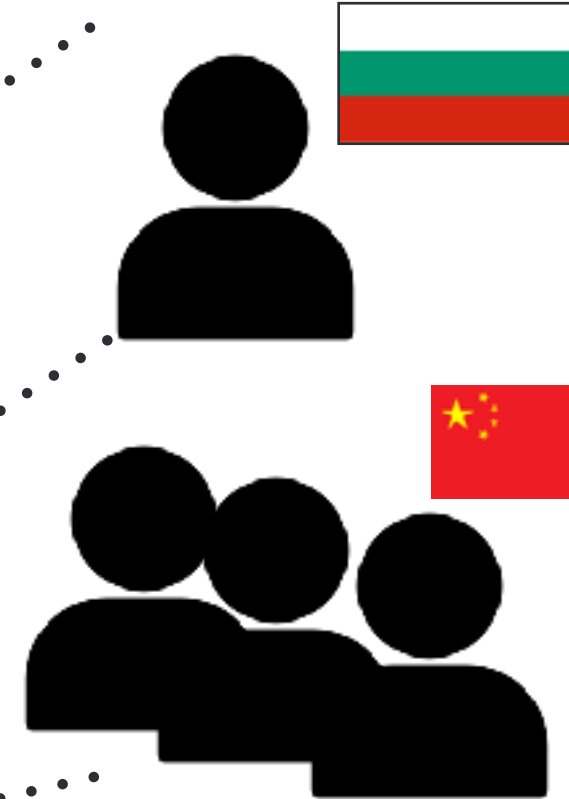
Master's course



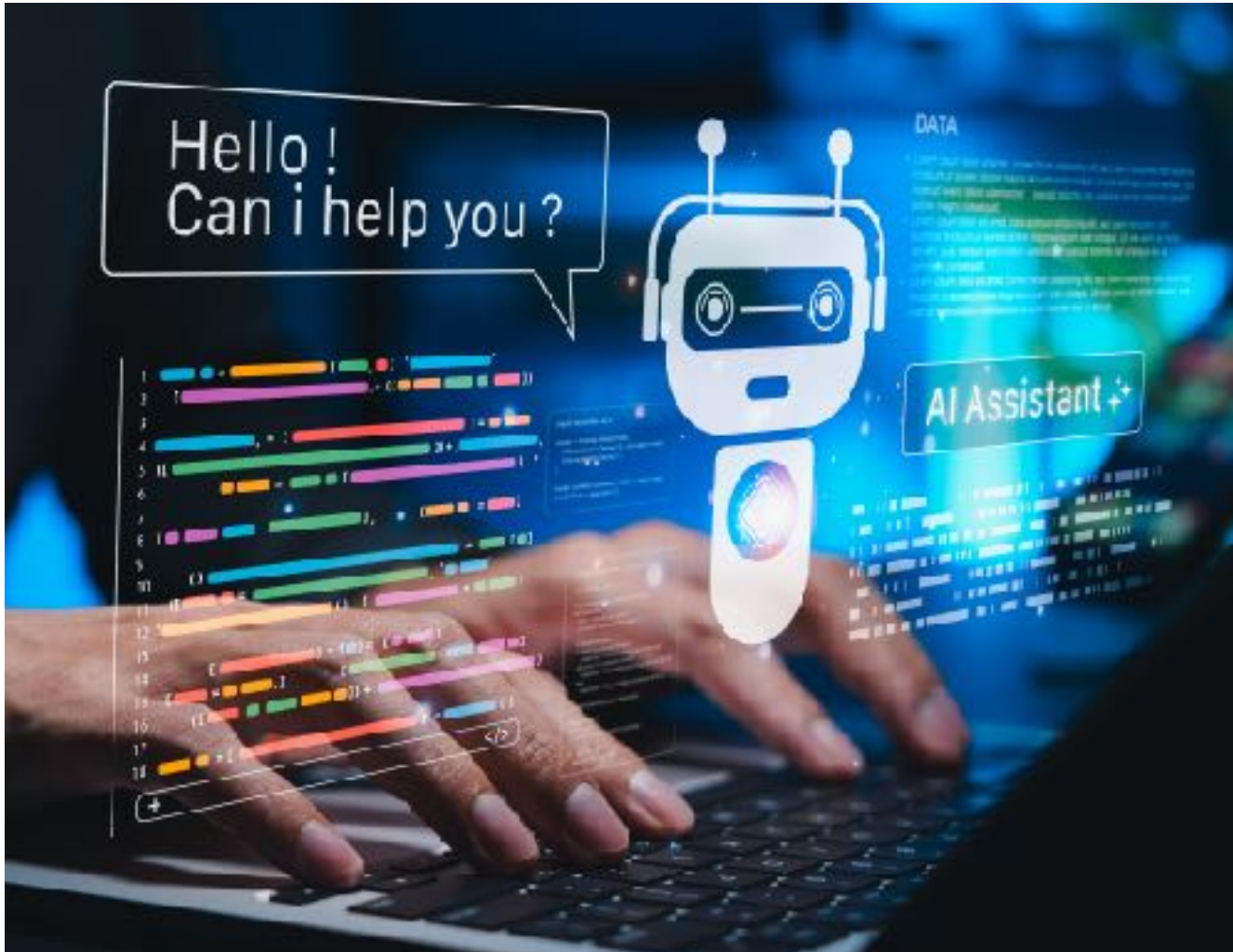
M2 (11)



M1 (14)



Research interests:



Software Development Acceleration with AI



Analyzing Productivity of Software Development AI



Testing for AI-assisted Software Development

Co-founder of OpenAI



Andrej Karpathy 

@karpathy



The hottest new programming language is 

5:14 AM · Jan 25, 2023 · **9.5M** Views

 1.5K

 8.5K

 54K

 5.6K

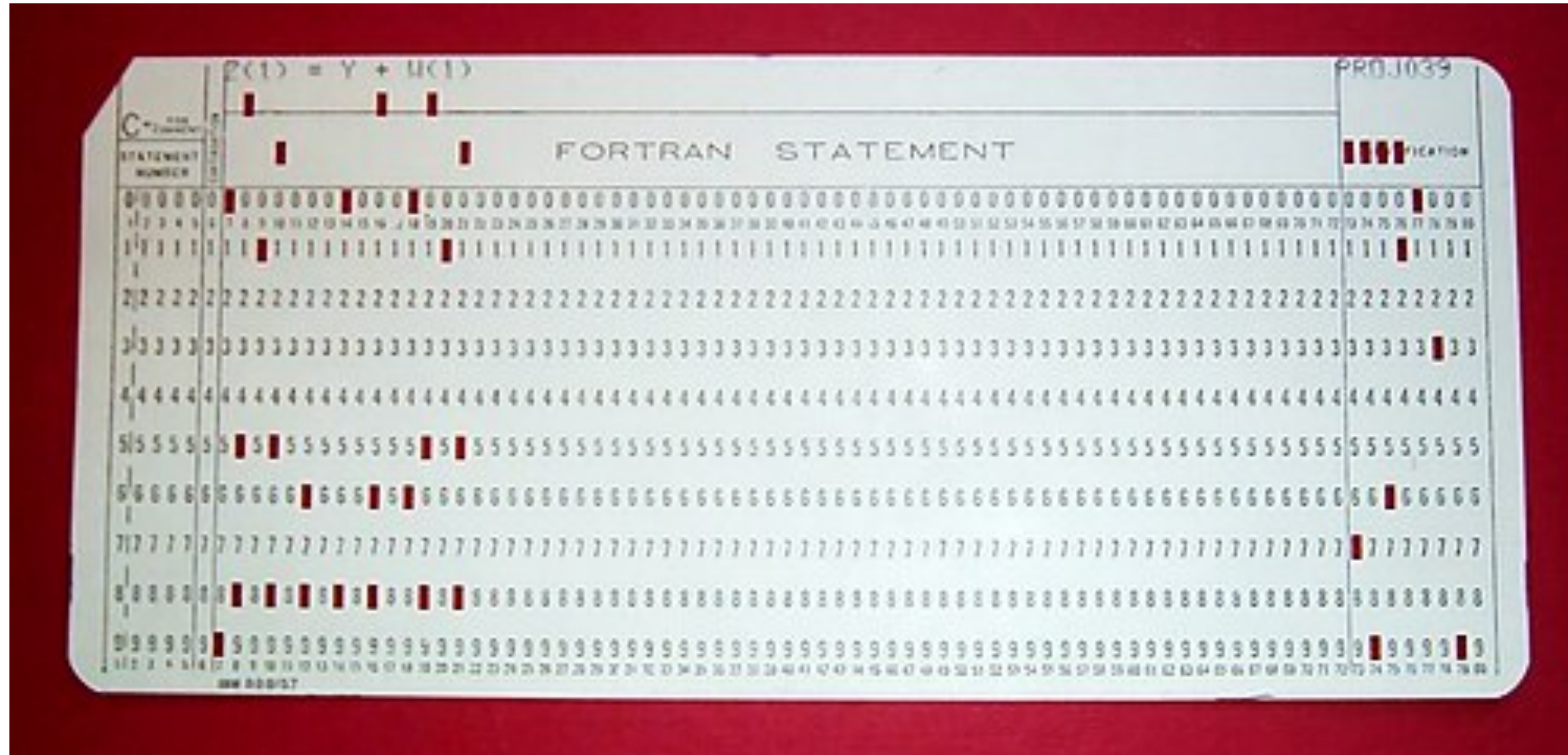


The background features a collage of overlapping clock faces. Large, bold, 3D-style numbers representing years from 2017 to 2021 are scattered across the scene. A semi-transparent white rectangular box is centered horizontally, containing the text 'How does programming style change?'.

