Programming for psychologists

Practical 5.2: AI-assisted coding

Matthias Nau

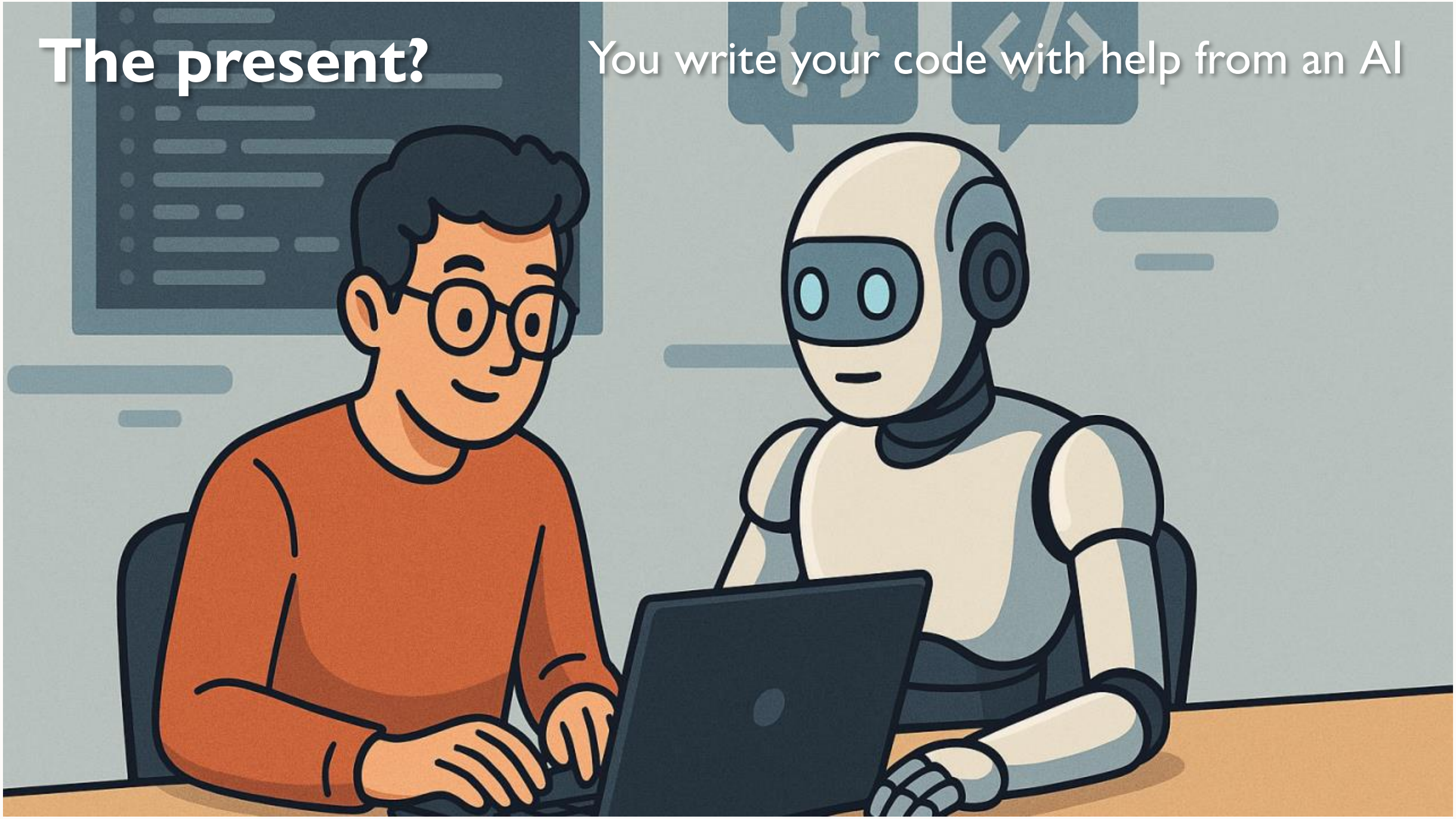
Disclaimer

This lecture is heavily biased by my own experiences and does not reflect generally agreed upon conventions. AI is new to everyone!

