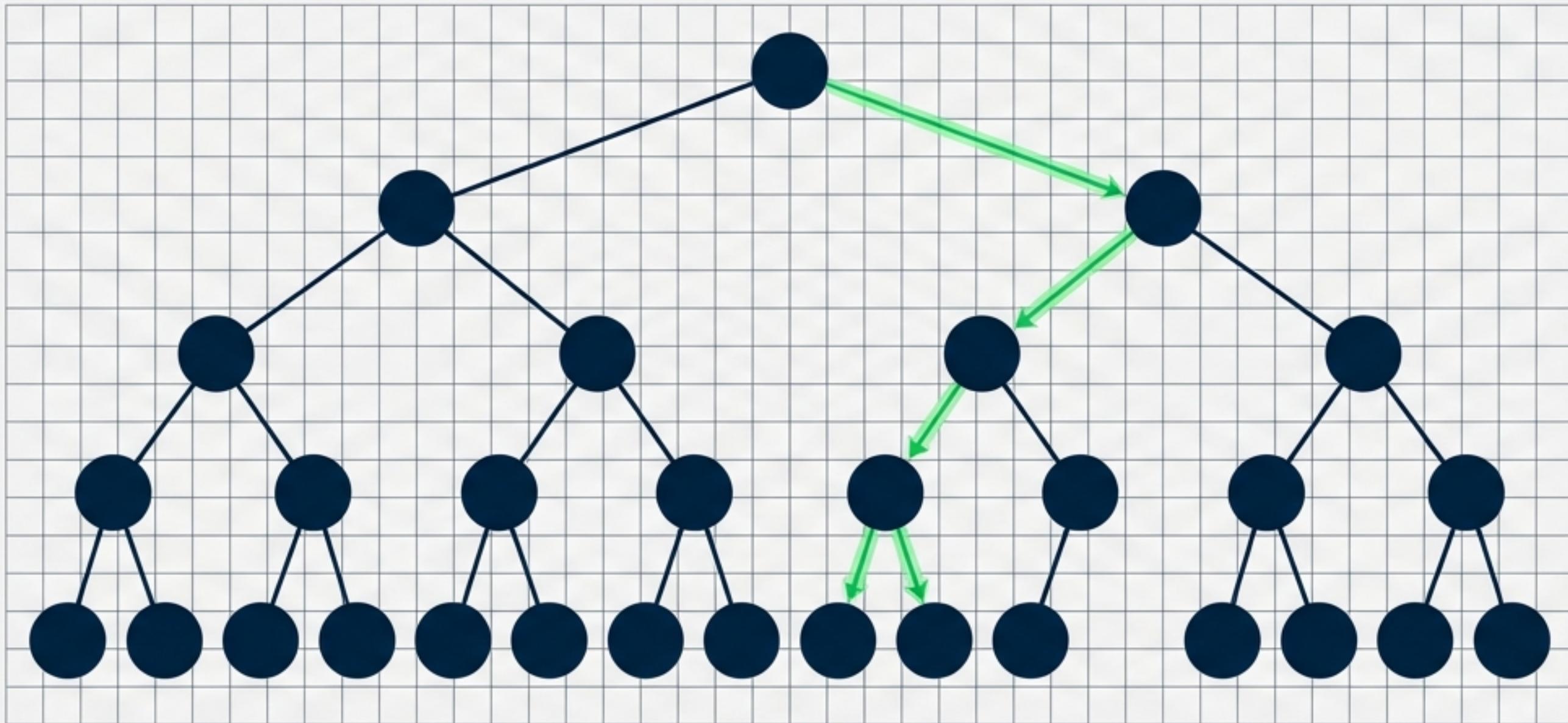


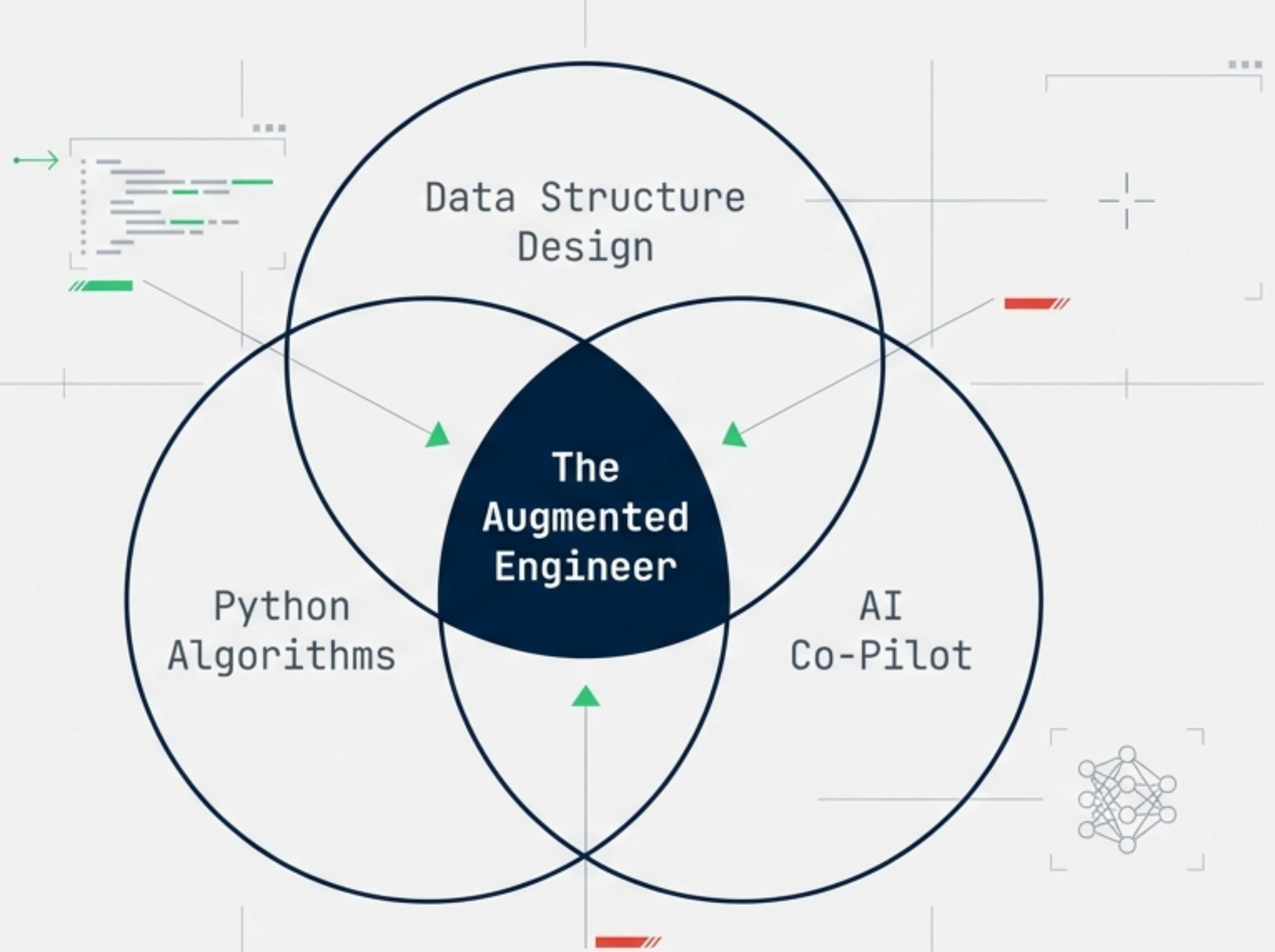