How does programming style change?

Punched cards

- Punch holes in paper cards, with **one card representing one line of code**



- » Bring the stack of cards to a computing center and hand them to an operator
- » Results returned as printouts hours or even days later

**If you have mistakes,
you have to recreate the whole card**



Machine code

- **The only language CPUs can directly understand and execute (Fastest execution speed)**
 - » Composed entirely of binary digits (0s and 1s)

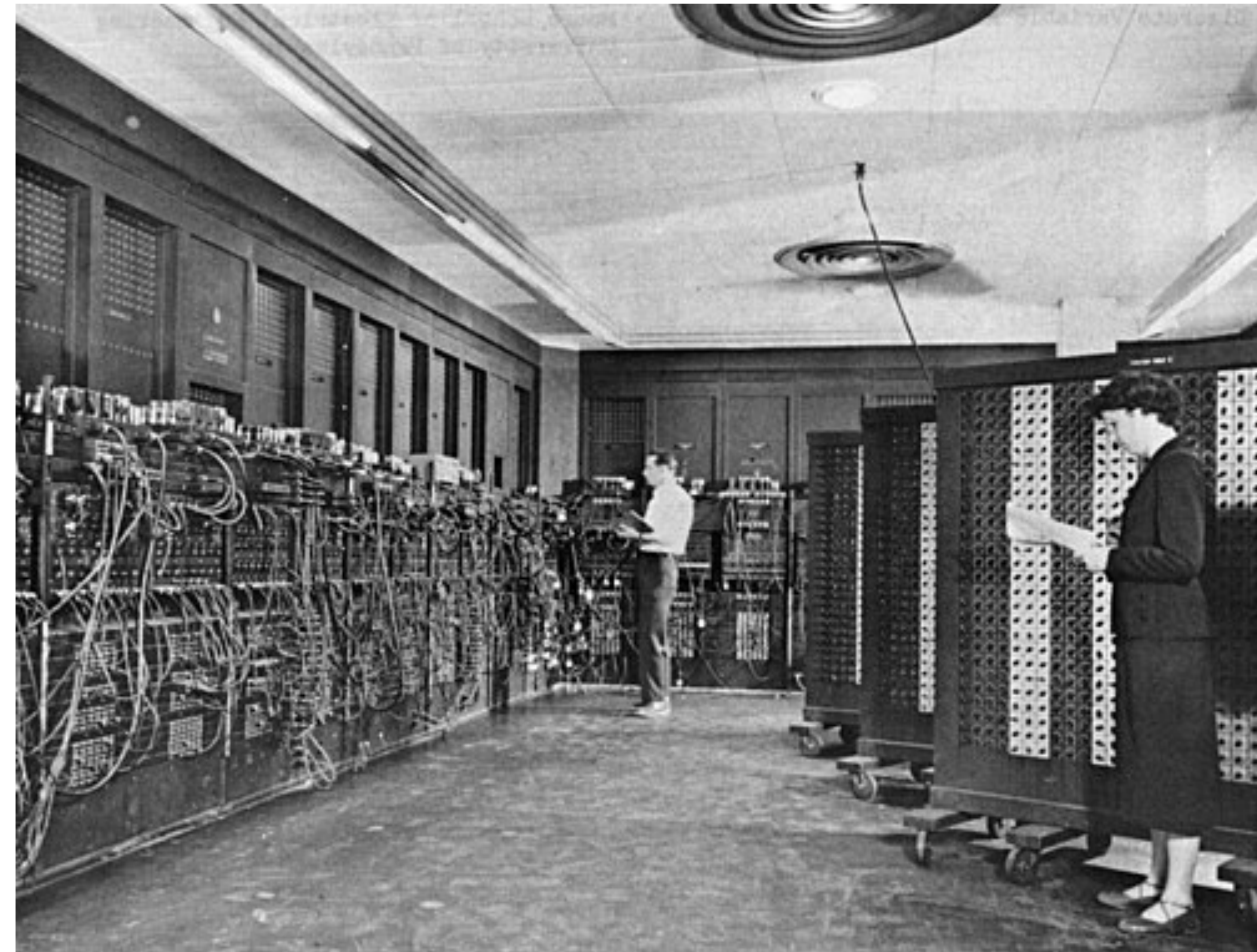
```
1 FOX 12:01a 23- 1
A 002000 C2 30 REP #$30
A 002002 18 CLC
A 002003 F8 SED
A 002004 A9 34 12 LDA #$1234
A 002007 69 21 43 ADC #$4321
A 00200A 8F 03 7F 01 STA $017F03
A 00200E D8 CLD
A 00200F E2 30 SEP #$30
A 002011 00 BRK
A 2012

r
PB PC NUmxDI2C .A .X .Y SP DP DB
; 00 E012 00110000 0000 0000 0002 CFFF 0000 00
g 2000

BREAK

PB PC NUmxDI2C .A .X .Y SP DP DB
; 00 2013 00110000 5555 0000 0002 CFFF 0000 00
m 7f03 7f03
>007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00:UU.....
```

**It is very difficult for human to understand
(Very low development efficiency)**



Assembly language

- Replaces machine code with human-readable symbols
 - » Converted to machine code by an assembler

```
MONITOR FOR 6802 1.4                9-14-80  TSC ASSEMBLER  PAGE    2
```

```
C000                                ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START  LDS    #STACK
```

```
*****
* FUNCTION: INITA - Initialize ACIA
* INPUT: none
* OUTPUT: none
* CALLS: none
* DESTROYS: acc A
```

```
RESETA EQU    %00010011
CTLREG EQU    %00010001
```

```
C003 86 13  INITA  LDA  A  #RESETA  RESET ACIA
```