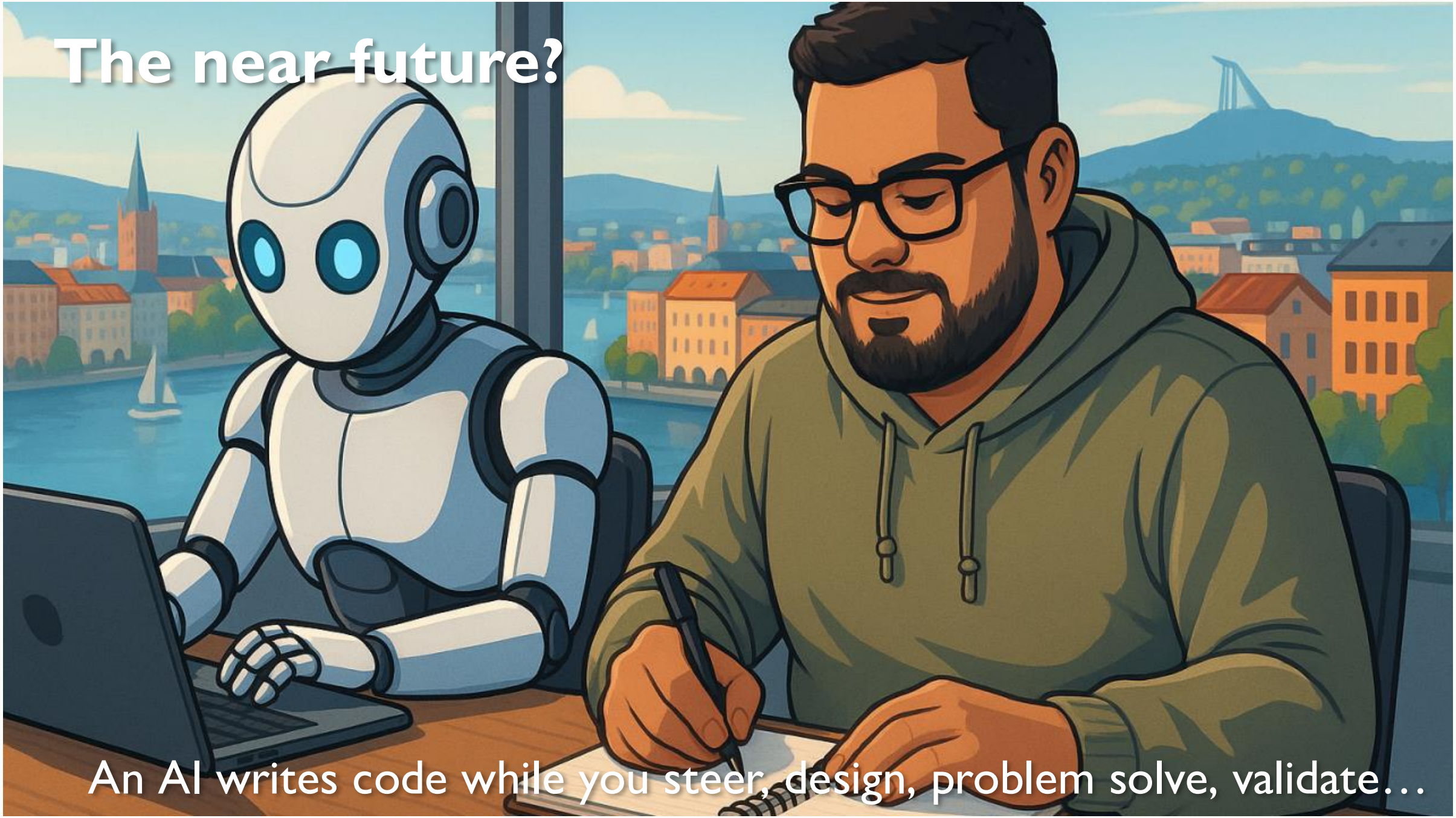
The past?

The present?

