A Five Step Framework for Effective AI-Assisted Coding

Context, Research, Problem Framing, Critical Thinking, and Refining

Andrew Stellman

©2026 Andrew Stellman, all rights reserved

Andrew Stellman

- Over 20 years training people on software development
- Author of 6 O'Reilly books including *Head First C#*
- Full-time software developer and team lead
- Passionate about all things code.

What's in this course

A really quick overview of what we're going to learn…
and then we'll jump right into it.

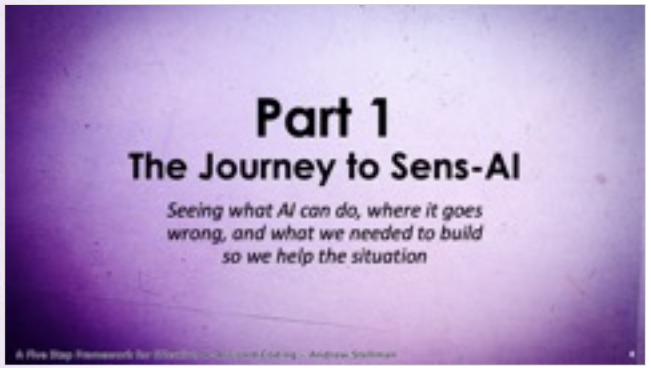
What this course covers

- **The Sens-AI Framework: a practical 5-Step framework for understanding and using AI tools**
  You'll learn about *research*, *context*, *problem framing*, *refining*, and *critical thinking*—the core habits of the framework that help you use AI tools effectively for real coding work.
- **Understanding how AI fits into software development**
  We'll explore how AI tools like ChatGPT and Copilot can help you design, write, and improve code—without replacing your own thinking.
- **Real-World Exercises and Examples**
  You'll get practice applying these habits to development tasks using Java, C#, or Python, using AI to solve problems while building the critical thinking skills every developer needs.

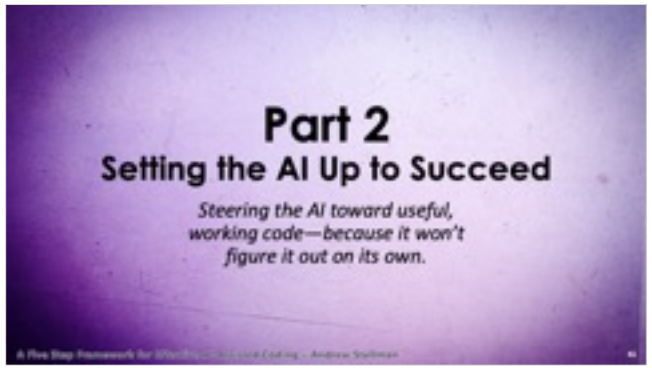
How this course is structured

- In the main **Presentation** sections, I'll do teaching and give demos using AI tools and using C# or Java as languages. You're encouraged to follow along on your own.
- In the **Exercise** sections, I'll give you time to keep working on your own. You can download the code and slides from the GitHub page for this course: https://bit.ly/coding-ai-course
- This course is divided into three parts with a short break between them. Each part ends with a **Q&A** section where I answer your questions.

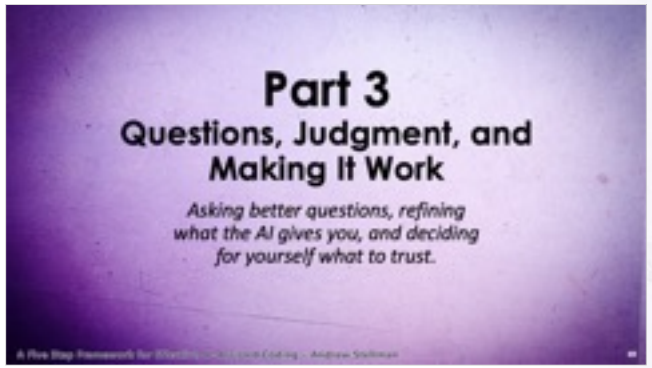
How this course is structured



Understand why using AI for coding is different. See what AI can do, where it goes wrong, and why we needed a framework that teaches developers better habits.



Learn how to give the AI what it needs to be useful, with clear goals, the right context, and your own research. Use these habits to steer the AI toward the code you want.



Practice problem framing, refining, and critical thinking so you can balance fast AI-assisted coding with the careful judgment that good development demands.

Before we begin, a quick poll...

How often do you currently use AI coding tools (like GitHub Copilot, ChatGPT, Claude, or Cursor) in your development work?

- Daily or almost daily
- A few times a week
- Occasionally (once a week or less)
- Rarely or never
- I've tried them but stopped using them

Your answers will help me calibrate this course to today's audience!

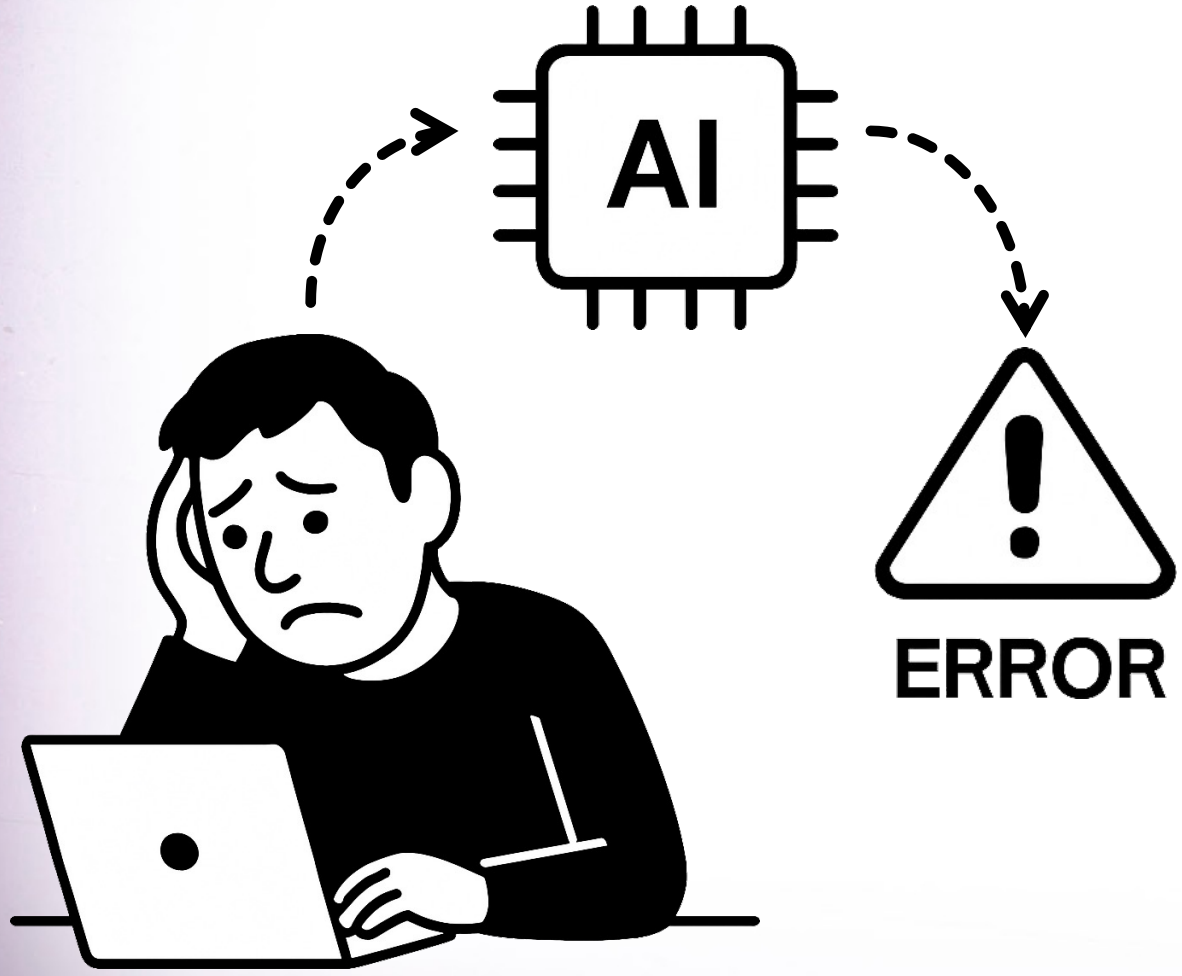
Part 1
The Journey to Sens-AI

Seeing what AI can do, where it goes wrong, and what we needed to build so we help the situation

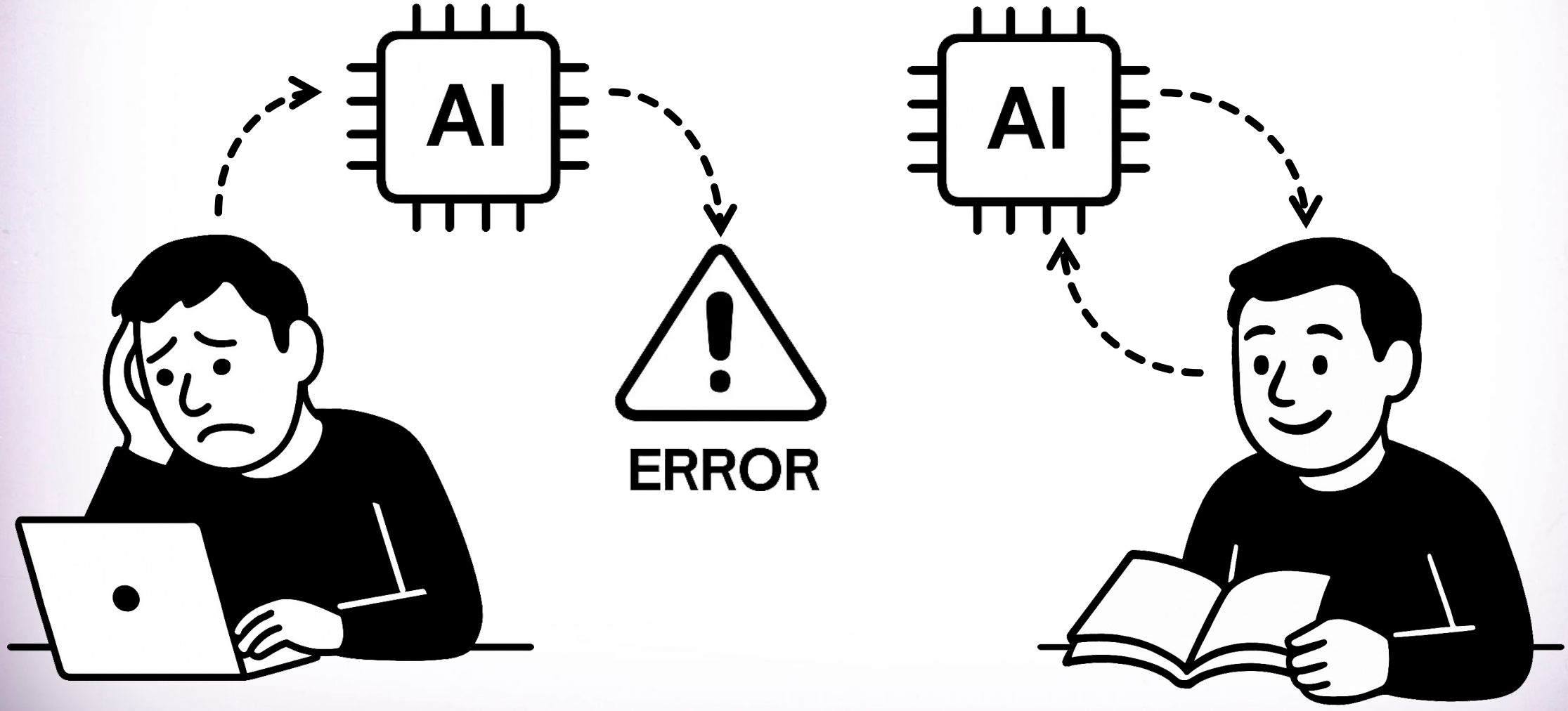
Bridging the AI Learning Gap

The Sens-AI Framework: what it does, why we need it, and a little bit about how it was developed.