Assigns a constant value to a symbol

Requires many lines of code; development is time-consuming

High-level programming language

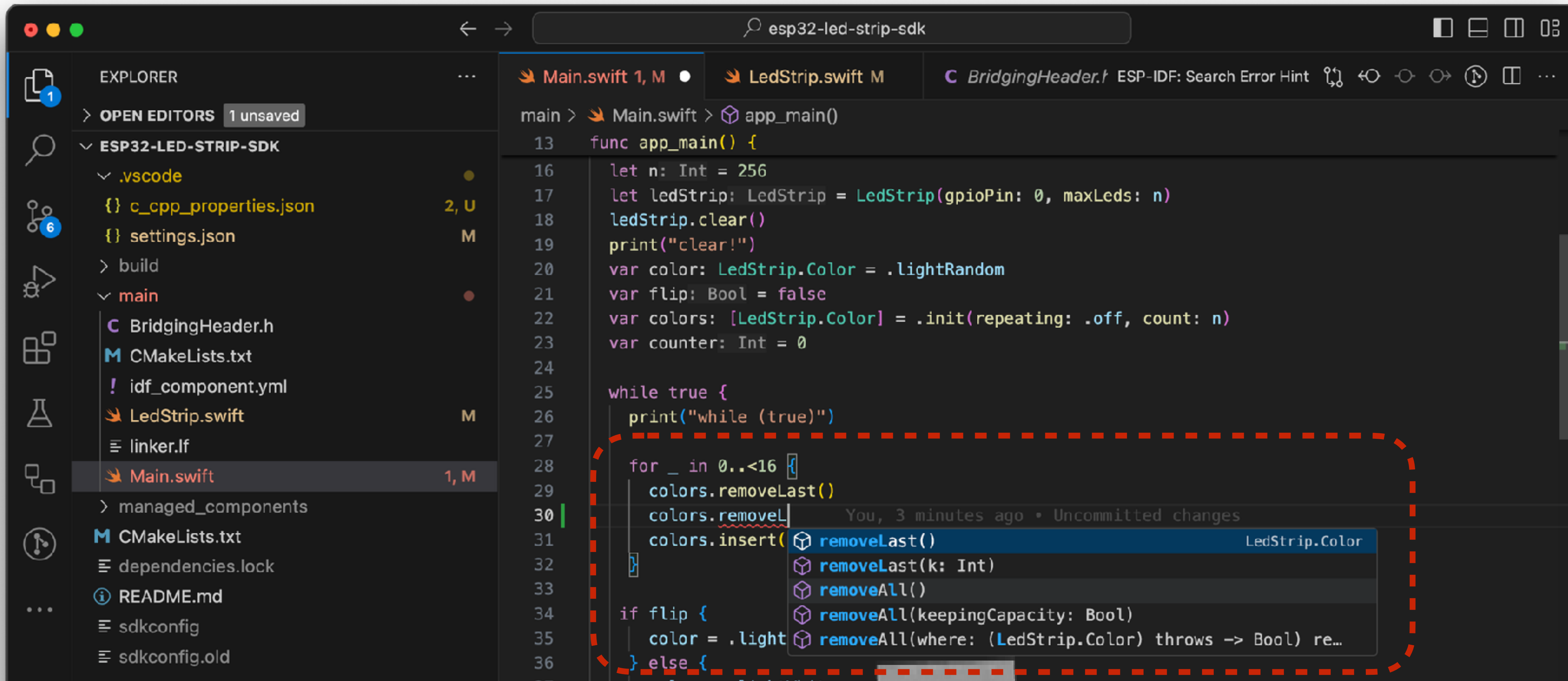


- **Compilers translate source code into machine code automatically**
 - » Languages designed to be closer to human thinking
 - » Improved portability across different machines

```
1 /*
2  * This line basically imports the "stdio" header file, part
3  * the standard library. It provides input and output functi
4  * to the program.
5  */
6 #include <stdio.h>
7
8 /*
9  * Function (method) declaration. This outputs "Hello, world\n
10 * standard output when invoked.
11 */
12 void sayHello(void) {
13     // printf() in C outputs the specified text (with optional
14     // formatting options) when invoked.
15     printf("Hello, world!\n");
16 }
17
18 /*
19 * This is a "main function". The compiled program will run the code
20 * defined here.
21 */
22 int main(void)
23 {
24     // Invoke the sayHello() function.
25     sayHello();
26     return 0;
27 }
```



IDE: Integrate Development Environment



The screenshot shows a code editor with the following Swift code:

```
main > Main.swift > app_main()
13 func app_main() {
16     let n: Int = 256
17     let ledStrip: LedStrip = LedStrip(gpioPin: 0, maxLeds: n)
18     ledStrip.clear()
19     print("clear!")
20     var color: LedStrip.Color = .lightRandom
21     var flip: Bool = false
22     var colors: [LedStrip.Color] = .init(repeating: .off, count: n)
23     var counter: Int = 0
24
25     while true {
26         print("while (true)")
27
28         for _ in 0..<16 {
29             colors.removeLast()
30             colors.removeLast()
31             colors.insert(
32                 removeLast()
33                 removeLast(k: Int)
34                 removeAll()
35                 removeAll(keepingCapacity: Bool)
36                 removeAll(where: (LedStrip.Color) throws -> Bool) re...
```

A red dashed box highlights the code block from line 28 to 36. A dropdown menu is open over the code, showing completion suggestions for the `removeLast()` method. The suggestions include:

- `removeLast()` (selected)
- `removeLast(k: Int)`
- `removeAll()`
- `removeAll(keepingCapacity: Bool)`
- `removeAll(where: (LedStrip.Color) throws -> Bool) re...`

Once typing some, the following commands appear

Prompts (Chat-based)

Interaction with AI using natural language

- » No programming syntax required
- » Large Language Models (LLMs) interpret and execute requests

Cannot Decompose Complex Tasks

They Cannot Use Tools

Humans Must Be the "Hands and Feet"

How to make a login function in Python?

A "login function" can mean different things depending on context (console app, GUI, or web app). I'll start with a **simple, clear console-based example**, then briefly show how it scales to better practices.

1. Very simple login function (console)

This checks a username and password against stored values.

python

Copy code

```
def login(username, password):
    correct_username = "admin"
    correct_password = "1234"

    if username == correct_username and password == correct_password:
        return True
    else:
        return False
```



Prompts (Agent-based)

AI systems that autonomously plan and execute multi-step tasks

- **Example:** "Create a login function"
With just the instruction, the AI will:

1. Analyze the requirements
2. Determine the file structure
3. Generate the code
4. Run tests
5. Self-correct any errors

They Use Tools



Welcome to Claude Code

> refactor the code

- Let me first explore the structure and identify what needs refactoring.
- **List** (Projects/your_project)...
 - └ Listed 12 paths (ctrl+r to expand)
- **Task** (Find all Python files)...
 - └ Done (9 tool uses · 12.9k tokens · 1m 4.9s)
- **Read** (Projects/your_project/your_file.py)...
 - └ Read 97 lines (ctrl+r to expand)
- Now I'll refactor the code to fix all the identified issues.
- **Update** (Projects/your_project/your_file.py)...
 - └ Updated your_file.py with 31 additions and 74 removals
 - 1 import pandas
 - 2 import csv
 - ... (content of the code changes)
- Refactoring Complete
 - I've successfully refactored the code with the following improvements:
 1. Removed unused import (csv)
 2. Extracted common code as helper methods
 - ... (content of the summarized changes)

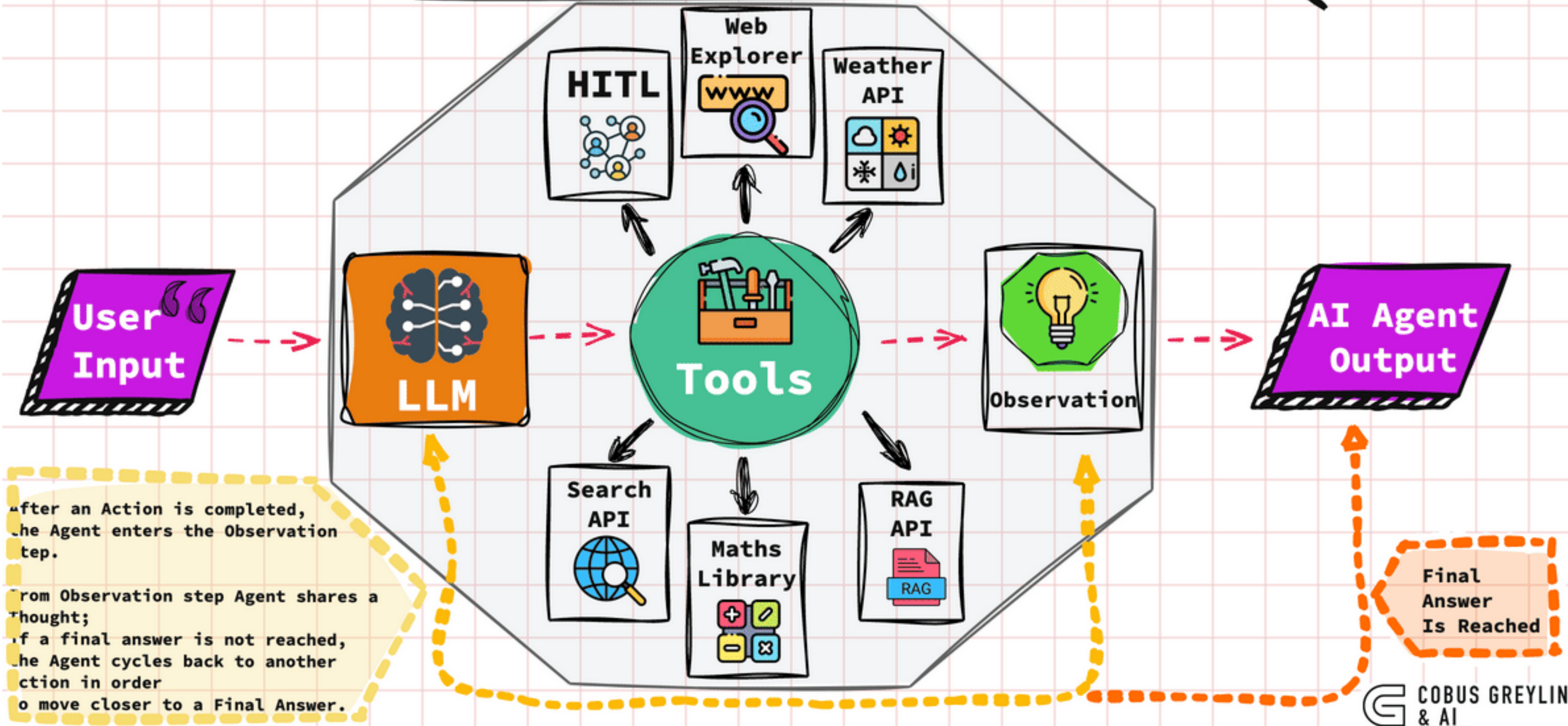
> commit my changes and create a PR



Processing... (1s · esc to interrupt)




AI AGENTS - AGENTIC APPLICATIONS



After an Action is completed, the Agent enters the Observation step.