Data Structures with AI Integration

Mastering Algorithms in the Age of Artificial Intelligence



Week 1: Introduction &
Complexity Analysis

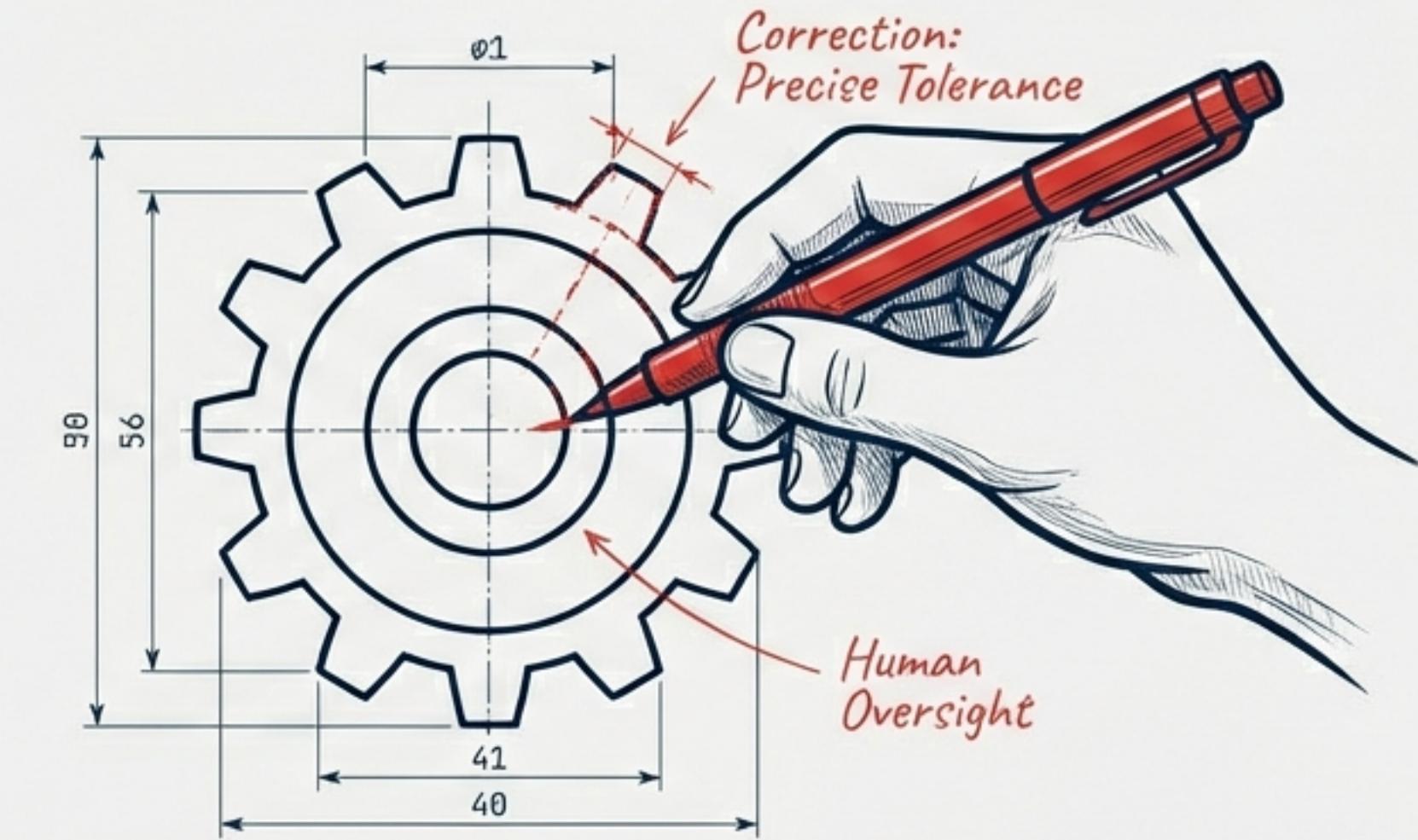