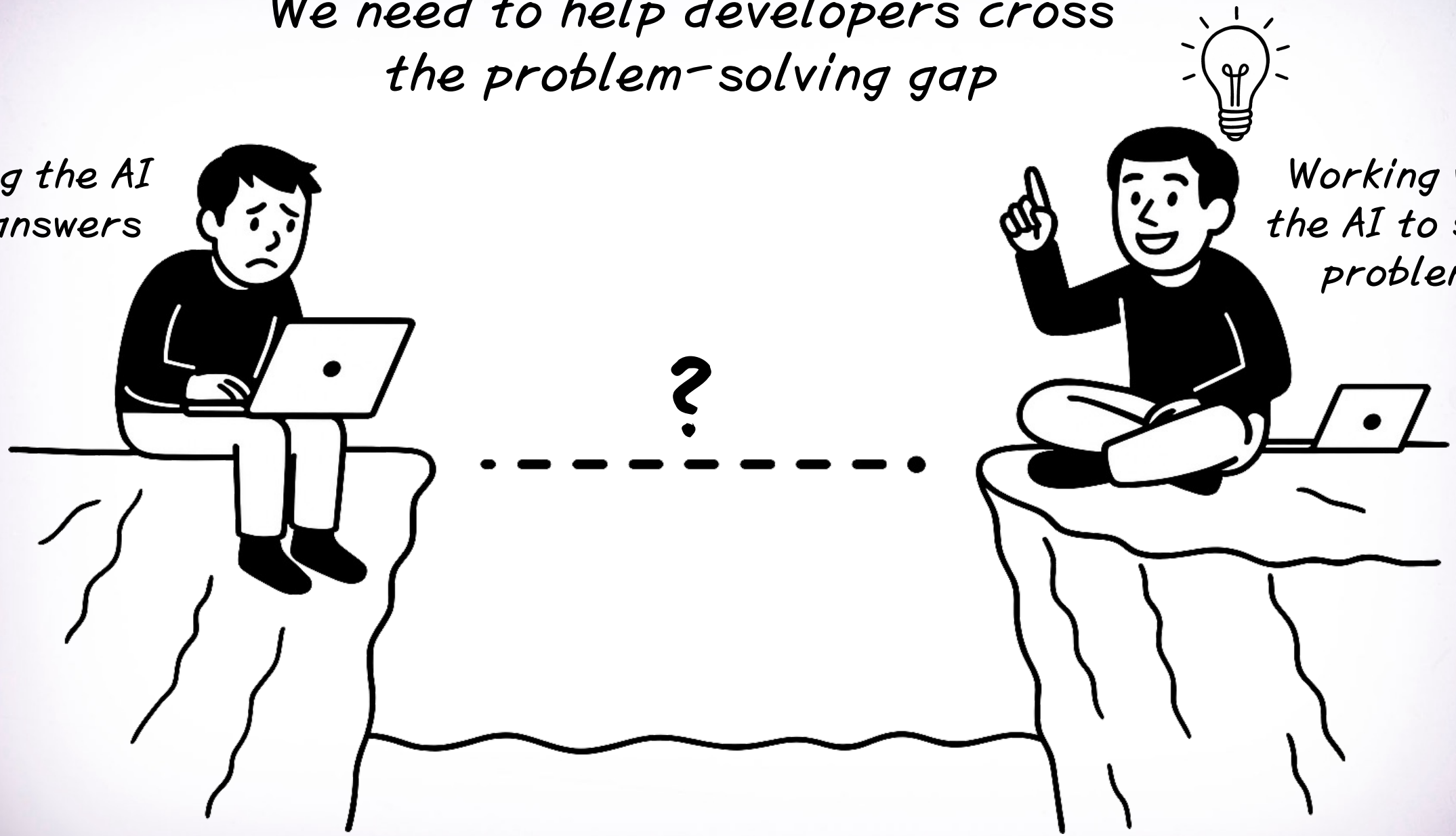
When developers only ask AI for answers, they often get stuck



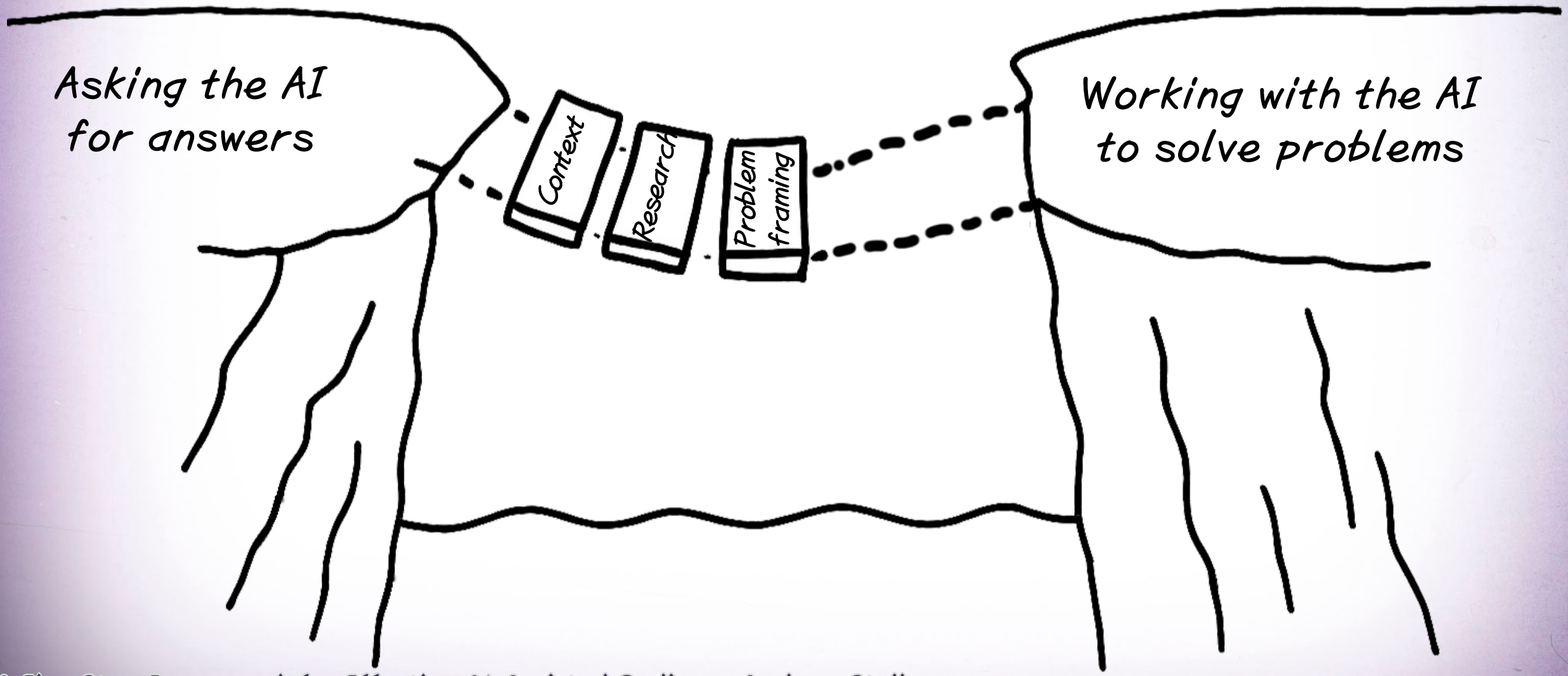
AI works best when developers think critically and guide it



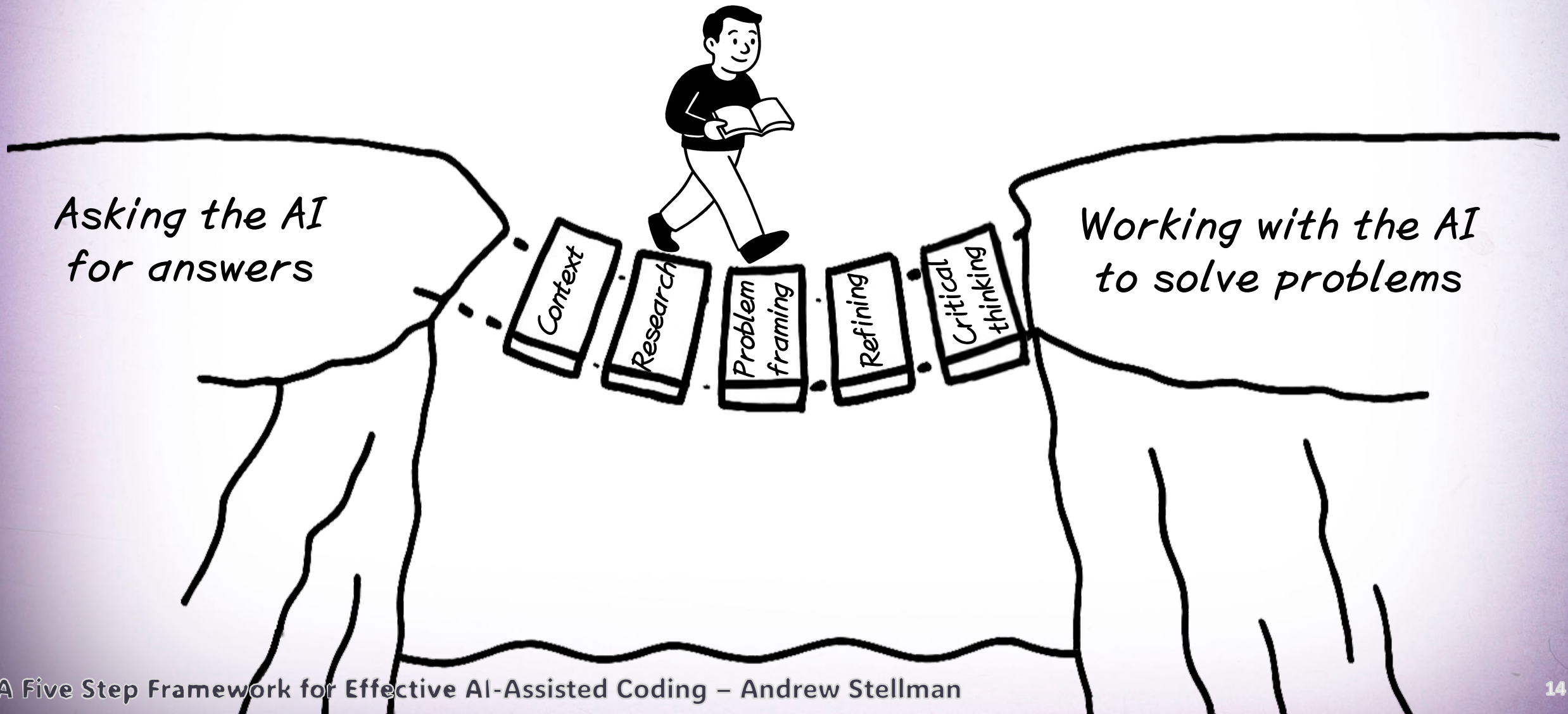
We need to help developers cross the problem-solving gap



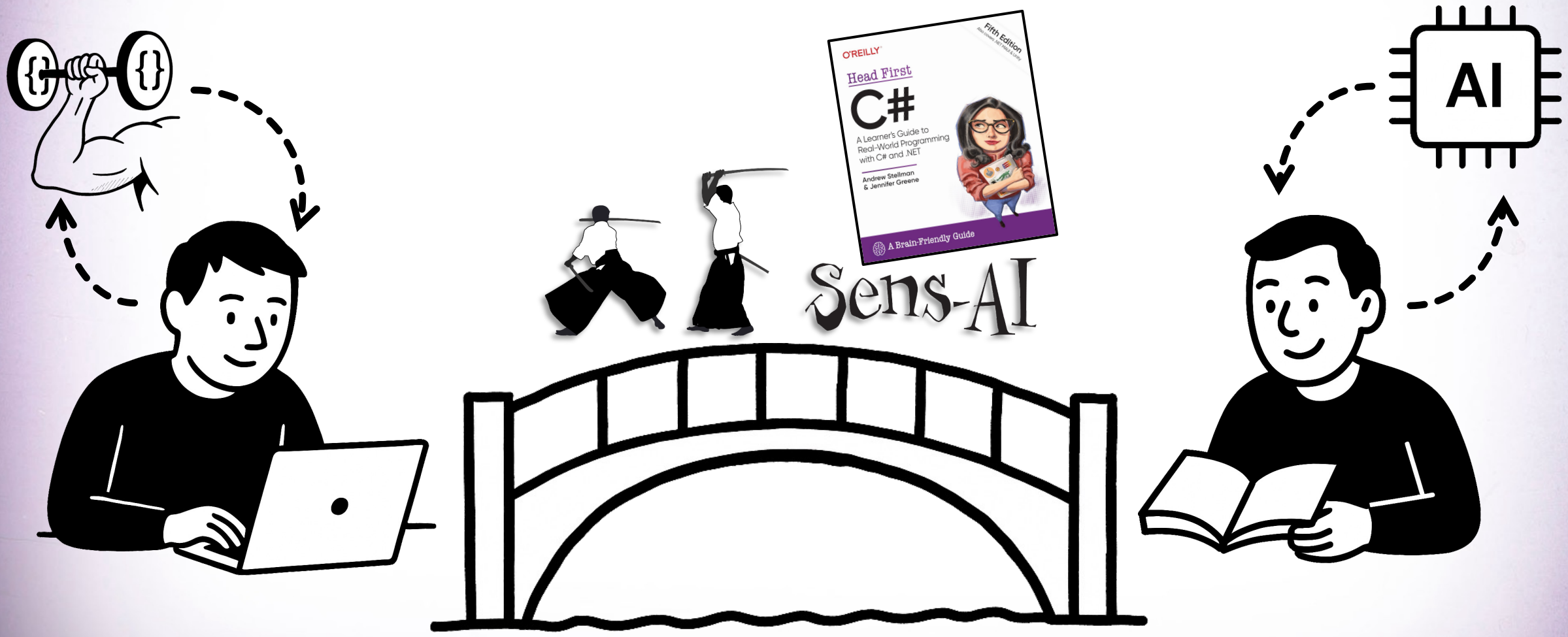
Bridging the gap starts with how developers work with AI



When developers learn to guide AI, it becomes a partner



Sens-AI bridges the gap by teaching essential AI habits and strengthening coding skills together



Exercise

You'll give the AI a starting point—and have it generate an entire app.

Use AI to write code (Part 1)

- Open an AI tool like ChatGPT, Copilot, Claude, or Gemini (I'll demo this in GitHub Copilot)
- Give the first prompt, choosing either C#, Java, or Python:

```
Here's a class. Use it for my next prompt, which will
describe a new project to build.
```

- Press shift-enter twice to add two lines to the end of the chat
- Go to https://bit.ly/sens-ai-course and copy the C#, Java, or Python Guy class (adapted from page 179 of *Head First C#* 5th edition).
- Paste the class in and submit your prompt.
- The AI should give you a response anticipating your next prompt.

Use AI to write code (Part 2)

- Start your next prompt by entering this text:

  ```
  Here's the starting point for a project. Replace
  all of the comments with working code that does
  what each comment describes. The code should add
  an if statement after an else to check for more
  than one condition. It first checks if whichGuy is
  Joe, then it checks if whichGuy is Bob, and if
  neither are true, it writes a line to the console.
  ```
  Press shift-enter twice to add two lines.
- Go to https://bit.ly/sens-ai-course and copy the instructions, which are copied from page 181 of *Head First C#* (5th edition).
- Submit the prompt. The AI should generate code for a complete app.
- Copy the code into your favorite IDE and run it.