From Observation step Agent shares a thought; if a final answer is not reached, the Agent cycles back to another action in order to move closer to a Final Answer.

AI Coding Assistant Statistics & Trends [2025]

Matt Li  Oct 16, 2025  Artificial intelligence

AI Coding Assistant Statistics & Trends [2025]

Matt Li



Oct 16, 2025



Artificial intelligence

Productivity and Efficiency Impact of AI Coding Assistants

- Developers save between 30–75% of their time on coding, testing, and documentation tasks when using AI coding assistant tools.
- M365 Copilot users save an average of 30 minutes per week on email and complete documents 12% faster.

On the Use of Agentic Coding: An Empirical Study of Pull Requests on GitHub

[MIKU WATANABE](#), Nara Institute of Science and Technology, Japan

[HAO LI](#), Queen's University, Canada

[YUTARO KASHIWA](#), Nara Institute of Science and Technology, Japan

[BRITTANY REID](#), Nara Institute of Science and Technology, Japan

[HAJIMU IIDA](#), Nara Institute of Science and Technology, Japan

[AHMED E. HASSAN](#), Queen's University, Canada

Large language models (LLMs) are increasingly being integrated into software development processes. The ability to generate code and submit pull requests with minimal human intervention, through the use of autonomous AI agents, is poised to become a standard practice. However, little is known about the practical usefulness of these pull requests and the extent to which their contributions are accepted in real-world projects.

In this paper, we empirically study 567 GitHub pull requests (PRs) generated using Claude Code, an agentic coding tool, across 157 diverse open-source projects. Our analysis reveals that developers tend to rely on agents for tasks such as refactoring, documentation, and testing. The results indicate that 83.8% of these agent-assisted PRs are eventually accepted and merged by project maintainers, with 54.9% of the merged PRs are integrated without further modification. The remaining 45.1% require additional changes benefit from human revisions, especially for bug fixes, documentation, and adherence to project-specific standards. These findings suggest that while agent-assisted PRs are largely acceptable, they still benefit from human oversight and refinement.

CCS Concepts: • **Software and its engineering** → **Integrated and visual development environments; Automatic programming**; *Software evolution; Maintaining software.*

Additional Key Words and Phrases: Agentic Coding, Coding Agent, Pull Requests, Model Context Protocol, Large Language Models

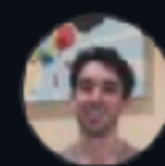
ACM Reference Format:

Miku Watanabe, Hao Li, Yutaro Kashiwa, Brittany Reid, Hajimu Iida, and Ahmed E. Hassan. 2025. On the Use of Agentic Coding: An Empirical Study of Pull Requests on GitHub. 1, 1 (December 2025), 24 pages. <https://doi.org/XXXXXXXX.XXXXXXX>

Move from wpa_supplicant to NetworkManager a

 Merged dbieber merged 17 commits into `main` from `claude-20250414-212357` on Apr 15

 Conversation 11  Commits 17  Checks 2  Files changed 3



[dbieber](#) commented on Apr 15


Summary

- Replace `wpa_supplicant` with `NetworkManager` for WiFi management
- Add migration script and documentation for existing users


Test plan


- Test WiFi network configuration through settings UI
- Test WiFi commands (add, remove, list networks)
- Test `wifi-scan` command to scan for available networks
- Test migration from `wpa_supplicant` to `NetworkManager` using the script
- Verify installation instructions are accurate


Let's move away from `wpa_supplicant` in favor of `NetworkManager` and `nmcli`.

 Generated with [Claude Code](#)



 [dbieber](#) and others added 2 commits [last month](#)

 [Replace wpa_supplicant with NetworkManager for WiFi management](#) ...

 [Fix indentation to use 2 spaces to match surrounding code](#)

On the Use of Agentic Coding: An Empirical Study of Pull Requests on GitHub

MIKU WATANABE, Nara Institute of Science and Technology, Japan
HAO LI, Queen's University, Canada
YUTARO KASHIWA, Nara Institute of Science and Technology, Japan
BRITTANY REID, Nara Institute of Science and Technology, Japan
HAJIMU IIDA, Nara Institute of Science and Technology, Japan
AHMED E.

Large language ability to get autonomous usefulness of
In this pap coding tool, agents for ta agent-assiste are integrate human revisi findings sugg and refine
CCS Concept Automatic I
Additional K Large Langu
ACM Refere Miku Watan Use of Agen
<https://doi.org>

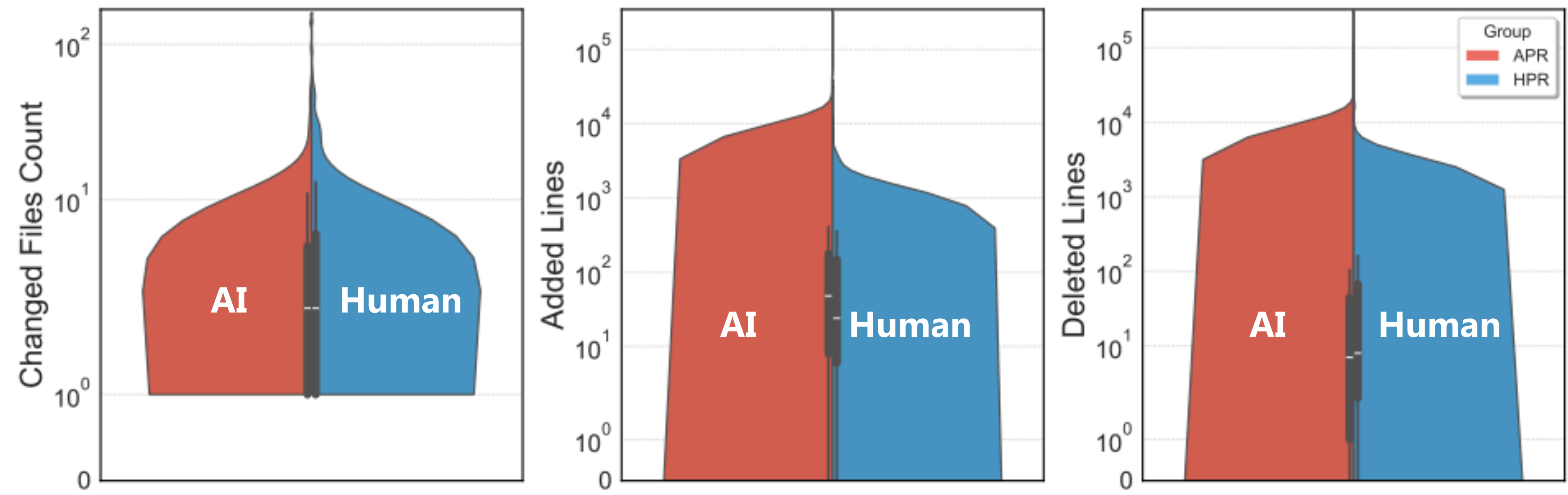
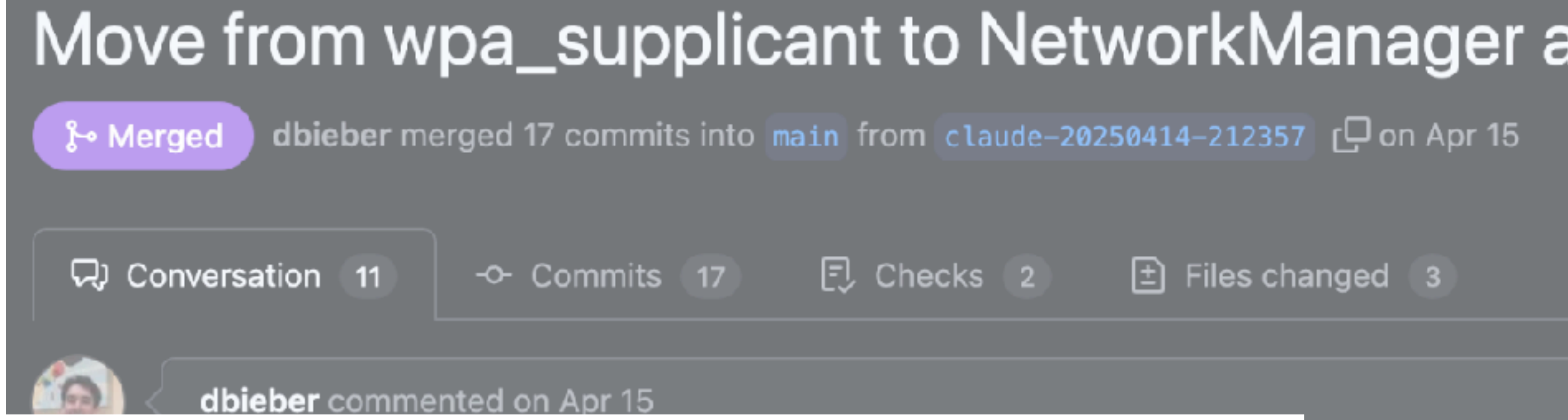