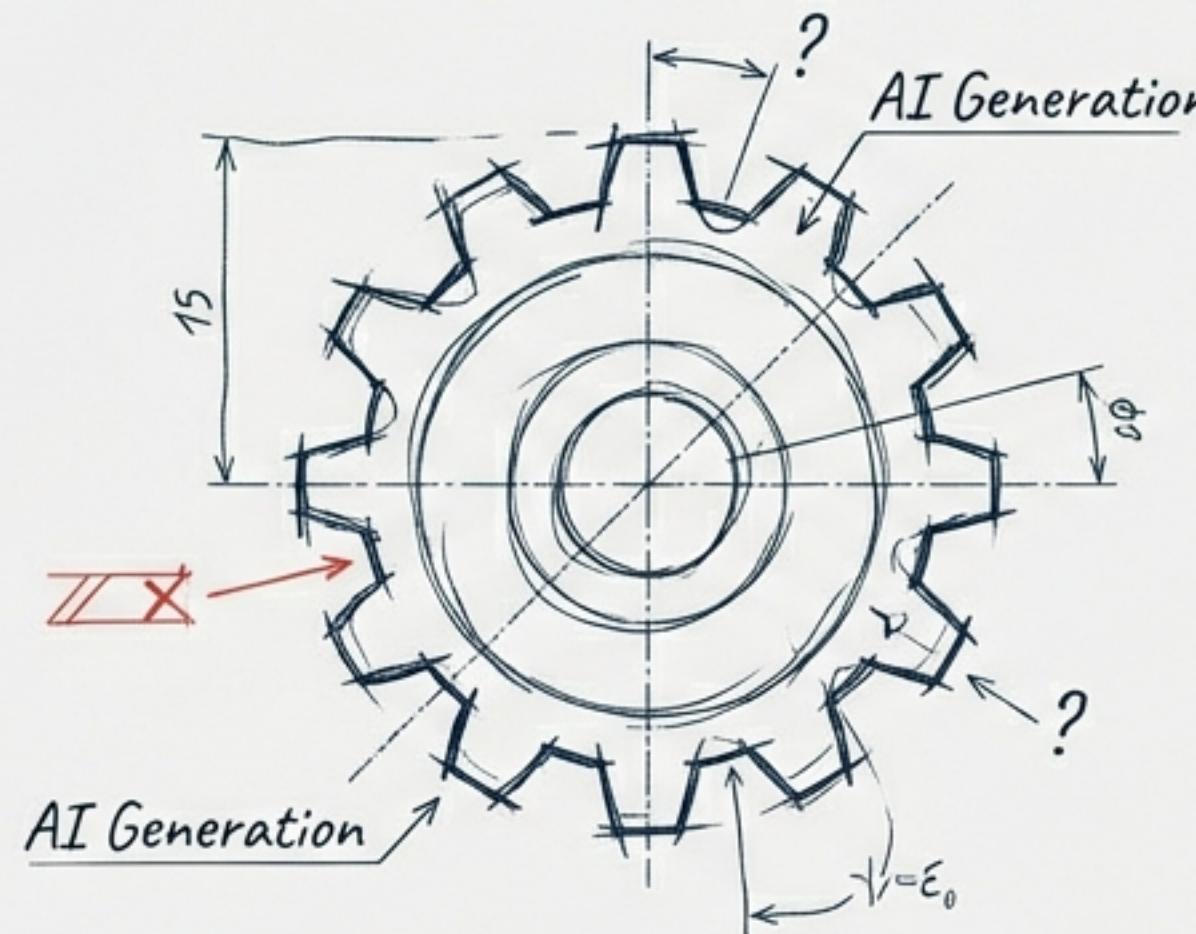
Developing the Augmented Engineer