You write your code with help from an AI

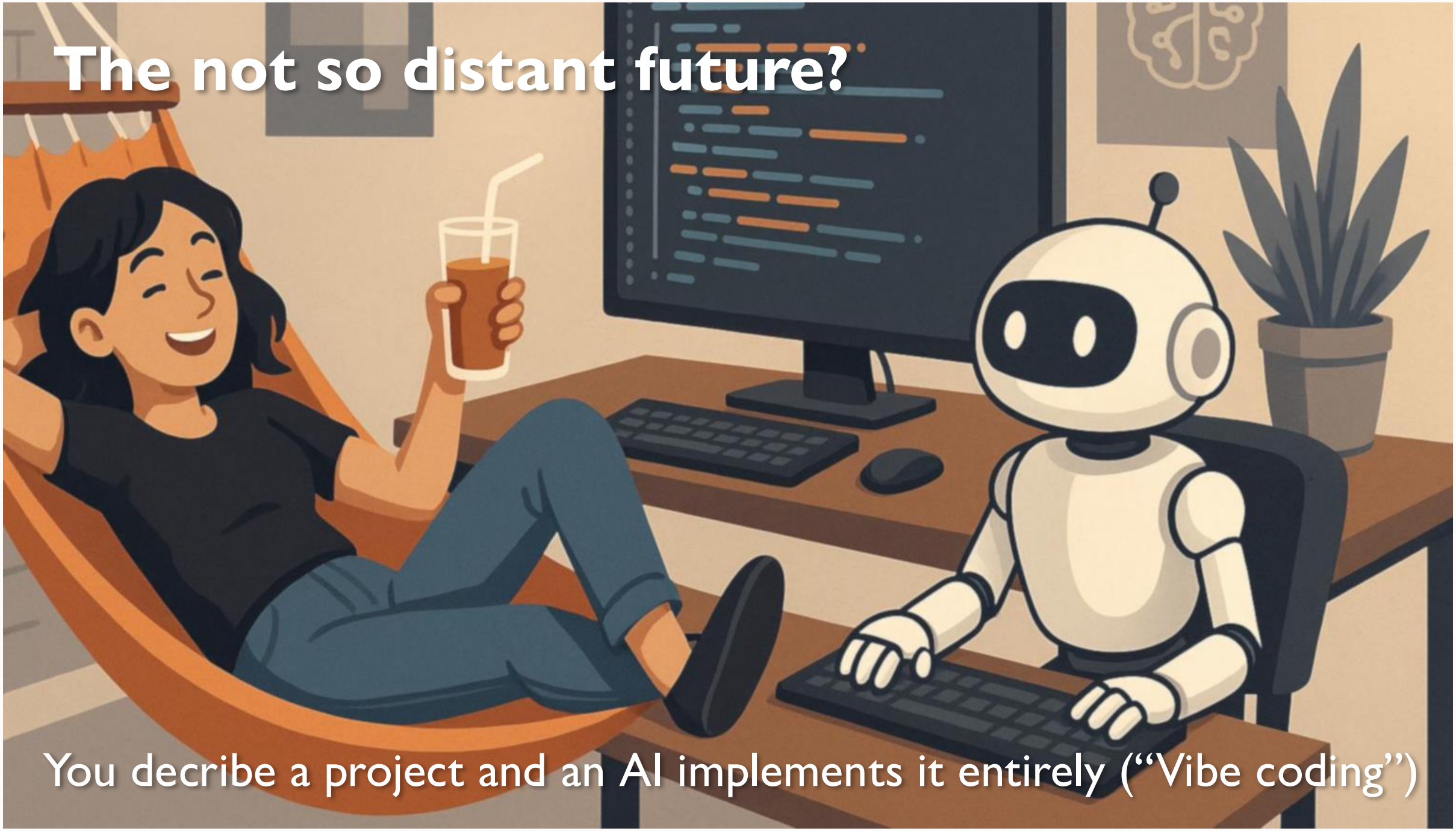


The near future?



An AI writes code while you steer, design, problem solve, validate…

The not so distant future?

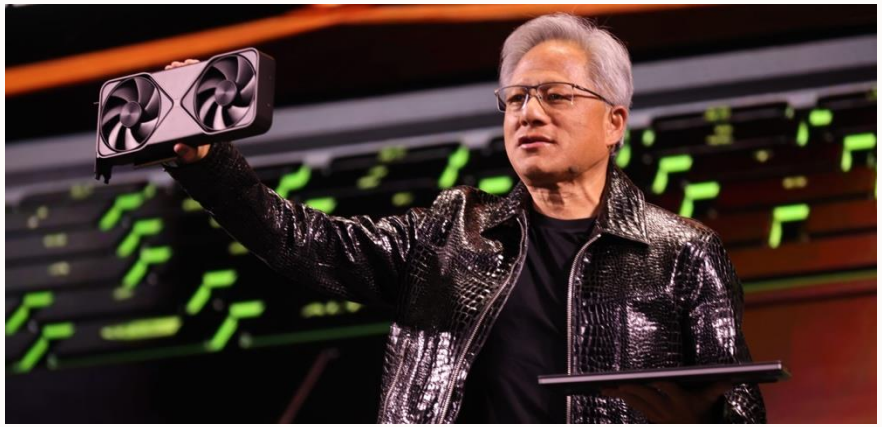


You decribe a project and an AI implements it entirely (“Vibe coding”)