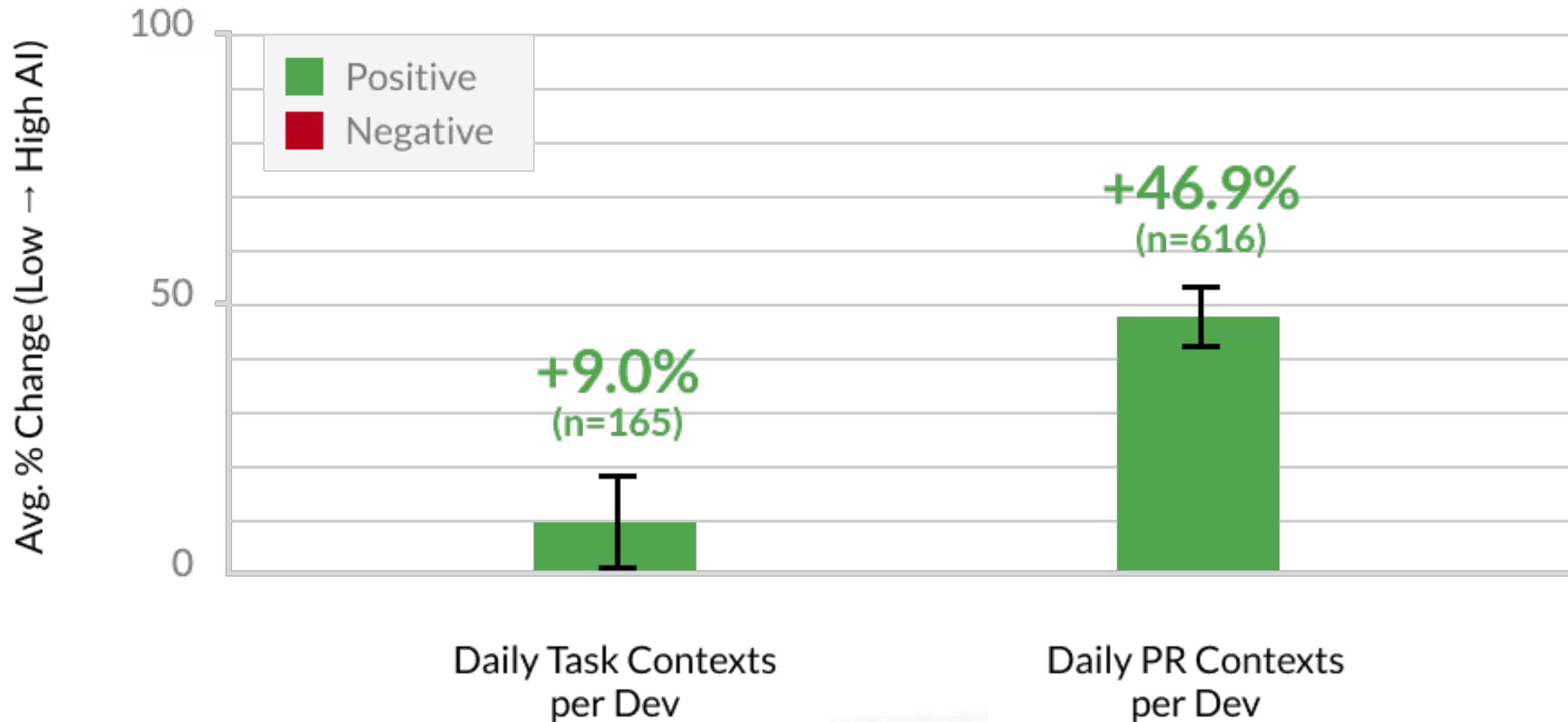
Fig. 4. Distribution of change metrics including changed files, added lines, and deleted lines in revised commits. Note that these do not include metrics from the first commit.

AI generate more code than Human

Fix indentation to use 2 spaces to match surrounding code

Developers juggle more workstreams as AI usage grows

Multi-tasking metrics: % change from low to high AI adoption



Teams using AIs complete 46.9% more tasks

BUSINESS INSIDER

Microsoft pushes staff to use internal AI tools more, and may consider this in reviews. 'Using AI is no longer optional.'

By [Ashley Stewart](#)

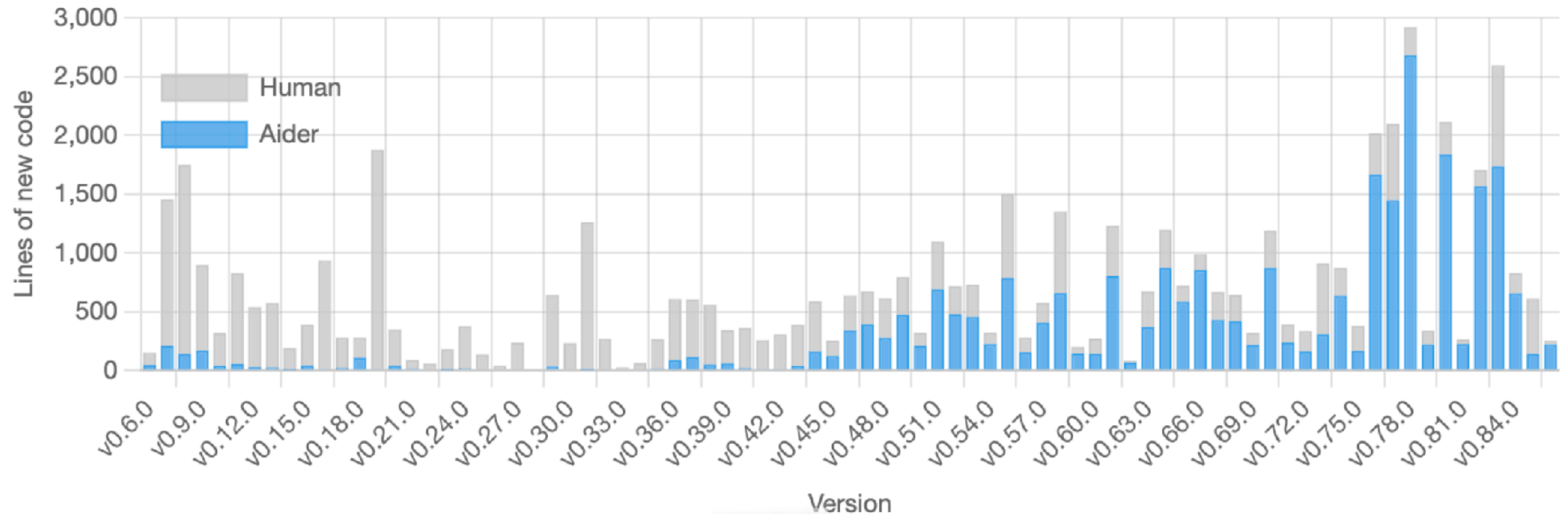
+ Follow



Julia Liuson, Developer Division corporate vice president at Microsoft Microsoft



Lines of new code, by release



AI generates AI by themselves



Do Agents generate perfect code?

Do we need the maintenance for agent-generated code?