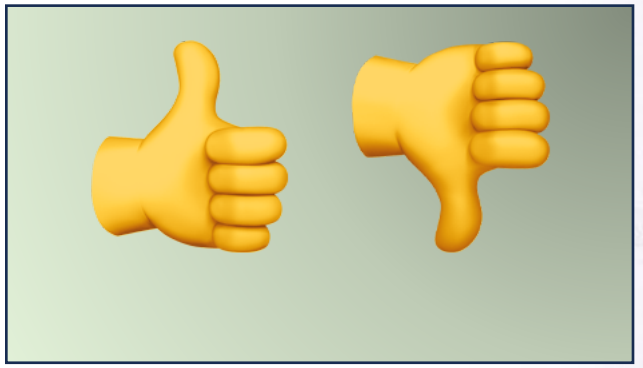
You just asked the AI to generate a complete project. How did it do?

Let's do a pulse check to see how your AIs did.

- Did it run the first time?
- Did it do everything you expected?

Answer 👍 if the answer to questions is **yes**

Answer 👎 if the answer to either question is **no**



Did anything about the result surprise you?

Let everyone know if something was weird! Put it in the chat.

Borrowed power and the illusion of understanding

How building an AI-assisted learning path in *Head First C#* became the foundation for the Sens-AI Framework

The Sens-AI Framework Helps You *Use* AI

This framework is about how you think about AI in your day-to-day work. It gives you practical habits for:

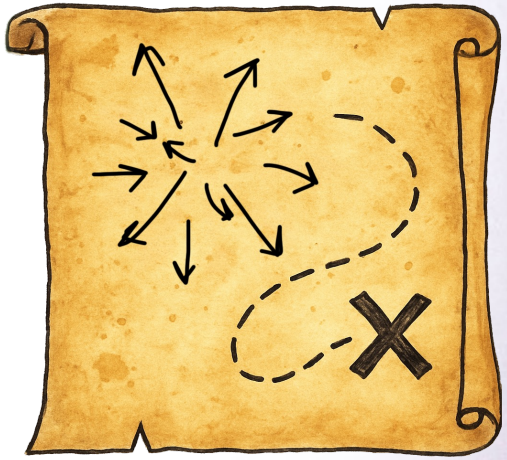
- Figuring out what you actually want the AI to do
- Giving better prompts with the right context
- Reading and understanding the output instead of just accepting it
- Iterating until it matches what you need

The framework is designed to help you navigate using AI effectively… ***like any other development tool****.*

The Sens-AI Framework Is Also About *Learning* AI

This framework is about using AI effectively. It's <u>also</u> about learning effectively. It's structured to help you:

- Ramp up on a complex set of skills in an orderly way
- Break down AI use into clear, learnable habits
- Build confidence applying those habits on real projects
- Make it easier to get the material into your brain and actually *retain* it



We developed it this way so you can learn to navigate complex ideas more easily, and so you can see <u>how</u> the learning path itself was designed.

Borrowed power and fall from grace

We used video game design techniques in *Head First C#* to set the stage for learners.

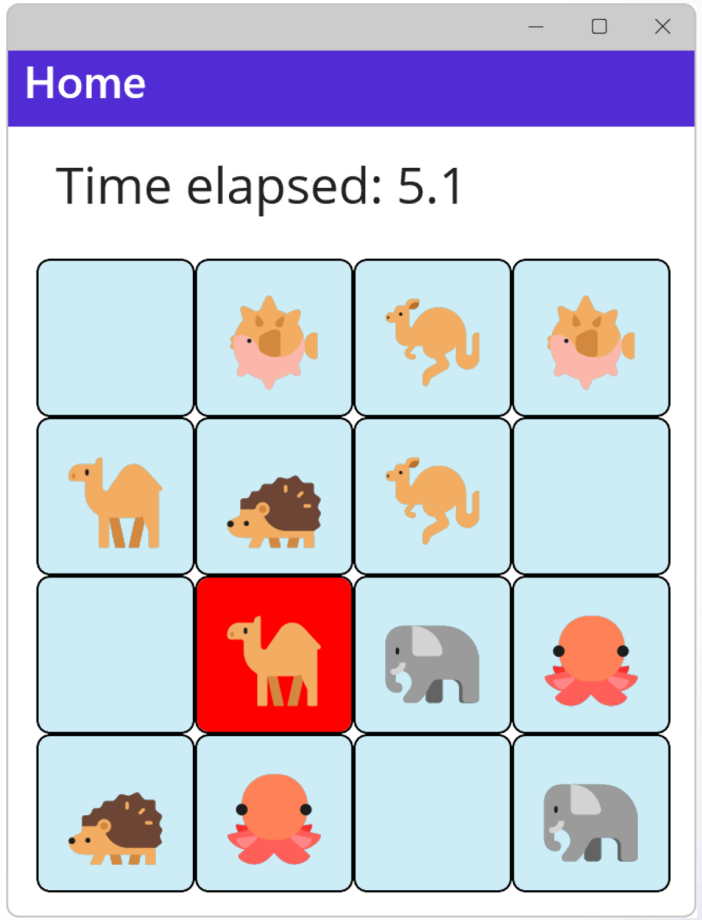
- **Borrowed power**: Players start with powerful abilities they haven't earned yet—just to show what's possible.
- **Fall from grace**: A specific version of Borrowed Power where the power is stripped away after the intro, giving players a clear goal to work toward.

God of War III (2010)

I wondered if we could use the same technique for teaching about AI.

Why borrowed power works for learning

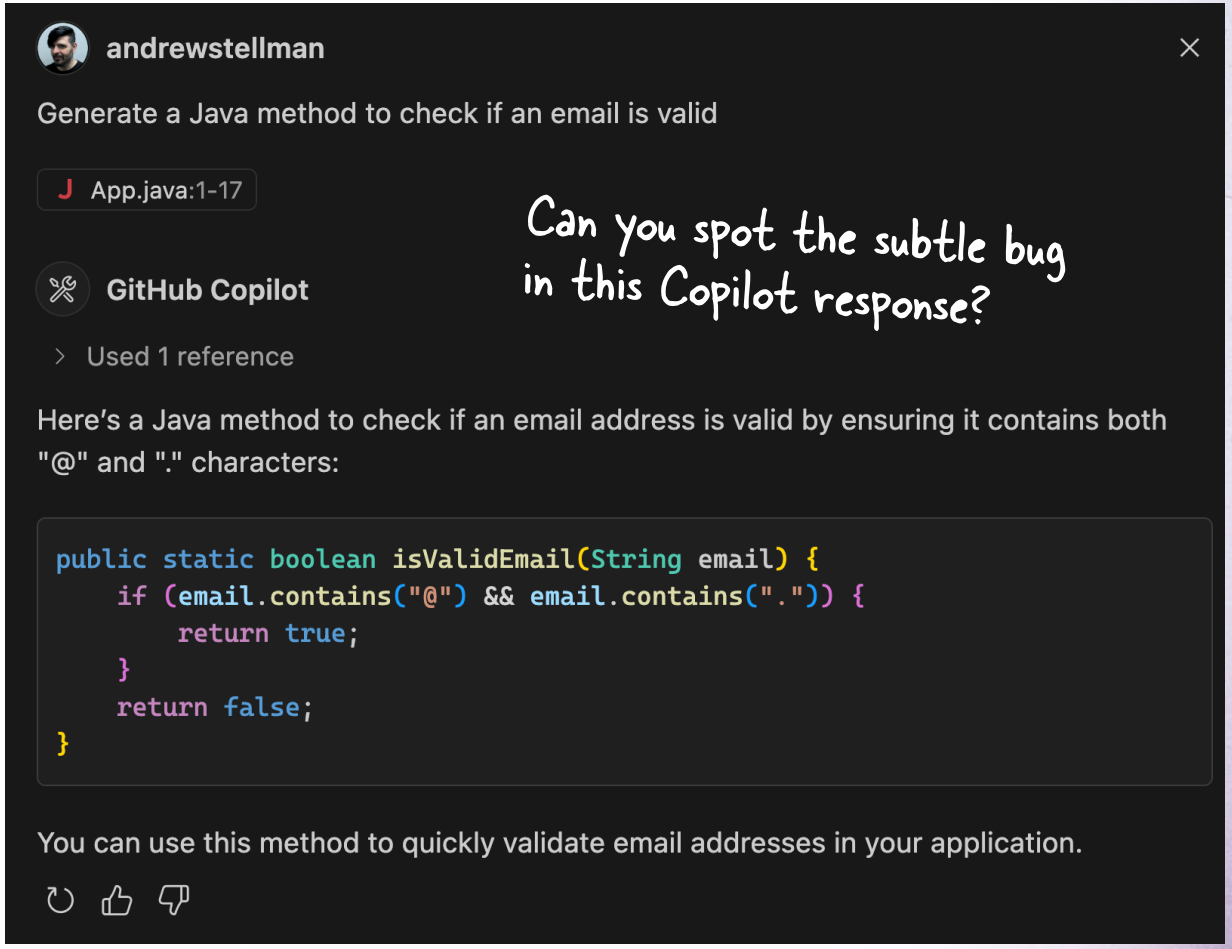
- With AI, you can generate working code right away, even full apps.
- In *Head First C#*, readers start by building a complete game *even before they fully understand how it works*.
- That early success feels real and **motivates** people to keep going.
- But that feeling of progress can be misleading if it's not backed by understanding.



The first chapter of Head First C# is all about building this game. The rest of the book is about learning what we just did.

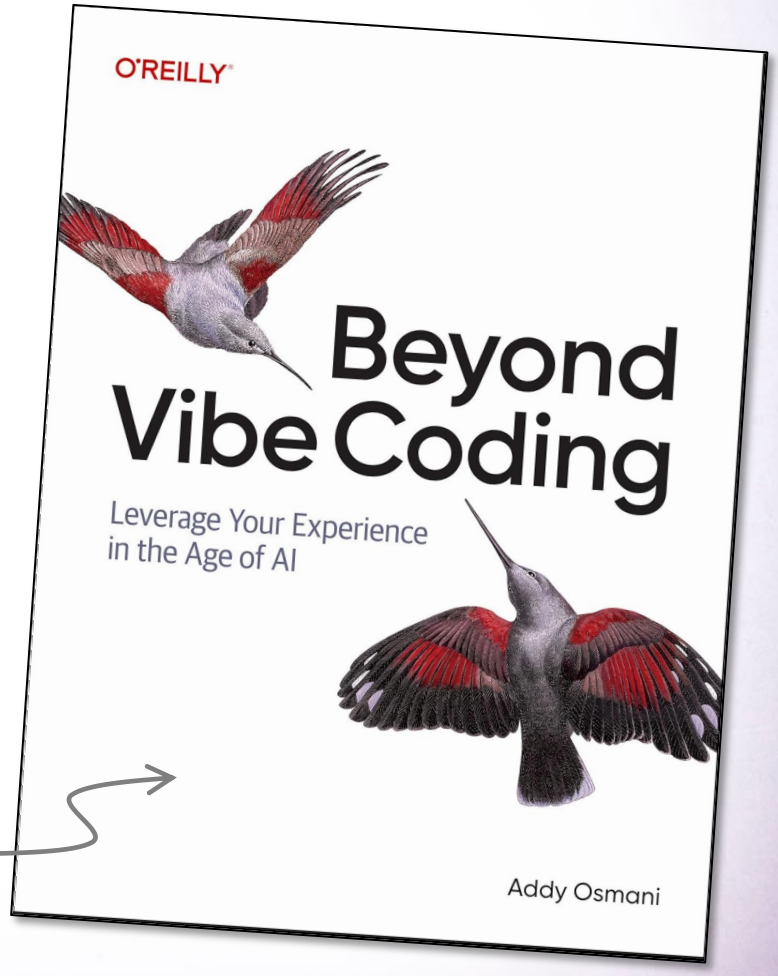
The illusion of understanding

- AI-generated code looks clean, complete, and confident. That makes it easy to assume it's correct… even when it's not.
- Obvious bugs are easy to catch, but we don't always know how to check for subtle mistakes .
- That **false sense of clarity** is what can make the power we borrow from an AI dangerous.



Vibe coding is an effective example of borrowed power

- *Vibe coding* is a widely recognized—and often effective—pattern where developers rapidly prompt, get code, and iterate.
- It treats the AI like an "answer machine" to quickly explore, refine, and shape ideas.
- It's popular because it helps offload details to the AI, making exploration faster and easier.
- But it's **borrowed power**: you're leaning on the AI's "understanding" instead of your own.
- The rewards are speed and creativity.
- But it's easy to "check out" and let the AI do all your coding for you. ***Where might that become risky?***



This is a great resource to help you understand the benefits and challenges of vibe coding.

From borrowed power to the Sens-AI Framework

- At the start of *Head First C#*, the newest learners don't yet have the knowledge to build full apps with AI.
- So we begin with a different kind of lesson: have the AI give a wrong—but confident!—answer.
- That moment shows learners how easily they can be misled.
- It helps them understand the need to slow down, question the output, and build real habits.

That was a key insight helping lead to the Sens-AI Framework.

And it's why we'll start with the habit that matters most: ***trust but verify****.*

Trust but Verify

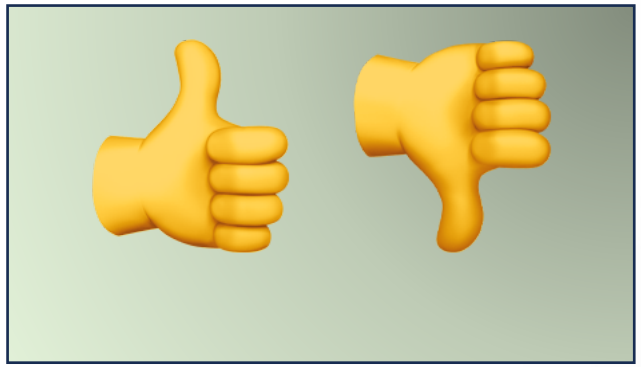
Ask an AI anything and it'll always give an answer…
with confidence. That doesn't mean it's right.