Do you even need programming skills nowadays?

Do you even need **programming skills** nowadays?

Jensen Huang, CEO of NVIDIA
(>4,5 Trillion USD market value)



"Skip coding and learn something else" (Feb 2024)

Matthias Nau, me
(hoping to get research funding)

"As researchers, we have the **responsibility** to conduct rigorous, reproducible, transparent science. While AI can accelerate our progress, it cannot, and should not, take this responsibility from us." (Nov 2025)

Ask yourself: **Can you take responsibility for code that you did not write and do not understand?**

AI-assisted coding

AI-assisted coding

Using **large language models** (LLMs) to help you write, understand, debug, and improve code. It is **collaborative programming** where an AI helps with coding, but **you make the decisions.**

Explosion of tools: ChatGPT, Copilot, Claude, Gemini, Cursor…





AI-assisted coding can greatly **accelerate your progress**, for example by prototyping code, teaching you new libraries, helping to debug, and write documentation. ***BUT!***

AI-assisted coding

Different models excel at **different tasks** and have **different biases**.
If you use them, diversify your repertoire and compare results across models.

Tool Type	Best For	Description
Conversational (ChatGPT, Claude)	Architecture design, complex debugging, learning new concepts	Deep reasoning and flexible problem-solving with extensive context handling, but requires manual code transfer and loses context between sessions
IDE Assistant (Copilot, IntelliSense)	Code completion, refactoring, maintaining flow	Seamless workflow integration with immediate feedback and preserved code context, but limited reasoning for complex architectural decisions
Autonomous Agent (Cursor, Claude Code, Aider)	Rapid prototyping, multi-file changes, large refactoring	High-speed implementation that can work independently across multiple files, but risks code divergence and requires careful monitoring

Bridgeford et al. 2025

Should we **ban AI** entirely from the classroom?

AGAINST THE UNCRITICAL ADOPTION OF 'AI' TECHNOLOGIES IN ACADEMIA

Olivia Guest[1,2], Marcela Suarez[1], Barbara C. N. Müller[3], Edwin van Meerkerk[4], Arnoud Oude Groote Beverborg[5], Ronald de Haan[6], Andrea Reyes Elizondo[7], Mark Blokpoel[1,2], Natalia Scharfenberg[1,2], Annelies Kleinherenbrink[1,8], Ileana Camerino[1], Marieke Woensdregt[1,2], Dagmar Monett[9], Jed Brown[10], Lucy Avraamidou[11], Juliette Alenda-Demoutiez[12], Felienne Hermans[13], and Iris van Rooij[1,2,14]

https://philarchive.org/rec/GUEATU

Group discussion

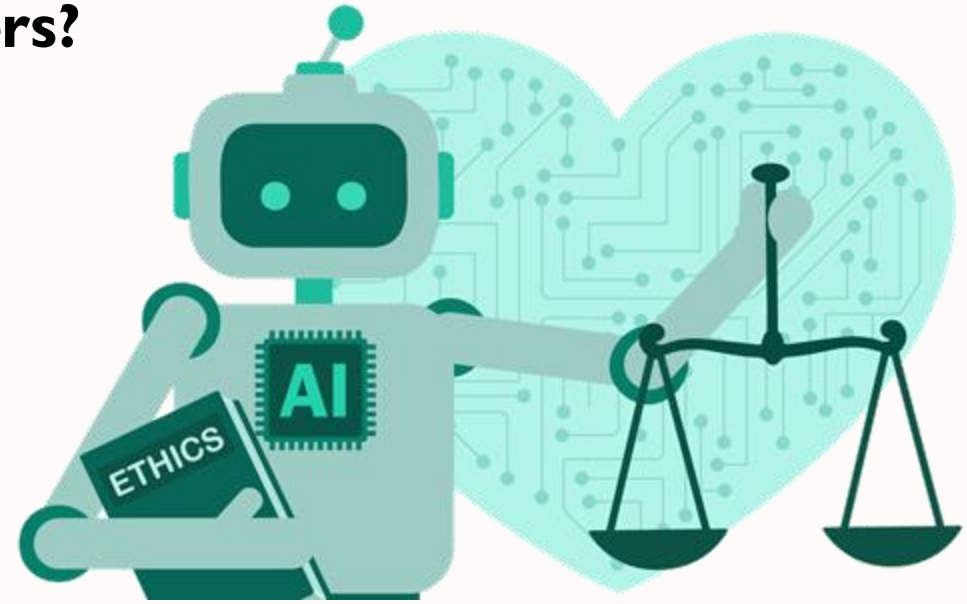
Group discussion: **Ethics & Responsibility**