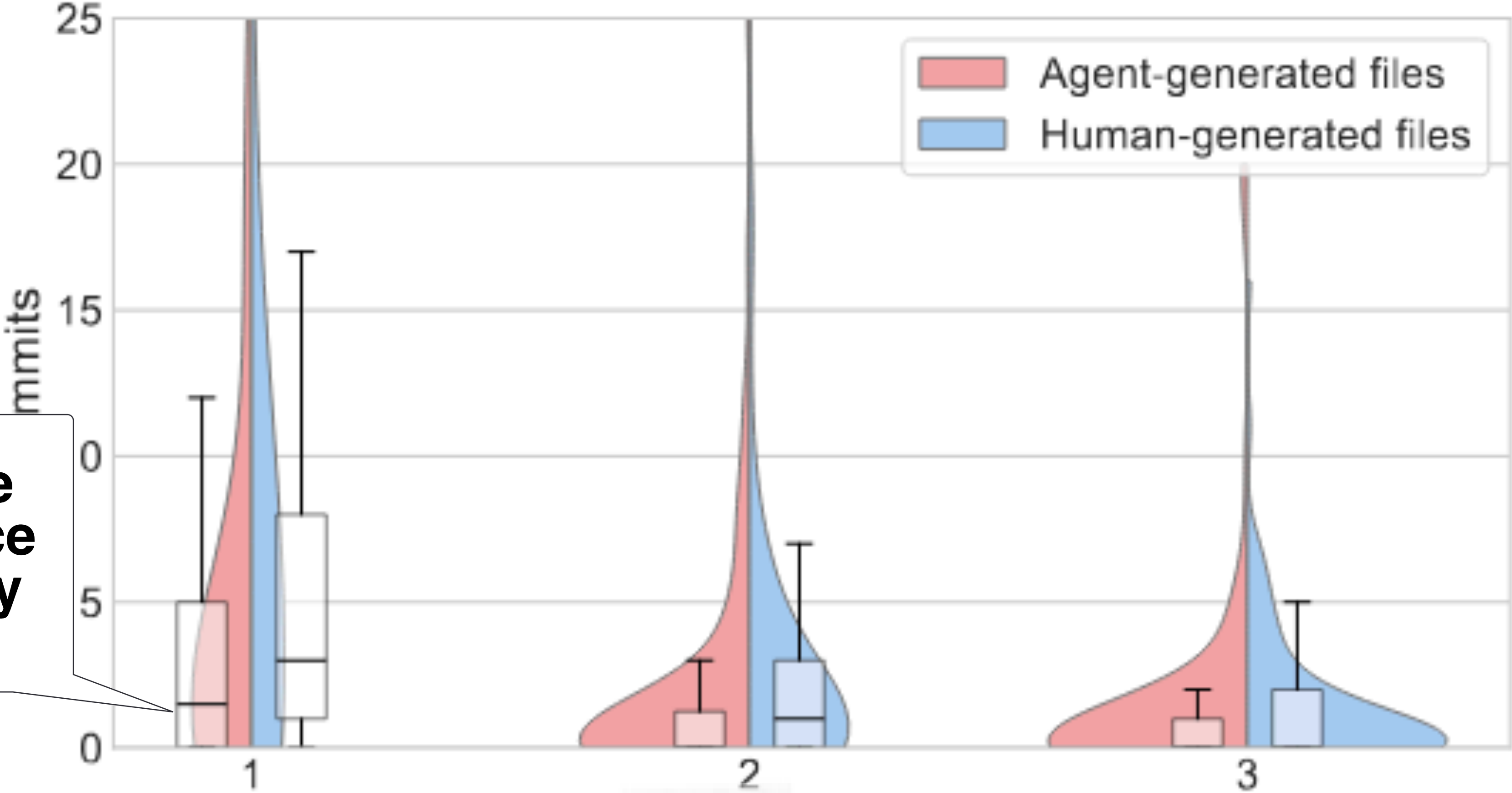
Table 1: Types of maintenance activities for Human-generated files and AI-generated files

Category	Description	Human	AI	Diff (%)
feat	Addition of new features	219 (14.86%)	269 (22.00%)	+7.14
build	Changes related to the build system	115 (7.80%)	125 (10.22%)	+2.42
style	Code style changes	53 (3.60%)	63 (5.15%)	+1.55
chore	Miscellaneous tasks or other changes	168 (11.40%)	139 (11.37%)	-0.03
perf	Performance improvements	33 (2.24%)	27 (2.21%)	-0.03
ci	Changes related to continuous integration (CI)	76 (5.16%)	62 (5.07%)	-0.09
test	Addition or improvement of tests	95 (6.45%)	71 (5.81%)	-0.64
refactor	Code refactoring (internal structural improvements)	234 (15.88%)	177 (14.47%)	-1.41
docs	Documentation-only changes	235 (15.94%)	142 (11.61%)	-4.33
fix	Bug fixes	243 (16.49%)	139 (11.37%)	-5.12

Note: Bold numbers indicate the most frequent category in the Human/AI columns and the largest differences in the Diff column.

Human-generated code has more bugs

Do we need the maintenance for agent-generated code?



86% of the maintenance are done by Human

We still need a lot of Human Efforts

RepairAgent: An Autonomous, LLM-Based Agent for Program Repair

Islem Bouzenia
University of Stuttgart
Germany
fi_bouzenia@esi.dz

Premkumar Devanbu
UC Davis
USA
ptdevanbu@ucdavis.edu

Michael Pradel
University of Stuttgart
Germany
michael@binaervarianz.de

Abstract—Automated program repair has emerged as a powerful technique to mitigate the impact of software bugs on system reliability and user experience. This paper introduces RepairAgent, the first work to address the program repair challenge through an autonomous agent based on a large language model (LLM). Unlike existing deep learning-based approaches, which prompt a model with a fixed prompt or in a fixed feedback loop, our work treats the LLM as an agent capable of autonomously planning and executing actions to fix bugs by invoking suitable tools. RepairAgent freely interleaves gathering information about the bug, gathering repair ingredients, and validating fixes, while deciding which tools to invoke based on the gathered information and feedback from previous fix attempts. Key contributions that enable RepairAgent include a set of tools that are useful for program repair, a dynamically updated prompt format that allows the LLM to interact with these tools, and a finite state machine that guides the agent in invoking the tools. Our evaluation on the popular Defects4J dataset demonstrates RepairAgent’s effectiveness in autonomously repairing 164 bugs, including 39 bugs not fixed by prior techniques. Interacting with the LLM

The current state-of-the-art in APR predominantly revolves around large language models (LLMs). The first generation of LLM-based repair uses a one-time interaction with the model, where the model receives a prompt containing the buggy code and produces a fixed version [17], [18]. The second and current generation of LLM-based repair introduces iterative approaches, which query the LLM repeatedly based on feedback obtained from previous fix attempts [19], [20], [21].

A key limitation of current iterative, LLM-based repair techniques is that their hard-coded feedback loops do not allow the model to gather information about the bug or existing code that may provide ingredients to fix the bug. Instead, these approaches fix the code context that is provided to the LLM. For example, in [19], [21], the test cases that fail are used to generate a new prompt that executes the tests on different code snippets.