What happens if we ask a simple question about code?

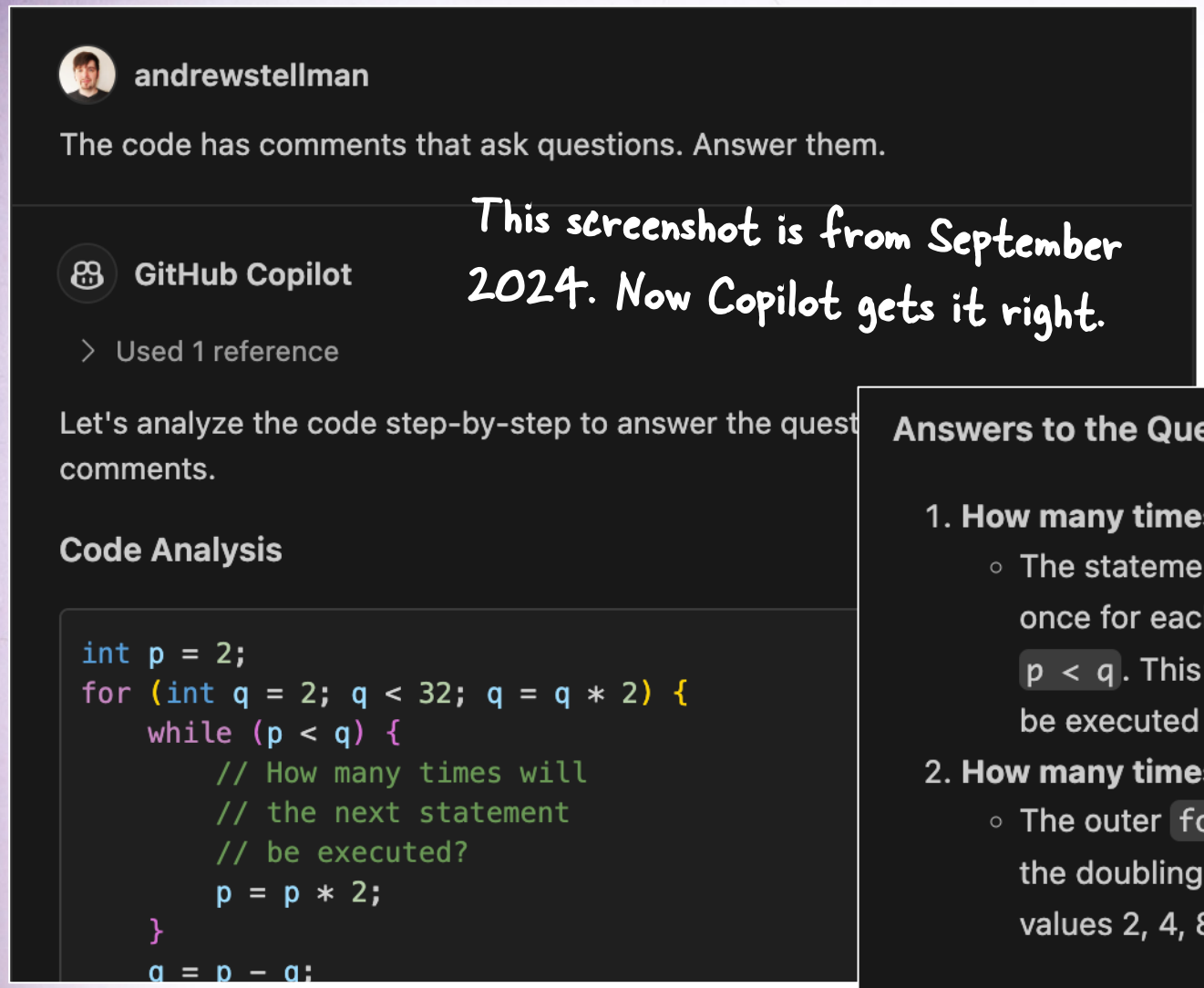
- This is an exercise from Chapter 2 of *Head First C#*. People who just learned about loops usually get it right. How do you think an AI does?

```
int p = 2;
for (int q = 2; q < 32; q = q * 2) {
    while (p < q) {
        // How many times will
        // the next statement
        // be executed?
        p = p * 2;
    }
    q = p - q;
}
// How many times will the loop run?
```

Answer 👍 if you think it gets the answer **right**



Answer 👎 if you think it gets the answer **wrong**



I entered the code into Visual Studio Code and used the GitHub Copilot extension to ask it to answer the questions in the comments. It gave the wrong answer. It always gives the wrong answer. It wasn't until mid-2025 that I saw an AI give the right answer to this question.

Answers to the Questions

1. **How many times will the next statement (`p = p * 2`) be executed?**
   - The statement `p = p * 2` inside the `while` loop will be executed once for each iteration of the `for` loop where `q` is a power of 2 and `p < q`. This happens when `q` is 4, 8, 16, and 32. Therefore, it will be executed 4 times.
2. **How many times will the loop run?**
   - The outer `for` loop will run until `q` is no longer less than 32. Given the doubling of `q` each iteration, the loop runs 5 times (for `q` values 2, 4, 8, 16, and 32).

Thus, the answers are:

1. The statement `p = p * 2` will be executed 4 times.
2. The outer `for` loop will run 5 times.

Not the right answer!

AIs may have started getting this right when the new edition of *Head First C#* was added to their training data.

GenAI isn't perfect (and that's okay!)

Let's start by understanding the capabilities—and *limitations*—of AI.

- **Mimicking Patterns:** LLM-based generative AI creates responses by mimicking patterns from the vast amounts of text they've been trained on, aiming to sound like a natural conversation.
- **Code Generation:** They use patterns from many code examples to "auto-complete" or suggest code snippets.
- **Perceived Understanding:** It's easy to think that an AI understands the content it generates or is pulling correct answers from a database. In reality, it's crafting responses based on learned patterns, not actual understanding or knowledge.

AIs don't inherently fact-check; they can produce convincing but sometimes incorrect or made-up answers. They do this a lot. ***A lot. ALL THE TIME.***

An example that shows how AI works

This screenshot is from 2024. ChatGPT now gives a more coherent answer.

A farmer needs to cross a river with a boat. He has a goat with him. The boat can only hold the farmer and one other item at a time. How does he get the goat to the other side of the river?

To get the goat across the river, the farmer can follow these steps:

1. **Take the goat across the river** and leave it on the other side.
2. **Return alone** to the original side.
3. **Take the boat back across the river** to the original side.
4. **Take the goat across the river** again.
5. **Leave the goat on the other side** and **return alone** to the original side.
6. **Finally, take the boat across the river** to the other side with the goat.

By repeating these steps, the farmer will successfully transport the goat across the river.

Actual screenshot of ChatGPT

The current generation of AIs often generate a complicated answer to a question about getting a goat across a river, usually involving multiple unnecessary crossings.

The AI generates this weird answer because its training data included thousands of examples of the classic "River crossing puzzle" that involves getting a goat, cabbage, and a wolf across the river. It uses the solution to that puzzle as a pattern to generate its answer.

Exercise

Let's see how smart AIs really are. Let's ask the AI to create code that tricks another AI—and then see if it actually works when we put it to the test.

Can You Trick the AI? (Part 1)

- Open an AI tool like ChatGPT, Copilot, Claude, or Gemini.
- Give the first prompt, choosing either C#, Java, or Python:

```
Create a short [C#/Java/Python] code example that will
cause an AI, if asked what the code produces, to
generate an incorrect answer.
```

- If it gives you an answer that's "likely" to fool an AI or will "often" fool an AI (or has similar language), ask it this:

```
Give me a code example that's 100% guaranteed to trip
up an AI if you ask it to tell you the output.
```

- If there's no explanation in the response, follow up with this prompt:

```
Explain why an AI will generate an incorrect answer if
you ask it the question that you generated.
```

Can You Trick the AI? (Part 2)

- Let's see how confident wrong answers hold up in another chat. Start by selecting the code sample the AI generated and copying it (if the AI you're using has a copy button, use that instead).
- Open another AI tool (like ChatGPT, Copilot, Claude, or Gemini).
- Start your prompt by entering this text:

```
What does the following Java code print, and why?
```

- Press shift-enter twice to add two lines. Then paste in the code you copied from the previous AI's answer.

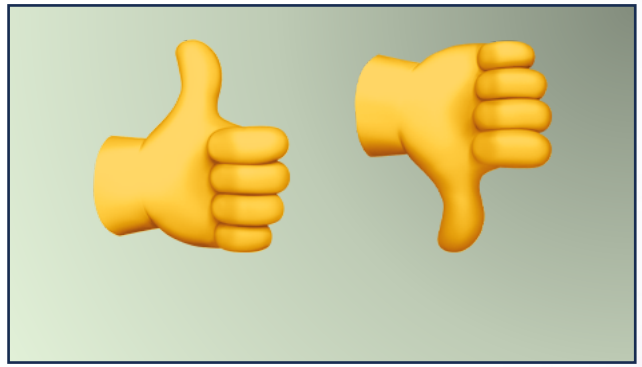
Pulse Check – What Happened?

It’s easy to assume AIs always give the right answer… until you test them.

This exercise is designed to help new AI users learn to question that assumption.

- Which AI sounded more confident?
- Which one was actually correct?
- ***Would you know if you hadn’t done this test?***

Answer 👍 if the first AI **successfully** tricked the second AI into getting it wrong



Answer 👎 if the first AI was **wrong** about being able to trick another AI

How to use AI safely: Trust but verify

AI doesn’t give you the right answer. It generates an answer that sounds right.

- **Verify Information Accuracy:** Always double-check facts and information provided by AI, as it can sometimes be incorrect or incomplete.
- **Test Generated Code:** Before using AI-generated code, thoroughly review and test it to ensure it works as expected and meets your requirements.
- **Understand Limitations:** Recognize that AI chatbots can misunderstand prompts or lack context, leading to unexpected results.
- **Critical Examination:** Treat AI responses as a starting point for further research and validation, not as the final word.

This is why we built Sens-AI.

Do you recognize the "fall from grace" pattern in the material we just went through?

Now we've seen what AI can do—and what it gets wrong.
We'll use that to build a way of using AI effectively.

Q&A

If you have any questions, ask them now. Make sure you enter them in the Q&A widget, because that makes it easier for me to keep track of them.

5 minute break

Grab some water, check your email, have a look at your socials… we'll be back here in five minutes.

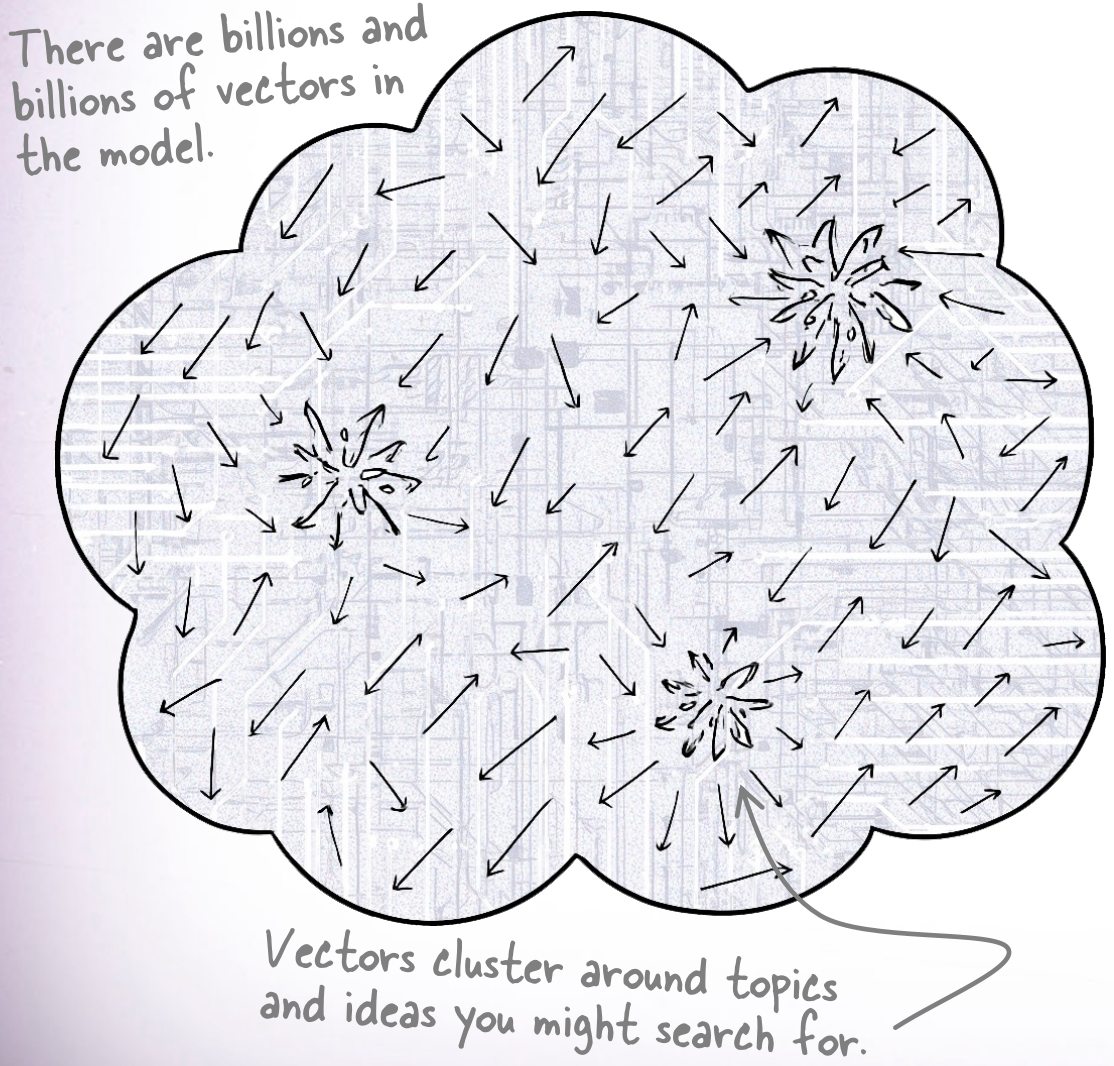
Part 2
Setting the AI Up to Succeed

Steering the AI toward useful, working code—because it won't figure it out on its own.