If a bug leads to a published error, can "AI wrote it" be an excuse?

Does using AI reduce your accountability, or increase it?

Should AI use be disclosed in scientific papers?

Is it ethical to use AI if the source and legal status of the training data is unknown?



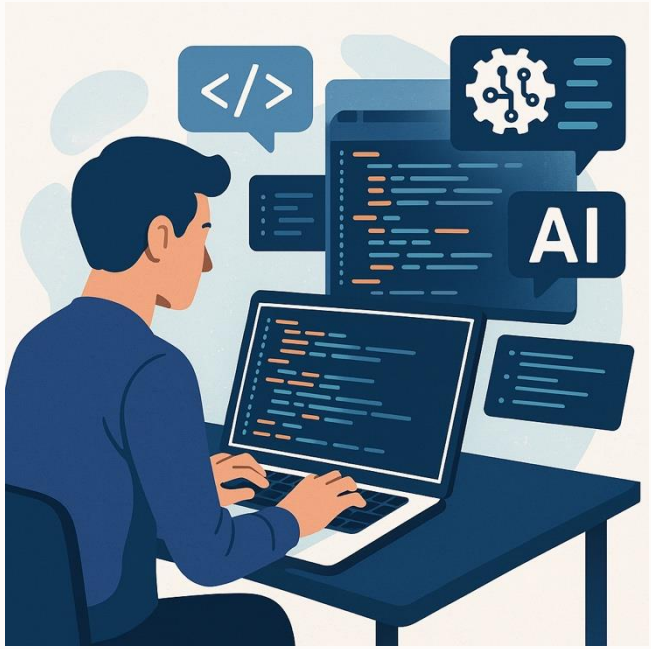
Group discussion: **Skills & Understanding**

How can AI-assistants help you with programming?

Is it ever ok to use code that you don't understand?

What skills do you think you will need to make the most of AI-assistants?

What skills do you see at risk when using AI?

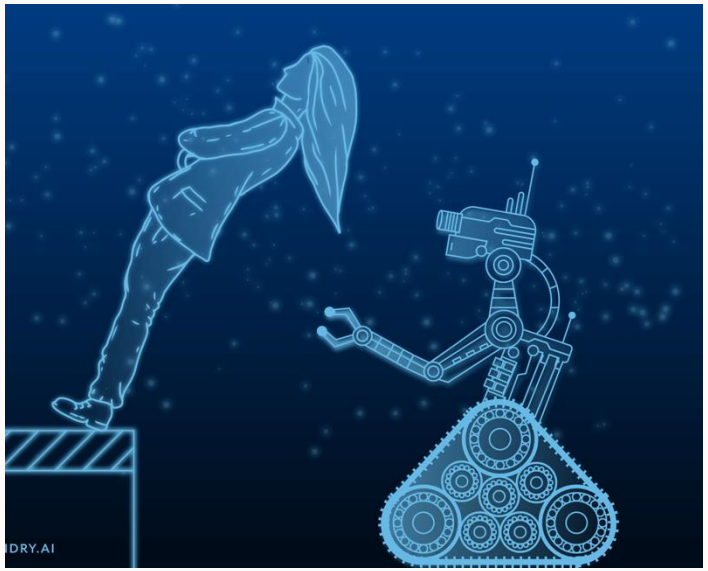


Group discussion: **Trust & Validation**

How can you ensure that AI-generated code does what you think it does?