The Mission

- Goal: Design, analyze, and implement efficient software.
- The Paradigm Shift: AI is a tool, not a crutch.
- Outcome: Mastery of hybrid engineering.

The Philosophy: Managing Your Fallible Intern



The Mindset

Treat AI as a fast intern.
It lacks context and
accountability.

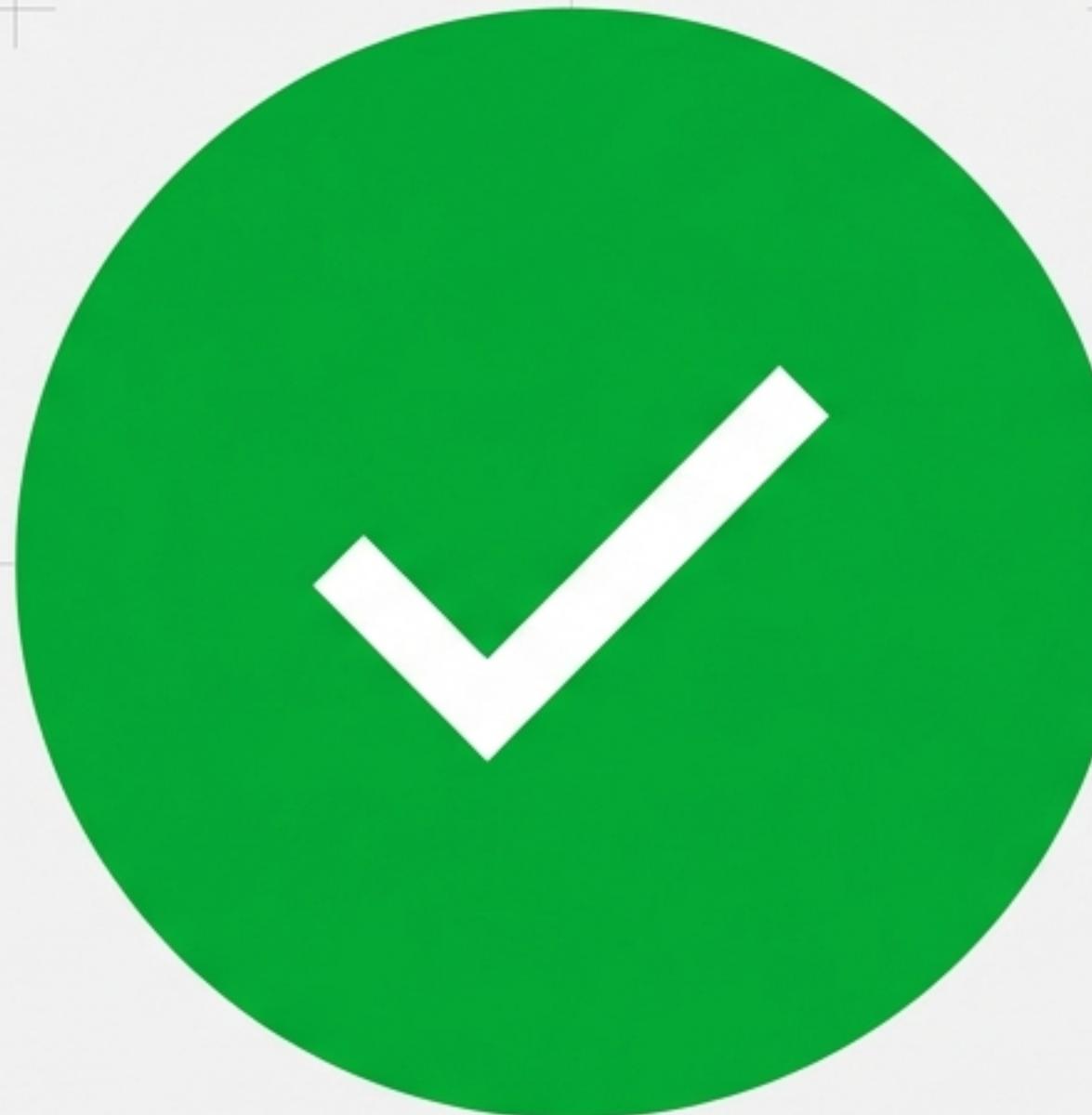
The Reality

AI hallucinations are
frequent. It will confidently
fail edge cases.

The Rule

You are responsible for every
semicolon. Manage the AI; do
not be replaced by it.

Rules of Engagement: Permitted Use



Green Light Activities

- **Scaffolding:** Generating boilerplate code (e.g., Class Node).
- **Exploration:** Concept analogies ('Explain Heapsort like I'm 5').
- **Debugging:** Locating syntax errors or logical bottlenecks.
- **Ideation:** Brainstorming industrial applications.

The Red Lines: Prohibited Use



Red Light Activities

- **Logic Outsourcing:** Asking AI to solve the problem before drafting pseudocode.
- **Blind Copy-Pasting:** Submitting code you have not read or understood.
- **The Hallucination Defense:** ‘The AI told me it was right’ = Grade 0.