Context steers the model

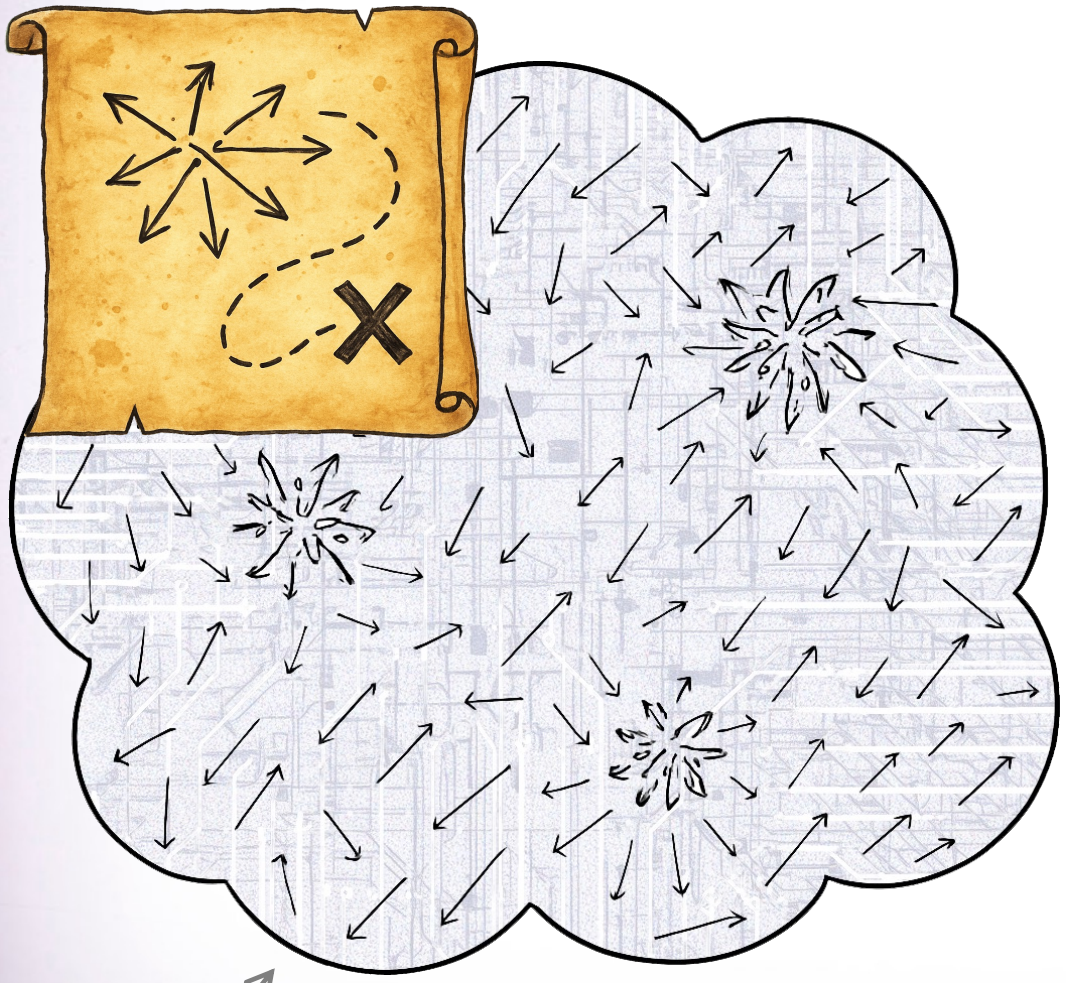
The AI is navigating a massive space of learned patterns. Your prompt is its only map.

The Model: A Vast Space of Interconnected "Ideas"



- The model *doesn't actually "know" anything*. It doesn't think, or understand, or form ideas.
- What it has is **patterns**—billions of them—learned from a massive mix of data: books, code, docs, blog posts, comics, movie scripts, poems, lyrics, and *so much more*.
- Those patterns form **clusters** in a giant cloud of vectors that *represent the ideas* in those sources.
- That's what the AI is **navigating** when it generates an answer.

Think of your prompt as a map



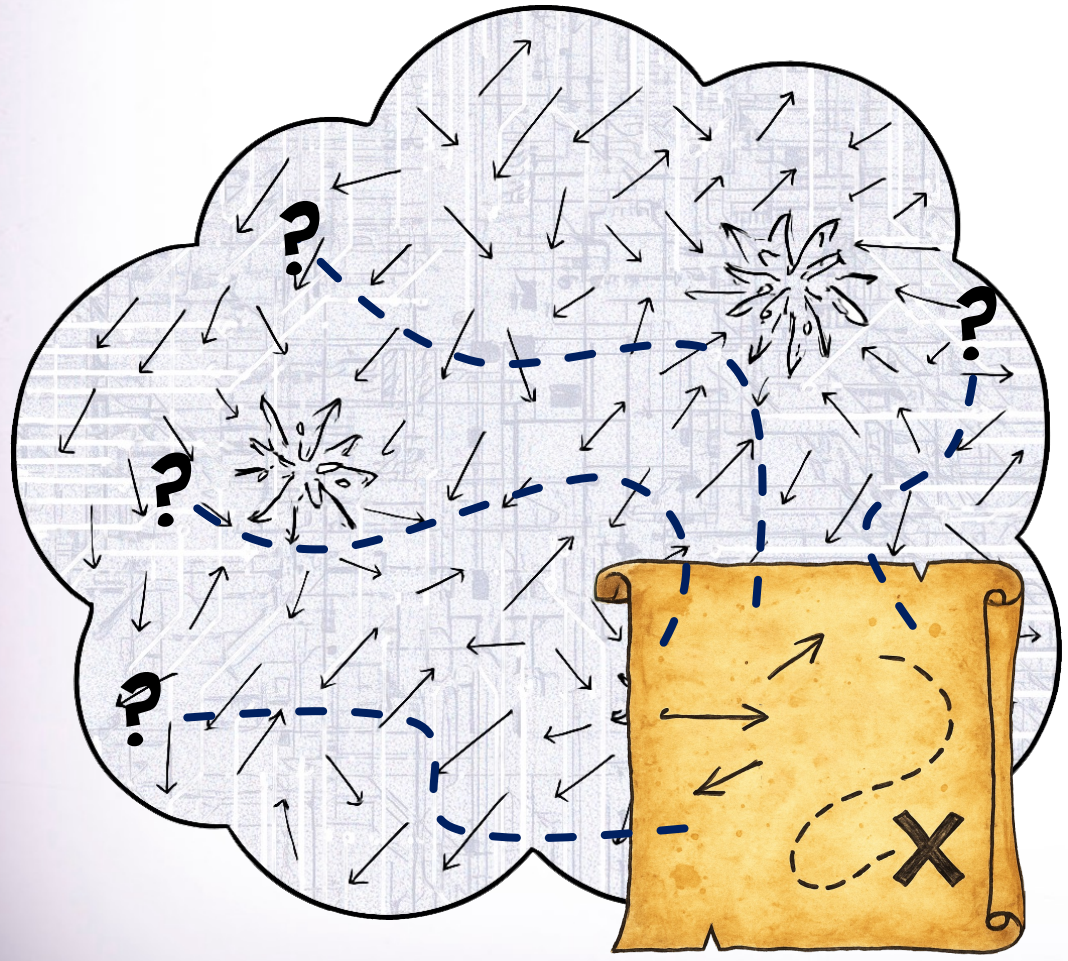
When you use an LLM, you have something you want to do:

- a question to answer
- a bug to fix
- a class to generate
- an email to explain
- a poem to write

Your **prompt** guides the AI to the right cluster of patterns—the part of the model that's most likely to produce something useful.

Technically, each vector represents a learned concept or token. The model's training created this space where similar ideas naturally cluster together.

Poor prompts don't tell the AI where to go

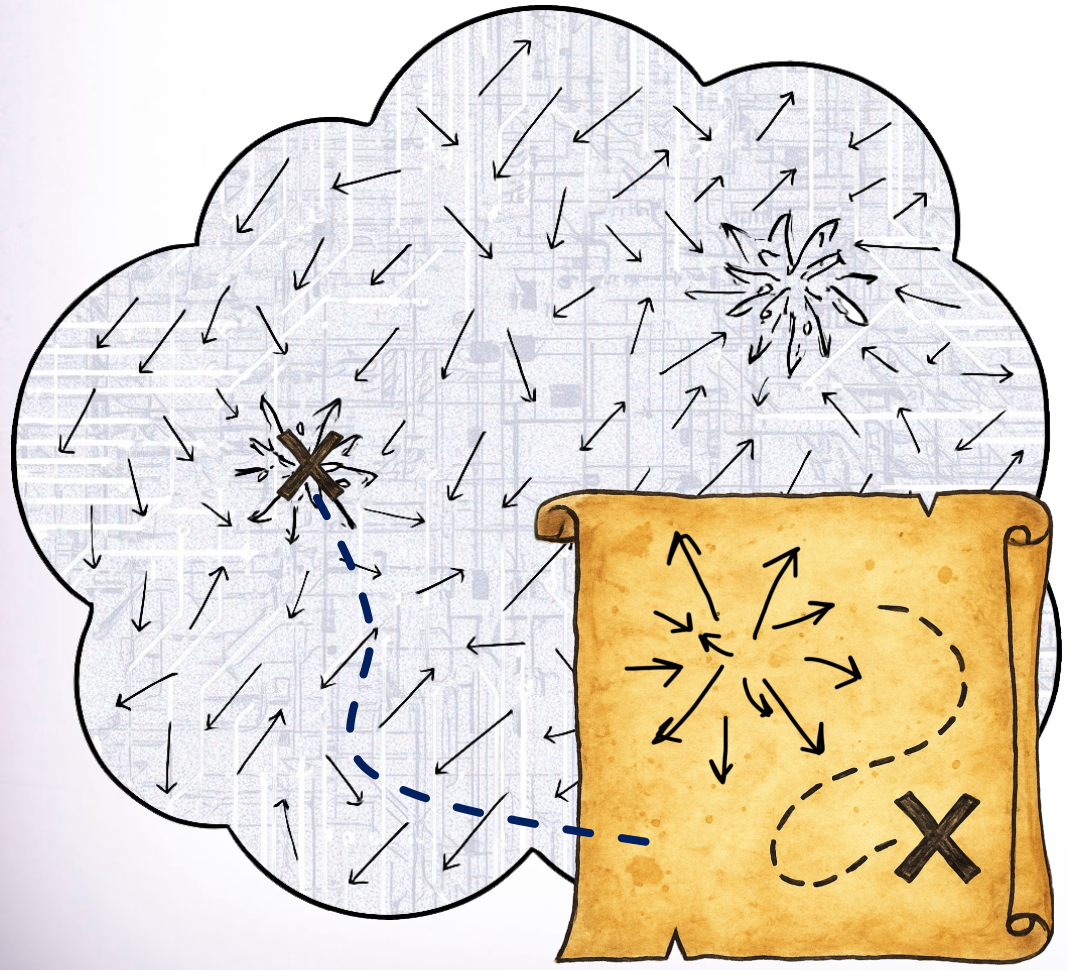


A vague prompt is like handing the AI a bad map. It might:

- Wander through unrelated clusters
- Chase surface patterns that sound right
- End up somewhere totally off

The model will still give you an answer, but that doesn't mean it found what you were looking for.

A good prompt maps a clear path

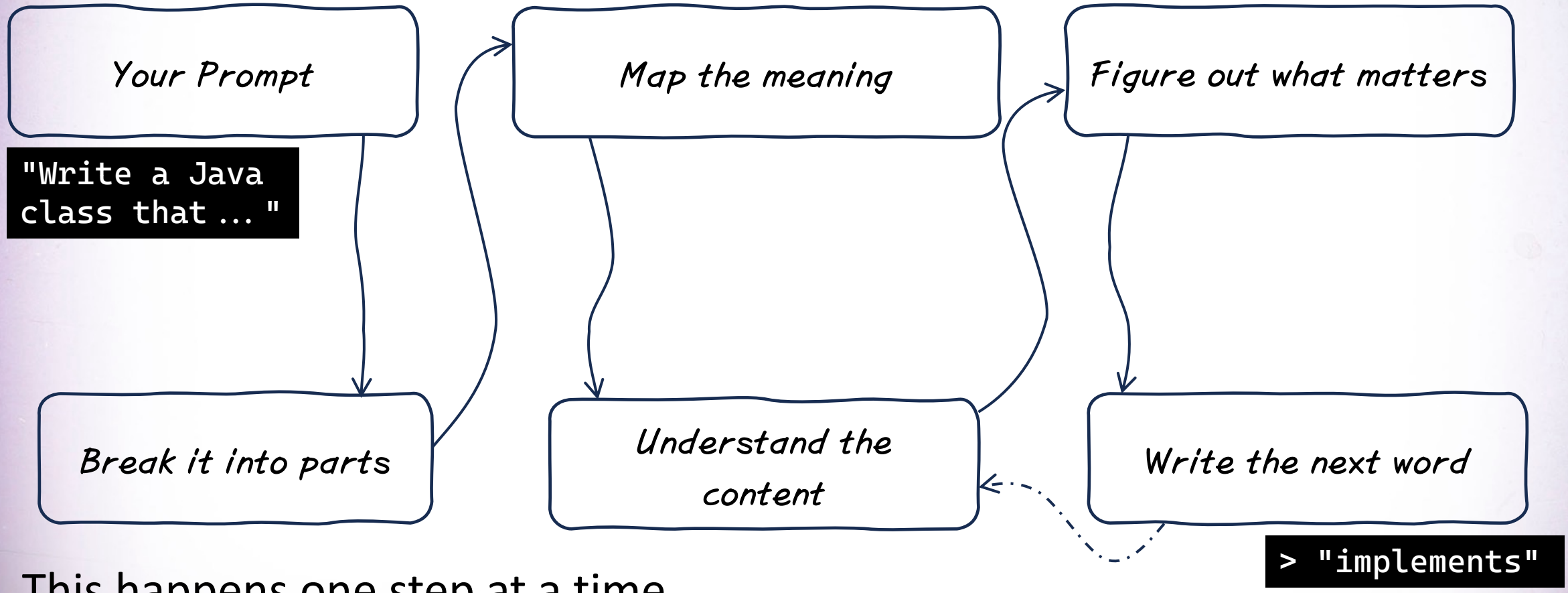


A good prompt gives the AI a direction to go. That means:

- The right context
- The right details
- A clear idea of what you're trying to get back

That's what helps the model land on something useful, instead of drifting through the wrong patterns.