Can you trust AI to handle edge cases or specific domain conventions?

How do you detect code that looks plausible but is wrong?



Group discussion: **Big picture**

How should we teach programming in a world where AI is abundant?

Are we making ourselves vulnerable by relying on models developed primarily by big tech in non-European countries?

How much environmental cost is acceptable for convenience in coding?

YOU REMAIN FULLY RESPONSIBLE FOR ANY CODE YOU USE!

IF YOU USE AI OR NOT, BE READY TO EXPLAIN ANY LINE OF CODE TO YOUR THESIS SUPERVISOR

Tips and considerations

General tips

- **Explain your code to a peer**. You will find the gaps in your own understanding.
- **Share your code online.** Not only will you help others, but you also hold yourself accountable to use AI-assistants responsibly.
- **Test your code with simple, known cases**. Use real or simulated data with a known pattern to validate your code.
- **Test each function**. It is not enough to test the full pipeline. Test each step!
- Run your code **line by line** and **print or plot** each sub-outcome.
- **Use version control** (e.g., Git & GitHub, see Lecture 5). It enables safe experimentation and testing of your code.

How to **prompt effectively** for coding

- **Ask about specific problems** and **avoid sweeping requests**

(e.g., ”Write code that reads a .csv file with XYZ format.”,
instead of “Write code that detects saccades.”)

- **Give context**. The more concretely you explain the goal of your code, the better.

- **Ask for stepwise reasoning**. Let the AI explain what the code does, but don’t trust the code or the explanation blindly (test, test, test!).

- **Specify** desired input/output formats, constraints and domain assumptions.

- **State the libraries** you want and inform yourself first about those you don’t know.

- **Ask for validation tests** before the actual code of interest.

Ten simple rules for AI-assisted coding

1: Gather domain knowledge first

2: Distinguish problem framing from coding

3: Choose appropriate AI model

4: Start by thinking through a potential solution

5: Manage context strategically

6: Implement test-driven development with AI

7: Leverage AI for test planning and refinement

8: Monitor progress and know when to restart

9: Critically review generated code

10: Refine code incrementally with focused objectives

Ten Simple Rules for AI-Assisted Coding in Science

Eric W. Bridgeford[1,†], Iain Campbell[2], Zijiao Chen[1], Zhicheng Lin[3,4], Harrison Ritz[2], Joachim Vandekerckhove[5], Russell A. Poldrack[1].

Abstract. While AI coding tools have demonstrated potential to accelerate software development, their use in scientific computing raises critical questions about code quality and scientific validity. In this paper, we provide ten practical rules for AI-assisted coding that balance leveraging capabilities of AI with maintaining scientific and methodological rigor. We address how AI can be leveraged strategically throughout the development cycle with four key themes: problem preparation and understanding, managing context and interaction, testing and validation, and code quality assurance and iterative improvement. These principles serve to emphasize maintaining human agency in coding decisions, preserving the domain expertise essential for methodologically sound research. These rules are intended to help researchers harness AI's transformative potential for faster software development while ensuring that their code meets the standards of reliability, reproducibility, and scientific validity that research integrity demands.

1 Introduction

The integration of artificial intelligence into scientific computing represents one of the most significant shifts in research methodology since the advent of personal computers. Large language models (LLMs) trained on vast corpora of code can now generate syntactically correct, functionally appropriate programs from natural language descriptions, a capability that was inconceivable just a few years ago [1]. Tools like GitHub Copilot, ChatGPT, and Claude have democratized access to sophisticated programming assistance, enabling researchers with limited coding experience to implement complex analyses and build robust scientific software [2]. Agentic coding tools like Claude Code and Cursor have further enabled entire coding workflows by invoking tools outside the language model.

AI-assisted coding tools have demonstrated measurable productivity gains in some controlled studies, with benefits spanning development speed, code quality, and maintainability [2]. However, the evidence for these benefits remains contested and situation-dependent. While some enterprise studies and developer surveys report significant productivity increases and improved code quality [3], recent randomized controlled trials with experienced developers found that AI tools actually slowed completion times, despite developers believing they were working faster [4]. Additional concerns about code quality have emerged, with research analyzing almost 200 million lines of code showing substantial increases in copy-pasted code and decreases in refactoring as AI use has become more prominent when using AI assistants [5]. These contradictory findings suggest that productivity effects are far from well-understood, and may vary based on developer experience, task complexity, and codebase characteristics. These questions become particularly acute in scientific computing, where code is not merely a means to an end but often embodies scientific reasoning, methodological decisions, and domain expertise. The validity, reproducibility, and interpretability of scientific software directly impact research integrity and the reliability of scientific findings [6].