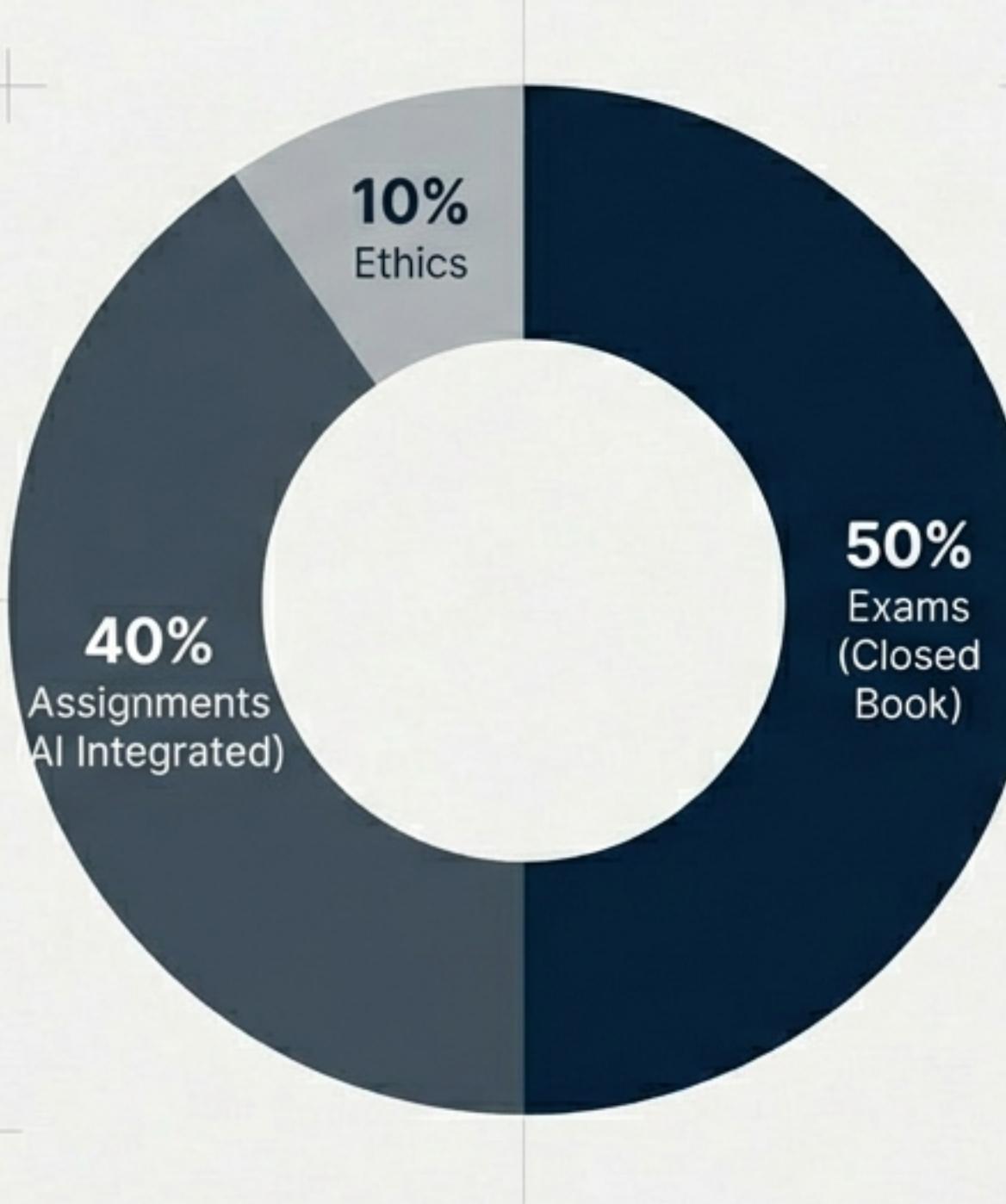
Documentation & Transparency



Proof of Ownership

- **Disclosure Statement** required for every assignment (Direct, Assisted, or Collaborative).
- **The Prompt Log:** A mandatory PDF export of your conversation history.
- **Transparency Standard:** If you can't document it, you didn't do it.