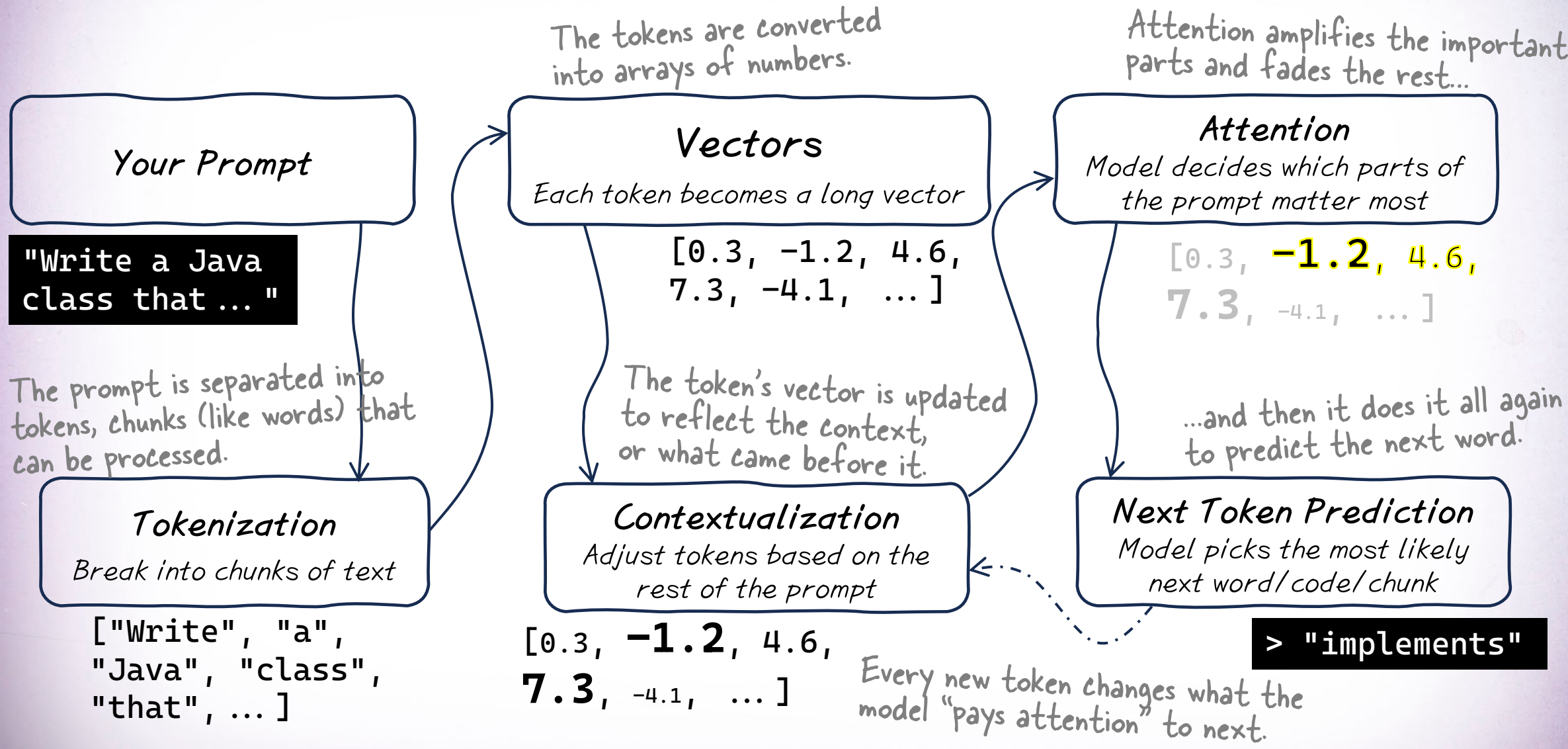
How an LLM processes your prompt



This happens one step at a time.

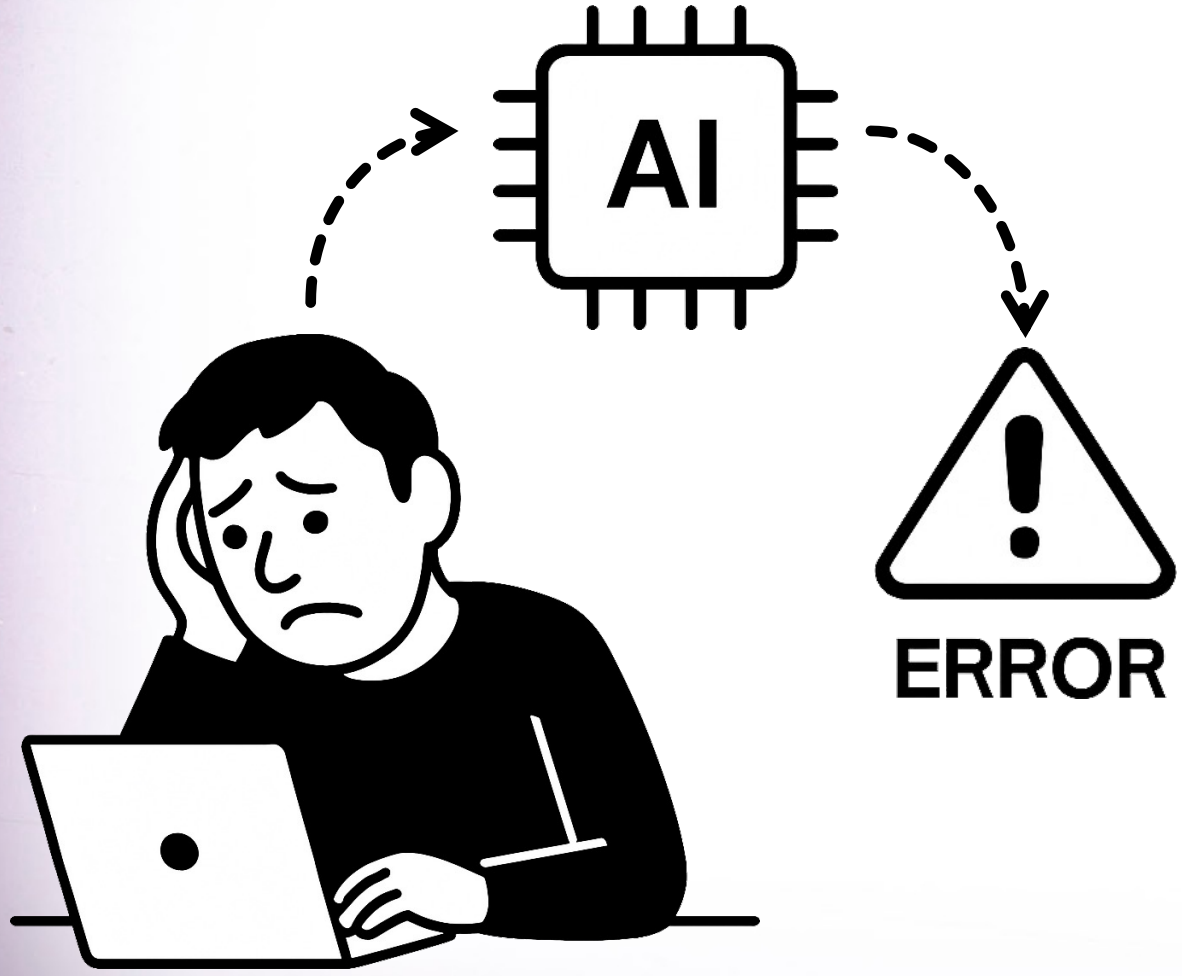
Every new word is based on everything it's written so far.

Processing the prompt – behind the scenes



This repeats for every new token in the response. It's always adjusting what matters, based on what it's written so far.

When the AI runs out of ideas…



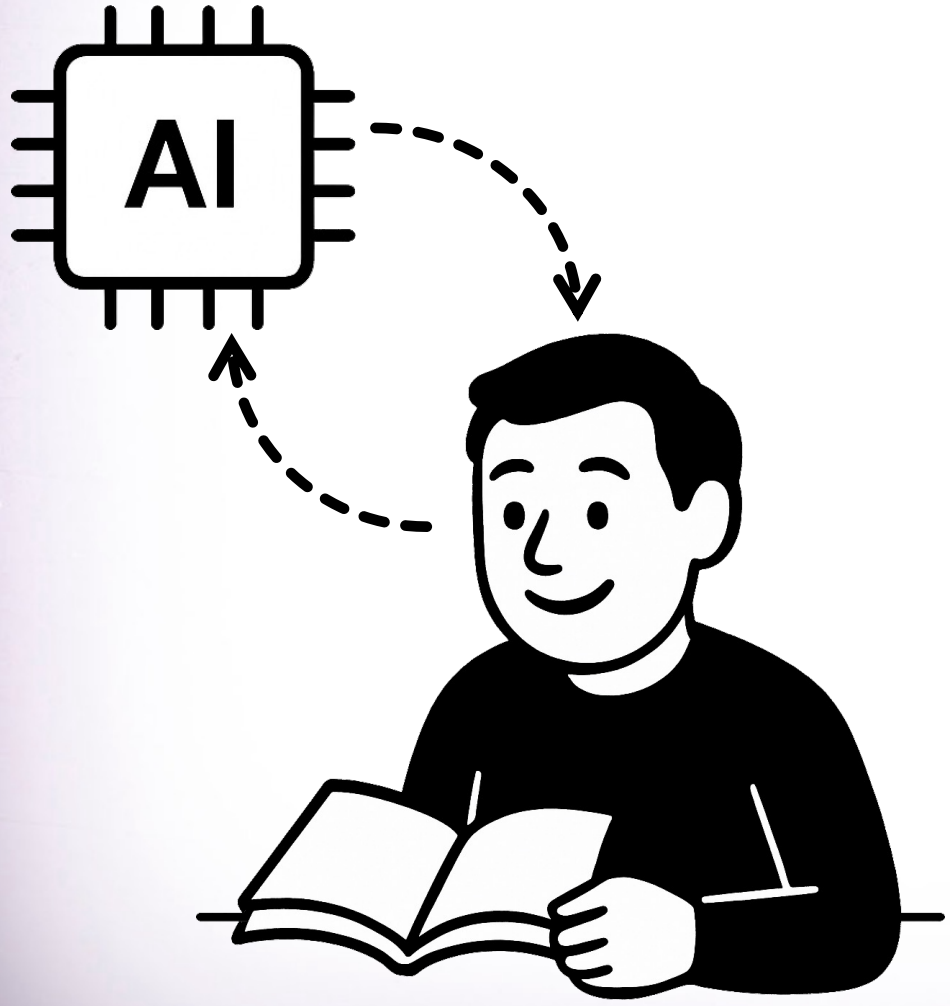
Back in Part 1 we talked about a **rehash loop**, where the AI keeps regurgitating what it already said.

We saw how that's an indicator that the AI ran out of context.

Now we know why: the AI ran out of patterns to draw from, and just kept going in circles.

The map led to a dead end.

…give it the context that it's missing.



The AI can only write what it sees in its context. If your prompt doesn't guide it to the right part of the model, it falls back on vague, generic patterns—causing a rehash loop.

Giving it the right examples helps it lock onto a better cluster and stay there. Sometimes you need to go out and find that new context yourself.

(That's what the next step in the Sens-AI Framework is all about.)

Ask yourself this question…

In the next section, we'll start with an exercise. While you're doing it—or watching me do it—*keep this question in mind*.

If the AI is **missing context**, and it's your job to **supply that context** so the AI can break out of its rehash loop…

Where does the new context come from?

Exercise

We'll ask the AI to explain a piece of code you already understand—then dig into how it came up with that explanation, and what it's missing.

Ask the AI to explain your code

- Pick a piece of code you've written (like a Java, C#, or Python class—just make sure it's code you already understand).
- Open a chatbot like ChatGPT or Claude, or use Copilot in your IDE. Give it this prompt.

  ```
  Add comments to this code that explain what it does.
  ```

- Press shift-enter twice to add two lines to the end of the chat, then paste in your code.
- Then read the results carefully. Does it make sense? Did it get anything wrong?

Understand what context the AI used to explain your code

- Add a follow-up prompt to your chat:

  ```
  What assumptions are you making about this code?
  ```

- Add another follow-up prompt to your chat:

  ```
  What information is missing that would help you understand this better?
  ```

Carefully read the AI's response—it's telling you what context you might need to give it next.