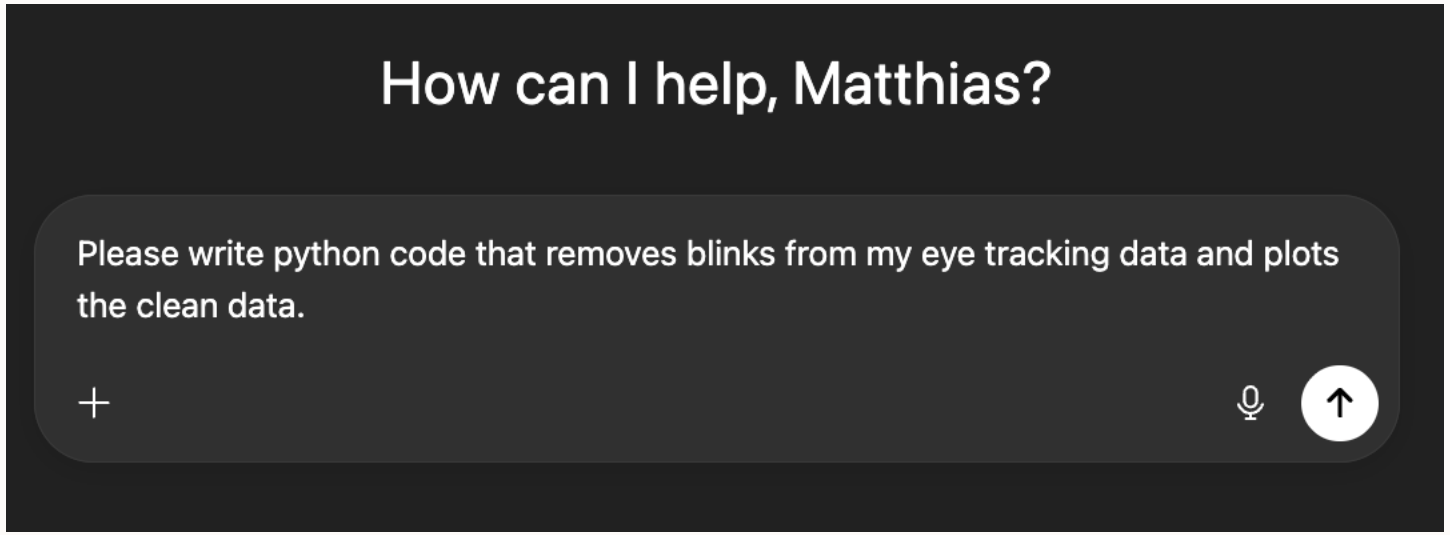
The implications for scientific computing are profound. Programming involves complex problem decomposition, algorithmic thinking, and domain-specific reasoning. These cognitive skills that atrophy with excessive AI dependence. Furthermore, scientific code often requires deep understanding of mathematical models, statistical methods, and domain-specific conventions that cannot be adequately captured by AI tools trained on general programming corpora. These challenges are compounded by the technical limitations inherent to current AI systems: their context windows constrain how much

[1] Stanford University, Stanford, CA, USA, [2] Princeton University, Princeton, NJ, USA, [3] University of Science and Technology of China, Hefei, China, [4] Yonsei University, Seoul, Republic of Korea, [5] University of California, Irvine, CA, USA
[†] Corresponding author: Eric W. Bridgeford (ericwb@stanford.edu).

Bridgeford et al. 2025

https://arxiv.org/pdf/2510.22254

Group practice: **Improve the following prompt**



Student answers: Will be added here.

Live demo: Coding with Claude vs. ChatGPT

Before the next practical, go through these slides again!

Do you know what the following terms mean?

- AI-assisted coding
- Vibe Coding
- Co-pilot
- Test-driven development
- Prompting

AI reduces typing, not thinking.
Your judgement remains irreplaceable.

Towards a better P(s)ychology!