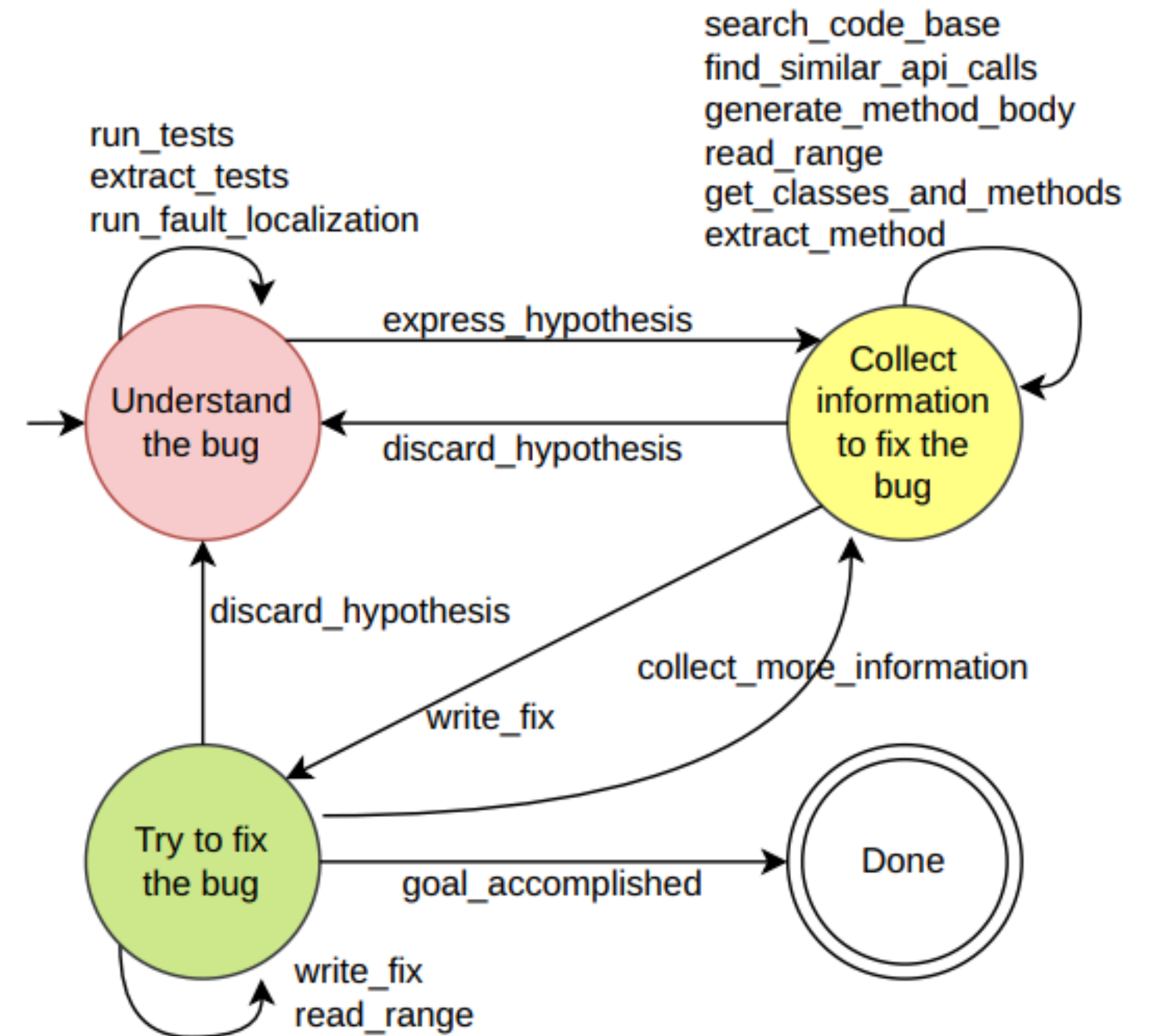


Fig. 2: State machine to guide selection of tools.

Only **20%** of bugs in the benchmark were fixed

70% of the fixes they could fix are small (one line)

What's holding back AI's productivity boost? | It's not the model—it's your system. GAINS™ reveals why →

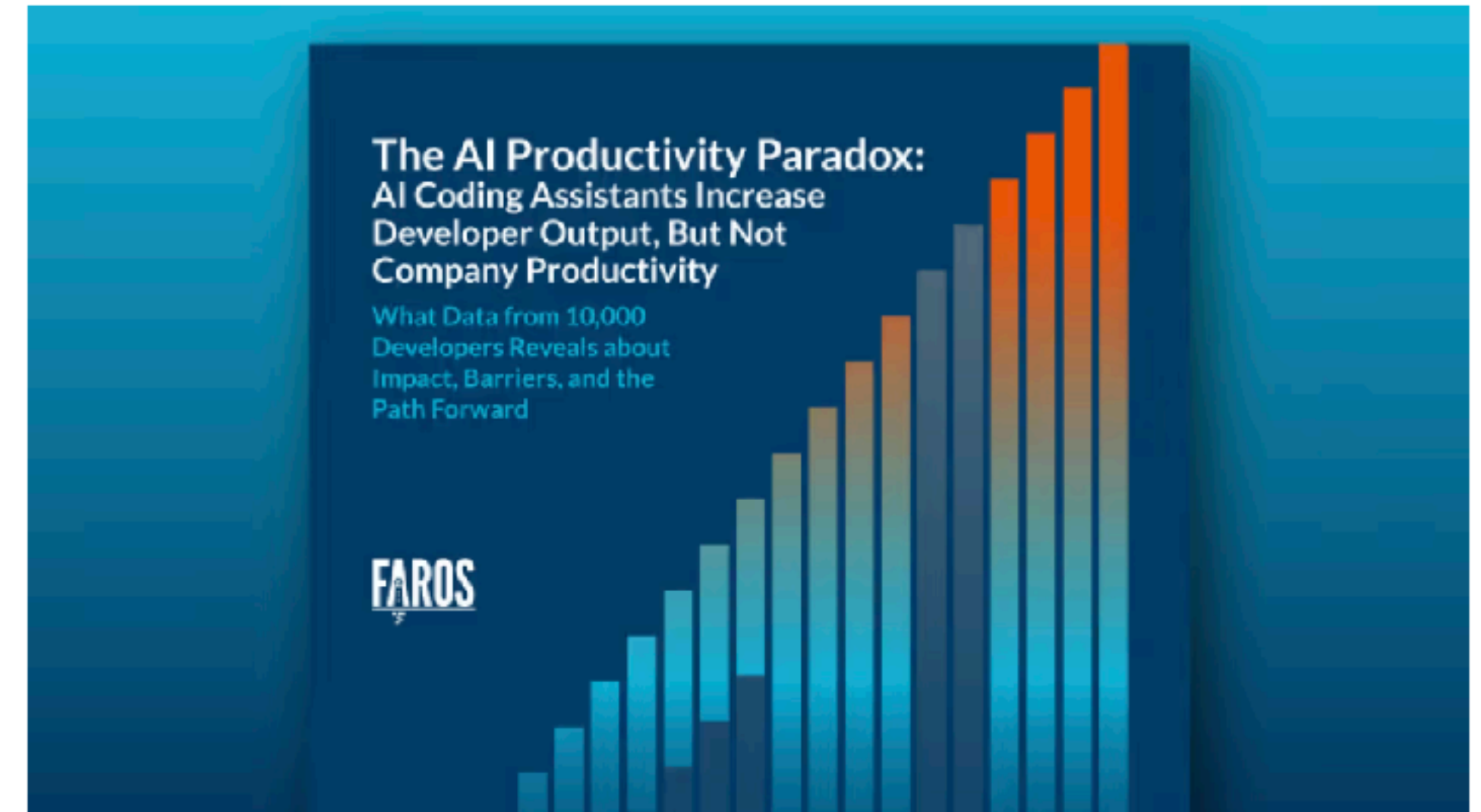
[AI](#)[News](#)[Editor's Pick](#)

The AI Productivity Paradox Report 2025

Key findings from the AI Productivity Paradox Report 2025. Research reveals AI coding assistants increase developer output, but not company productivity. Uncover strategies and enablers for a measurable return on investment.