Research helps you send a signal

When your AI gets stuck, learning from real sources—including the AI itself!—gives it something solid to match and helps you fill in the context it's missing.

AI as a research tool

- AI chatbots excel at quickly summarizing complex topics and providing context, making them valuable for initial research or understanding new concepts.
- AIs are useful for getting rapid overviews or refreshing your knowledge on specific subjects, saving time in the research process.
- AI's ability to draw connections between different fields makes it great for understanding interdisciplinary relationships and broadening your perspective.

Think of AI as an intelligent research assistant: it's knowledgeable and efficient, but like any assistant, it can sometimes make mistakes or misinterpret information.

How to research effectively with AI

Approach your interaction with AI as an ongoing conversation. Continuously refine your questions based on the AI's responses to get increasingly detailed and relevant information.

- Start with broad questions to establish a general understanding, then gradually refine your prompts to dig deeper into specific aspects of the topic that you're researching.
- When something is too complex, ask the AI for simpler explanations, relevant examples, or comparisons to familiar concepts and ideas.
- Use AI to find related areas for further exploration, helping you build a comprehensive view of the subject and guiding your research path.

Keep AI research limitations in mind

- AI makes up 🐂💩. *All the time.* An AI will often generate information that sounds plausible but may be **inaccurate or even <u>entirely fabricated</u>**, especially for very specific or niche topics.
- AI models have a knowledge cutoff date and might not have the most up-to-date information, particularly for recent events or rapidly evolving fields.
- AI responses can exhibit biases present in their training data or inconsistencies between different queries, potentially leading to skewed perspectives.

There's a slide at the end of this deck with additional resources. I was researching books to include on that slide, and asked Claude for help. It generated one useful reference… and a giant pile of 🐂💩. And that's okay! It gave me a good start. It was useful! But I had to *carefully* review its output.

Actual screenshot of Claude

AS Here is the updated deck. Slide 56 is copied from an old presentation about Blazor. Replace it with an equivalent slide with additional resources for the material in this deck. Preferably use resources provided by O'Reilly on the O'Reilly Learning platform.

Certainly! I'll create a new slide with additional resources relevant to this course on prompting Copilot and ChatGPT for code, focusing on materials available on the O'Reilly Learning platform. Here's a revised version of slide 56:

Additional resources

Want to dive deeper into AI-assisted coding and prompt engineering? Here are some great resources available on O'Reilly Learning: Not the author or publisher of this book.

- "Building AI Applications with ChatGPT APIs" by Murat Durmus (O'Reilly, 2023): https://learning.oreilly.com/library/view/building-ai-applications/9781098152475/
- "Prompt Engineering for Generative AI" by Soyeon Caren Han, et al. (O'Reilly, 2023): https://learning.oreilly.com/library/view/prompt-engineering-for/9781098153427/
- "GitHub Copilot for Programmers" by Phil Haack (O'Reilly, 2023): https://learning.oreilly.com/library/view/github-copilot-for/9781098146849/
- "AI-Assisted Development" by Laurence Moroney (O'Reilly, 2023): https://learning.oreilly.com/library/view/ai-assisted-development/9781098152017/

These books don't exist at all!

- For more hands-on practice, check out O'Reilly's interactive labs on AI and machine learning: https://learning.oreilly.com/labs/

AI research best practices

- Approach AI-generated information with a healthy dose of skepticism, treating it as a starting point rather than the definitive truth.
- Make a habit of verifying key points and important information from authoritative and peer-reviewed sources to ensure accuracy.
- Use AI as a tool to guide and enhance your research process, but don't rely on it exclusively. Combine it with traditional research methods and critical thinking for the best results.
- Play around with how you ask your questions. Sometimes, just rewording things a bit can unlock new insights or show you angles you hadn't thought of before.

AI is a powerful and efficient tool for research, but it's most effective when combined with critical thinking skills and traditional research methods.

Exercise

Every interaction with AI is an opportunity to do research. You'll try using AI to improve your own code. Then you'll dig into why it made those changes.

Ask the AI to refactor your code

- Choose classes you've written (in Java, C#, or Python), or go to https://bit.ly/sens-ai-course and copy our sample. Make sure it's real code (not AI-generated) that you understand.
- Enter this prompt into the AI's chat:

```
Refactor this code to improve encapsulation and
make it easier to maintain.
```

- Press shift-enter twice to add two lines to the end of the chat, then paste in your code.
- Does it follow good practices? Did it change anything that affects how the code works?

Learn how the AI refactored your code

Do some research and dig into the refactoring the AI just did.

- Ask the AI to explain the principles it followed:

```
What design principles did you use to refactor?
```

- Then add a follow-up prompt:

```
What are the benefits and risks of changing the code this way?
```

- One more:

```
What additional information would help you improve this refactor?
```

Pulse Check – What Happened?

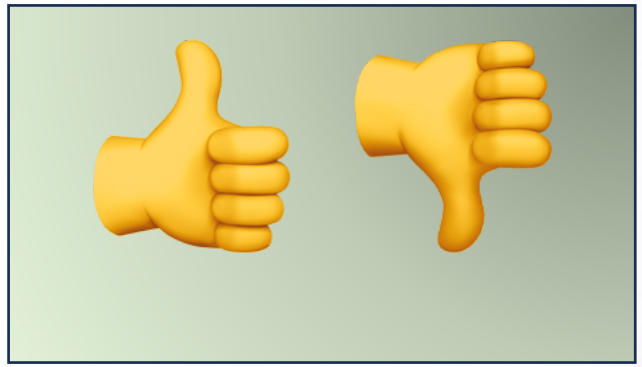
In that last exercise, you gave the AI real code and asked it to improve it.

Then you pushed it to explain why it changed what it did.

That's a very different interaction than just saying "refactor this."

- If the AI had been stuck in a rehash loop, **could asking it to explain its thinking have helped**?
- Would that **change how you write your next prompt?**

Answer 👍 if you think asking the AI to explain might help it get unstuck.



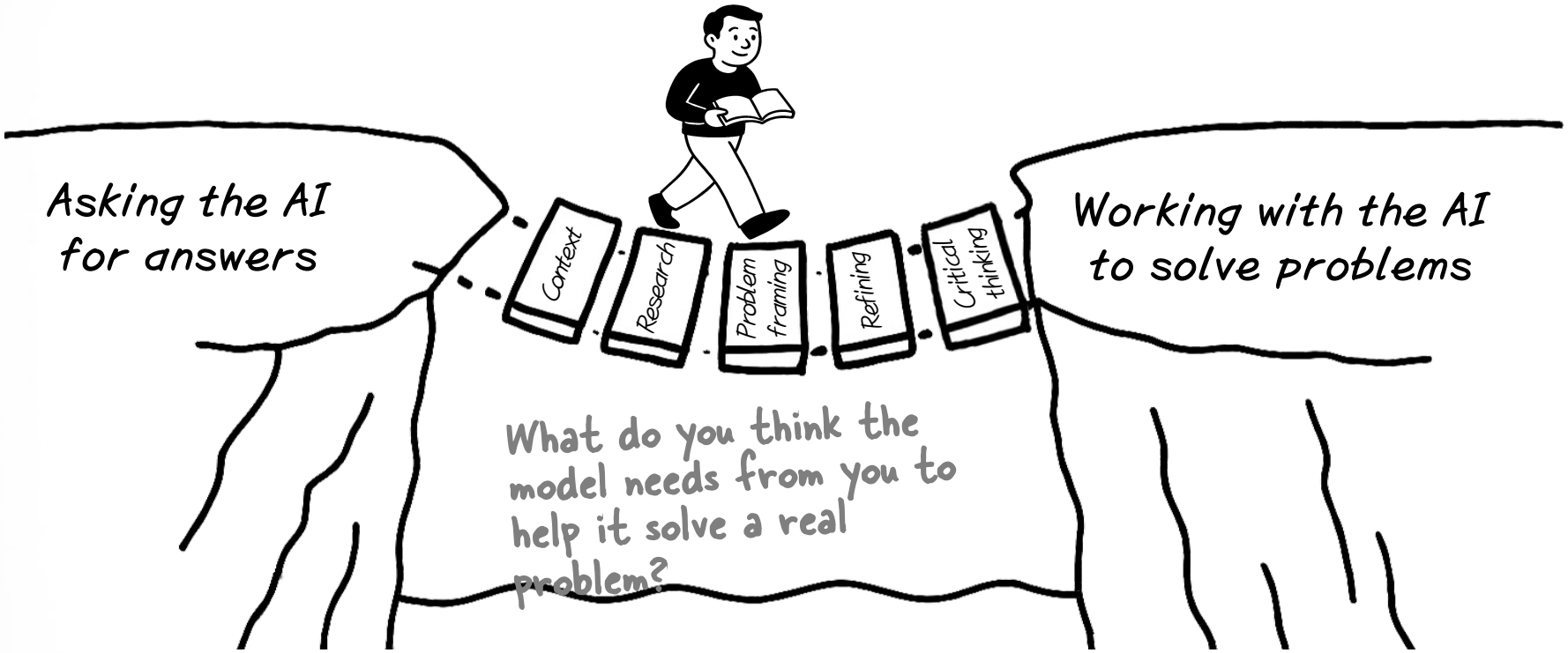
Answer 👎 if you think it would still stay stuck in the rehash loop.

How to use research to guide the AI

The AI doesn't understand your code. It matches patterns from training data. Research helps you fill in context and gives the AI a better pattern to match.

- **Start with code you understand.** That's the clearest context you can give—and makes it easier for you if the AI gets stuck.
- **Ask the AI to explain itself.** This helps it surface assumptions.
- **Use research to strengthen your next prompt.** Give it real design principles, patterns, and examples to follow.
- **If the AI gets stuck, add more context.** Changing how you ask the question and giving the AI one more piece is often all it needs.

Now you're using the Sens-AI Framework.



You've used context and research to guide the model. Let's use that knowledge to ask the right questions.

Q&A

If you have any questions, ask them now. Make sure you enter them in the Q&A widget, because that makes it easier for me to keep track of them.

5 minute break

Grab some water, check your email, have a look at your socials… we'll be back here in five minutes.

Part 3
Questions, Judgment, and Making It Work

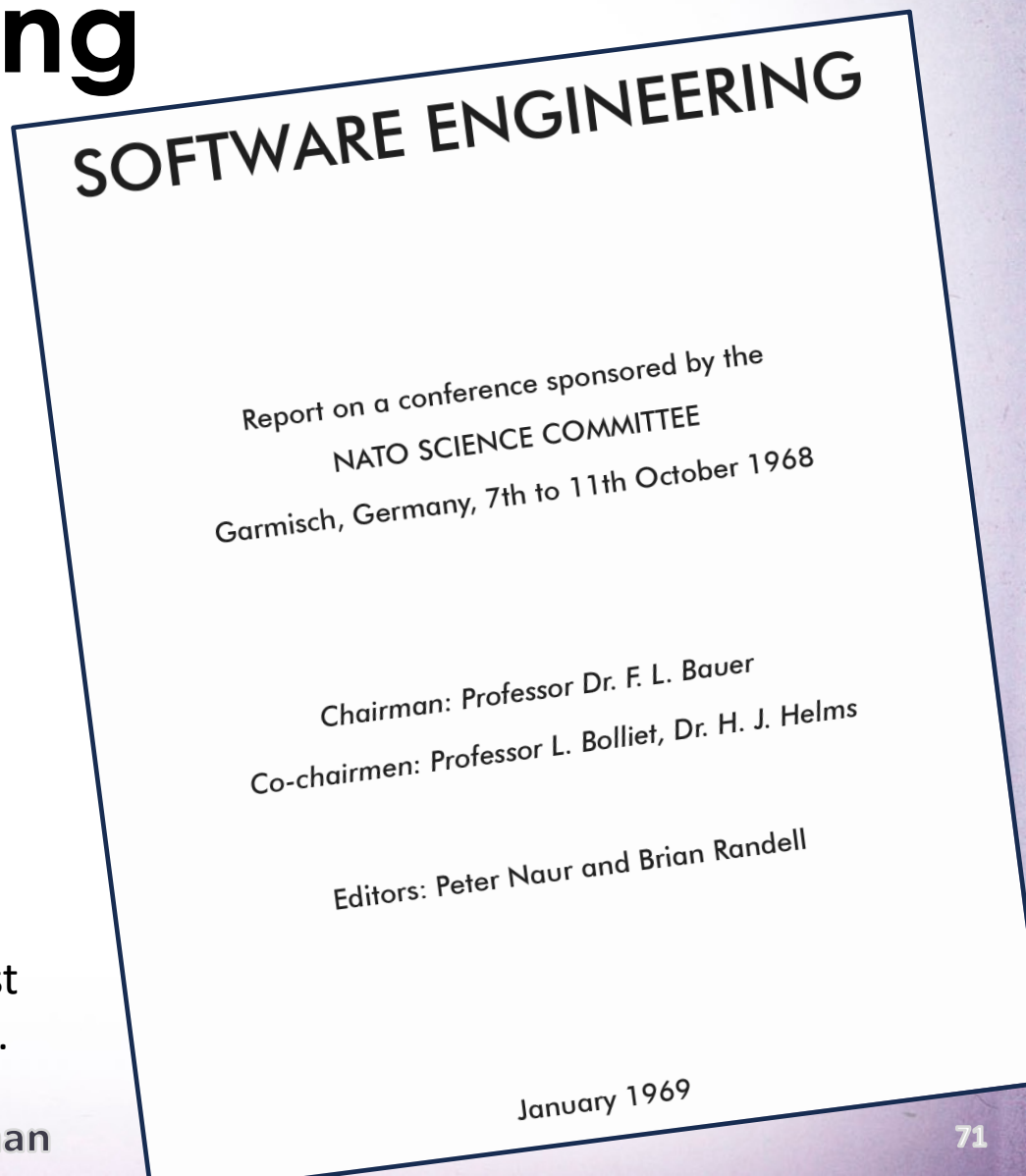
Asking better questions, refining what the AI gives you, and deciding for yourself what to trust.