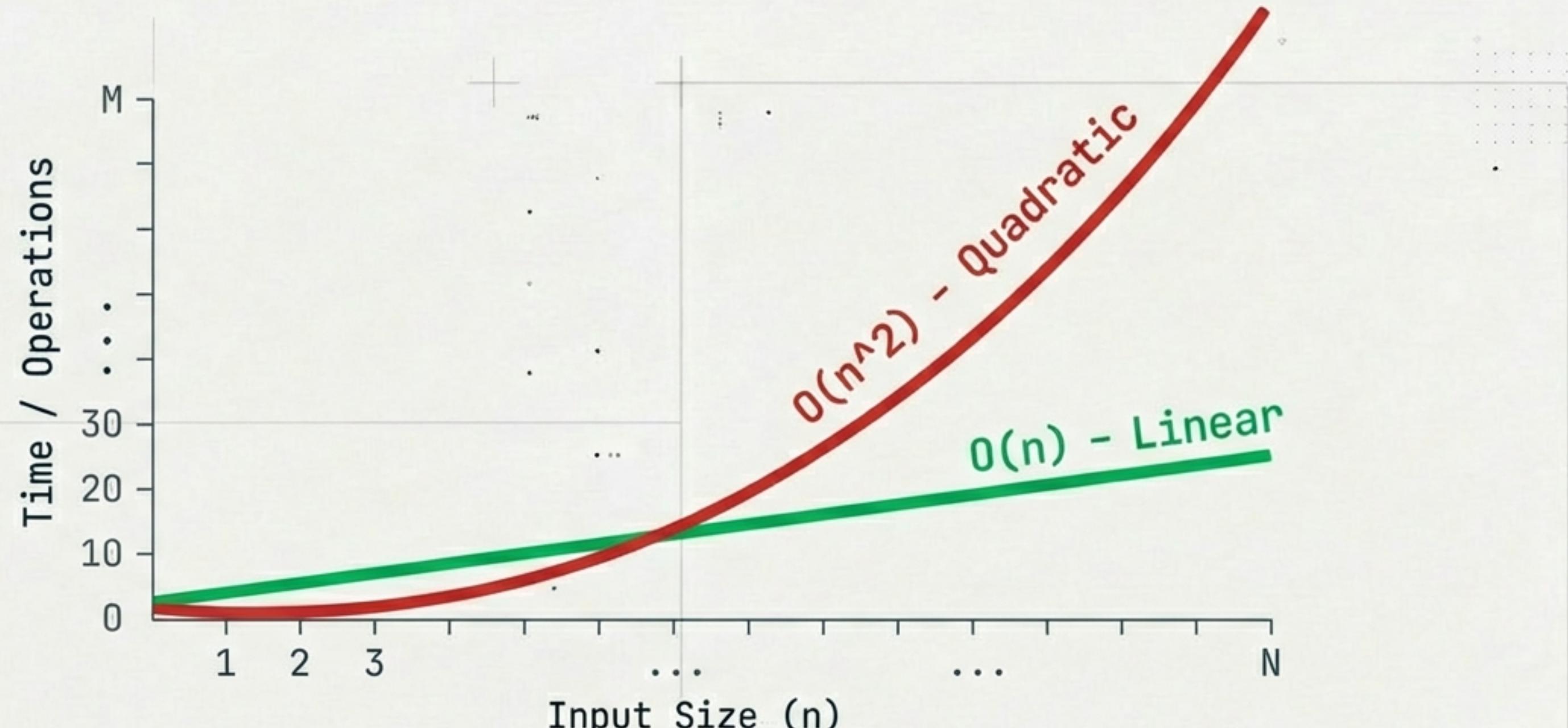
Assessment & The 'EYC' Rule



The 'Explain Your Code' Rule

Random 5-minute viva sessions.
If you cannot explain the logic of
your submission line-by-line, the
grade is reset to 0.

Week 1 Topic: Complexity Analysis



Why It Matters: Bad data structures explode under load.
We must identify efficiency before implementation.

Week 1 Focus: AI Integration

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which contains a folder named 'COMPLEXITY_ANALYSIS' with files 'main.py', 'utils.py', and 'data.json'. The main editor area displays two Python functions:

```
1 def linear_search(arr, target):
2     for i in range(len(arr)):
3         if arr[i] == target:
4             return i
5     return -1 # O(n)
6
7 def bubble_sort(arr):
8     n = len(arr)
9     for i in range(n):
10        for j in range(0, n-i-1):
11            if arr[j] > arr[j+1]:
12                arr[j], arr[j+1] = arr[j+1], arr[j]
13    return arr # O(n^2)
```

To the right of the code, there is a 'Chat' sidebar. It contains a message from an AI asking to explain the difference between $O(n)$ and $O(n^2)$ with code examples. Below it, another message from the AI provides Python examples and an explanation of the time complexities.

Task: Ask AI to explain the difference
• between $O(n)$ and $O(n^2)$ with code
examples.

Verification:
✓ Check the math manually. Does the
✗ generated code actually match the
complexity claimed?

Summary & Next Steps

- Setup IDE Extensions
- Review Syllabus
- Start Complexity Module

**Remember: You are the Pilot;
AI is the Co-Pilot.**

JetBrains Mono
Next Week: Functions, Arrays, and Pointers