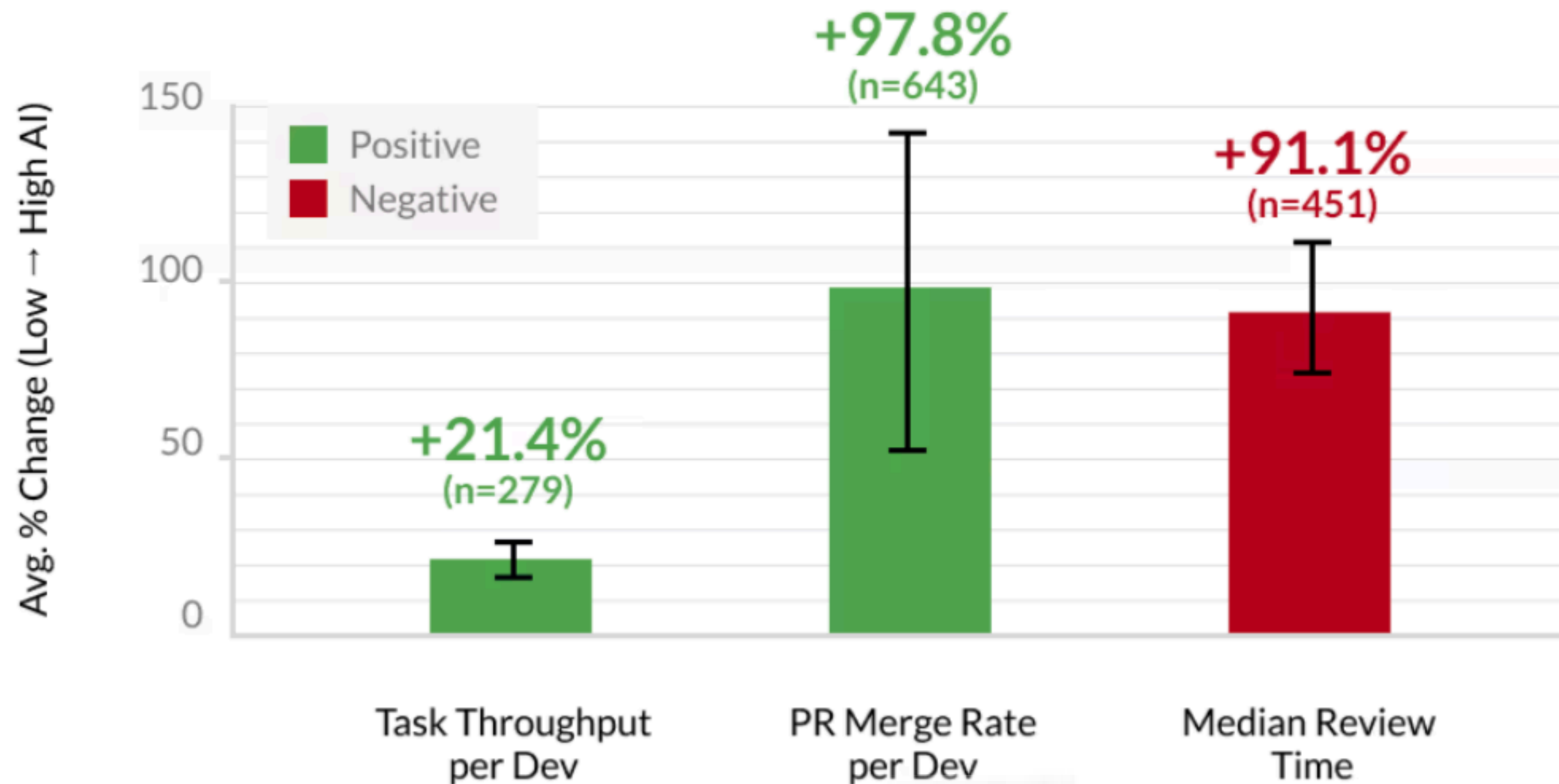
Neely Dunlap



What's holding back AI's productivity boost? | It's not the model—it's your system. GAINS™ reveals why →

AI boosts output, but human review becomes the bottleneck

Velocity metrics: % change from low to high AI adoption



PR review time increases 91%

Artificial Intelligence Isn't a Silver Bullet

New solutions bring both possibilities and problems



Punched cards

- Punch holes in paper cards, with **one card representing one line of code**



Machine code

- The **only language CPUs can directly understand and execute (Fastest execution speed)**
 - Composed entirely of binary digits (0s and 1s)



Assembly language

- Replaces machine code with **human-readable symbols**
 - Converted to machine code by an assembler



High-level programming language

- Compilers translate source code into machine code automatically
 - Lar
 - Imj



Prompts (Chat-based)

Interaction with AI using natural language

- No programming syntax required
- Large execution



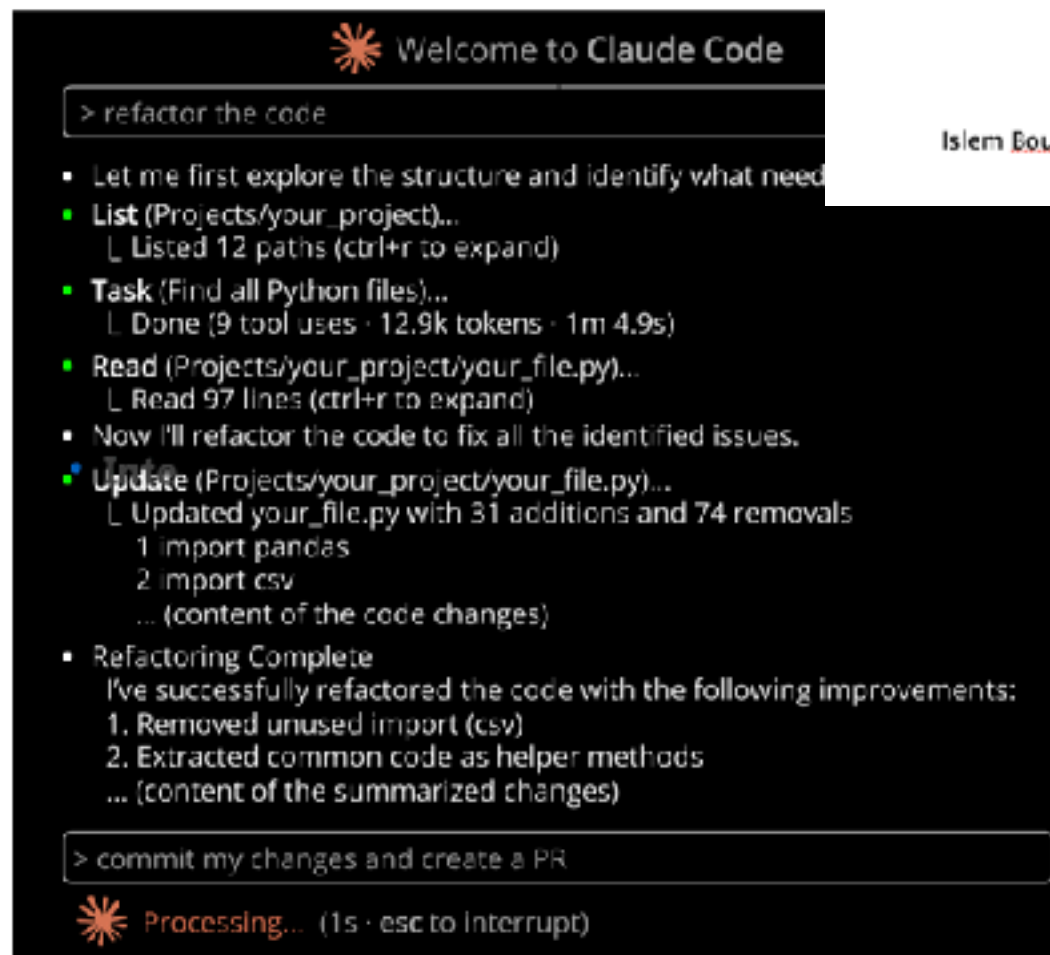
Prompts (Agent-based)

AI systems that autonomously plan and execute multi-step tasks

- Example: "Create a login function"**
With just the instruction, the AI will:

- Analyze the requirements
- Determine the file structure
- Generate the code
- Run tests
- Self-correct any errors

They Use Tools



Developers juggle more workstreams as AI usage grows
Multi-tasking metrics: % change from low to high AI adoption

Avg. % Change (Low → High AI)

Do we need the maintenance for agent-generated code?

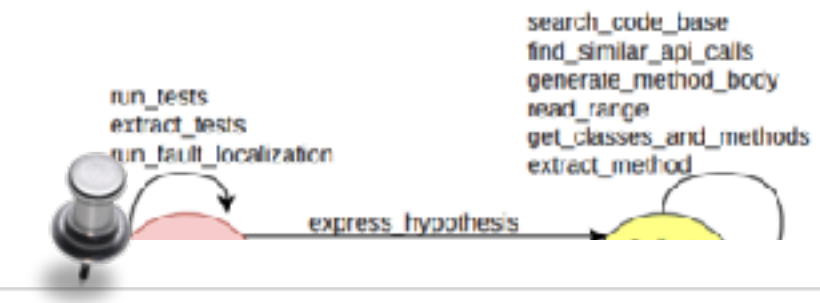


RepairAgent: An Autonomous, LLM-Based Agent for Program Repair

Islem Bouzenia
University of Stuttgart
Gen
li_bouze

Premkumar Devanbu
UC Davis

Michael Fradel
University of Stuttgart



Abstract—Autonomous repair agents on system retain a powerful technique for system repair. RepairAgent introduces RepairAgent, a large language learning-based agent that autonomously identifies bugs, gathering information and formulating hypotheses while deciding which information and tool contributions that are useful for these tools, and agent in invoking the Defects4J dataset to autonomously repair bugs not fixed by prior tools.

The AI Pin

Key findings from 1 assistant increase enablers for a mea



AI boosts output, but human review becomes the bottleneck
Velocity metrics: % change from low to high AI adoption



Artificial Intelligence Isn't a Silver Bullet

New solutions bring both possibilities and problems