Problem framing

AI can only work with the problem you give it. When you describe the problem clearly enough for the AI, you help it find a useful solution.

Over fifty years later, and we're still asking for the wrong thing

1968 NATO Conference, Software Components Lecture - M.D. McIlroy



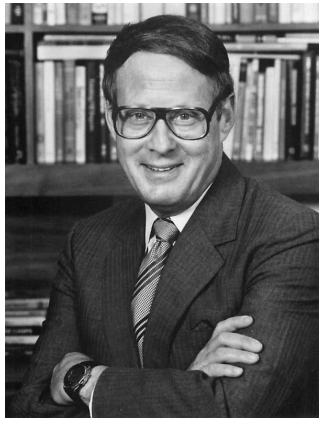
Explaining what we want to build has always been one of the hardest parts of software engineering... and now we're facing it again with AI.

This isn't a new problem

Back in the 1960s they realized the "software crisis" was caused by teams having difficulty describing the problems to solve. We're still facing the same exact problem—just with AI!

A few quotes from the *1968 NATO Conference on Software Engineering* report…

There is, in many instances, **no way even to specify** in a logically tight way what the software product is supposed to do or how it is supposed to do it.
— Dr. Edward E. David Jr., Bell Labs



New York Times

IT History Society

The most deadly thing in software is the concept, which almost universally seems to be followed, that you are going to **specify what you are going to do, and then do it**.
— Douglas Taylor. Ross, MIT

So when a problem is specified in all detail the formulation can be mapped into the solution; but **most problems are incompletely specified**. Where do you get the additional information to arrive at the solution which includes more than there was in the first specification of the problem?
— Prof. Dr. W.L. van der Poel, Technische Hogeschool

Wikipedia

Developers crafting AI prompts run into *exactly the same problem* today.

The “Do What I Meant, Not What I Said” Problem

This is problem has been frustrating development teams since before the term “software engineering” was even invented.

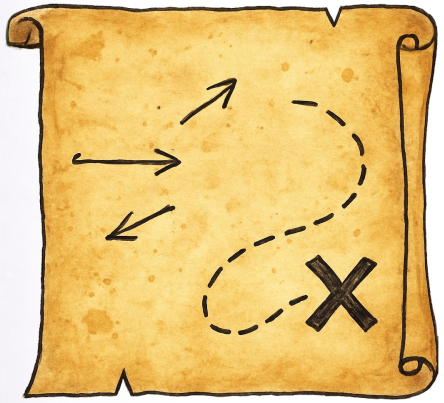
You told it what to do, and it *technically* did—but not how you meant.

- If your prompt is fuzzy, the pattern that the AI picks up will probably go somewhere you didn’t expect.
- The AI wasn’t “wrong.” It just did exactly what you said.
- AI doesn’t know your intent. It just matches the words you gave it.
- This is the **same problem** developers have always had when requirements are unclear.

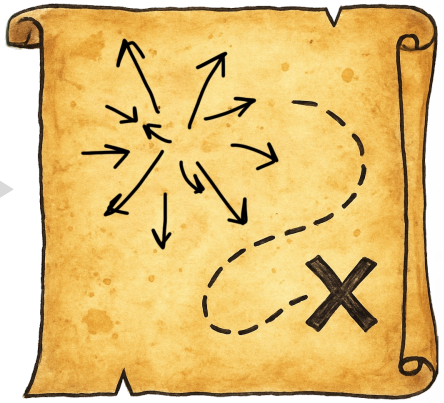
*AI **doesn’t know your intent**. It just matches the words you gave it.*

What Is Problem Framing?

Problem framing means guiding the AI toward the right solution by describing the problem in a way it understands.



```
Here's what I'm trying to do
Here's what might make it tricky
Here's what it should look like
```



Bad framing = less effective map
The AI still tries to match a pattern, but that's not leading it to the right place.

Good framing = a better map
Giving the AI a clearer goal, better structure, and context helps it figure out where to go.

How to frame your problem effectively

Here's what I'm trying to do

- Give the AI a clear, specific goal. If you're asking a question, make it clear what kind of answer you're expecting.

Here's what might make it tricky

- Point out anything that might confuse it, like edge cases or ambiguity.
- Help it avoid obvious mistakes by flagging what not to do.

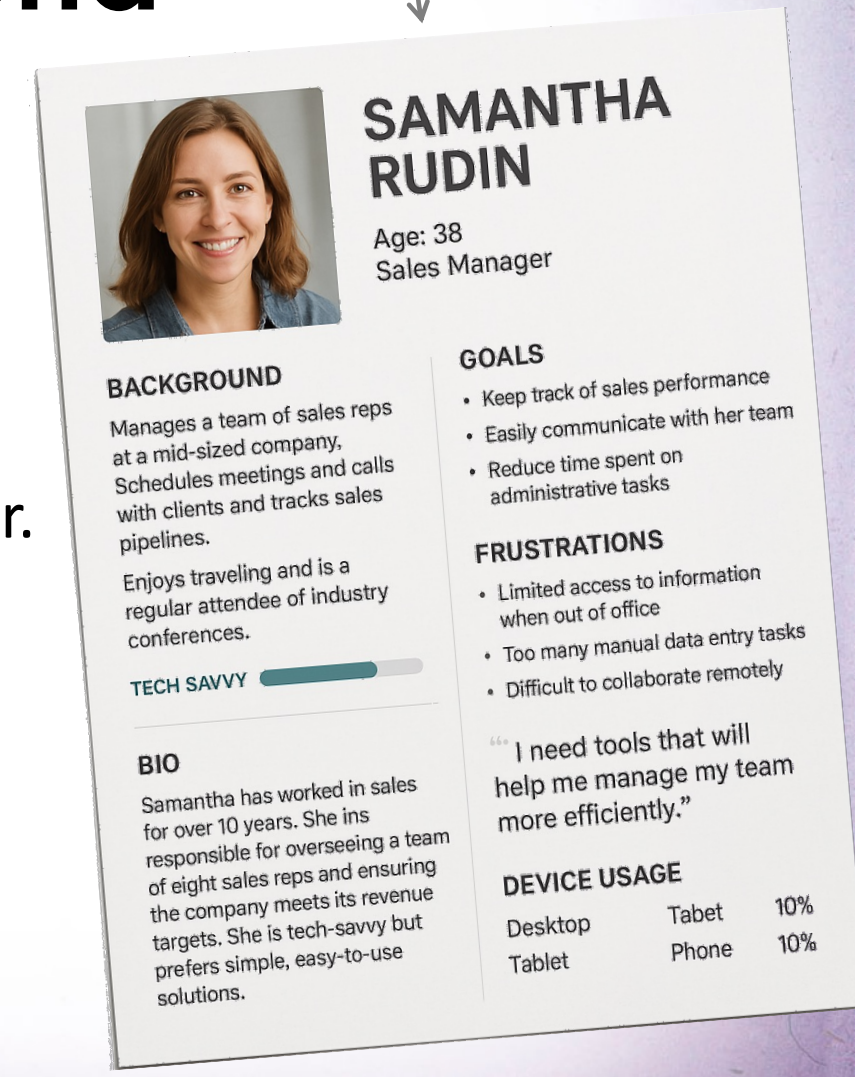
Here's what it should look like

- Use examples to shape the structure and style of the response.

Problem framing helps you avoid "do what I meant, not what I said."

Problem framing technique: Tell the AI to adopt a persona

When UX engineers create personas, they often look like this.



UX designers use ***personas*** to represent the people they're designing for. When we talk to AI, we flip that idea: **you're designing who the AI should act like.**

- **Personas shape expectations.** UX personas help teams focus on real user needs. Giving the AI a persona tells it *how* to think, not just *what* to answer.
- **It sets the tone and level.** Saying "act like a senior developer" gives you more grounded, practical advice than just "explain this."
- **It triggers better patterns.** Framing the prompt this way steers the AI toward more relevant analogies, code, and structure.

Using a persona to shape the answer

The persona "senior C# developer mentoring a junior" sets tone, depth, and intent. Now the AI's not just explaining—it's teaching.

"Explain how interfaces work" is a clear request, with just enough scope to guide the AI's answer.

```
You're a senior C# developer mentoring a junior. Explain how interfaces work, with a real-world example and a short analogy.
```

"real-world example and short analogy" tells the AI how to structure the response and anchors the pattern.

Exercise

Let's get some practice using problem framing to improve an ineffective prompt.

Start with an ineffective prompt

- Open an AI and enter this prompt into the chat:

  ```
  Write a method that takes a string and returns a summary.
  ```

- Look at what it generates. Think about why it might not match what you meant.
- Start a new chat and enter the same prompt.
  - Did it use the same language?
  - Did the code do the same thing?
  - Try it a few times. Read the code. What changed?

Improve the prompt by reframing the problem and using a persona

- Start a new chat session and give the AI an improved version of the prompt (choosing either C#, Java, or Python):

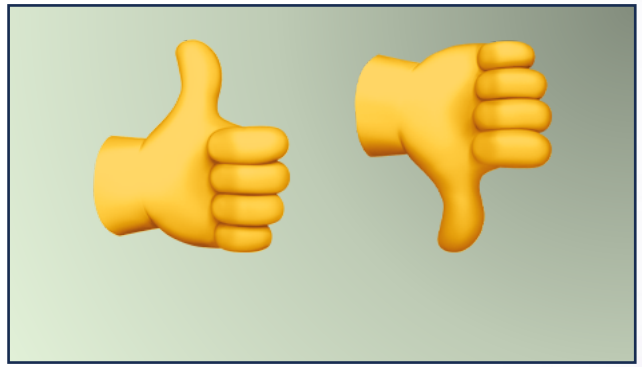
```
You're a technical writer working on [C#/Java/Python]
tutorials. Write a method that summarizes a string by
keeping only the most important sentences. It should
work on plain text—not depend on external APIs or
machine learning libraries.

Before you write the code, show me two example inputs
and outputs so I can see how the summary should work.
```

- Read the example inputs and outputs, then tell the AI to go ahead.
- Start another chat session and enter the same prompt. Ask it to make some changes to the inputs and outputs. What effect does that have?

Pulse Check – Did problem framing help?

- **The persona helps the AI "think" more like a developer.** Framing it as a technical writer changed the tone, depth, and priorities of the answer.
- **The examples guided the shape of the code.** Giving inputs and outputs helped the AI pick a pattern that matched what you wanted.
- **Framing gave you influence, not just output.** It worked because you weren't simply requesting code but designing the AI's process for generating it.

Answer 👍 if you felt like you had more control over the AI's response with the second prompt.



Answer 👎 if you felt the first prompt gave you more control and produced more consistent results.

Refining

Prompting is iterative because the way we think about our problems is, too. We learn more about the problem with every small change to the prompt.

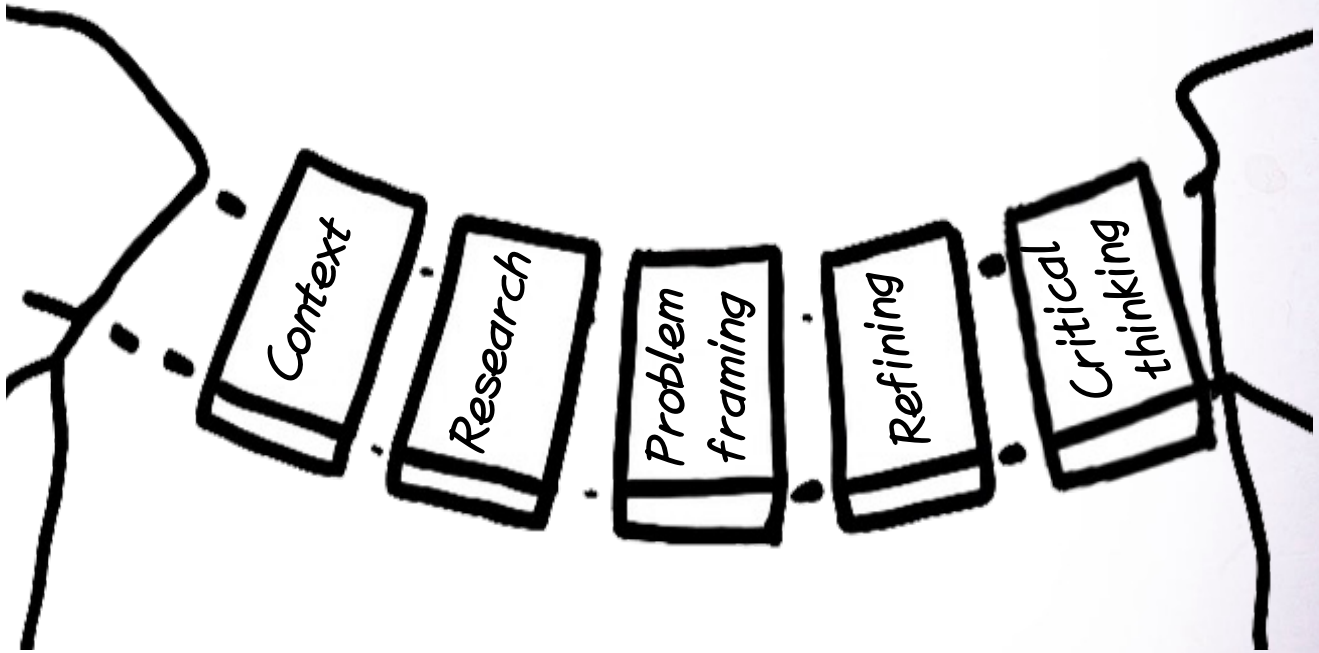
The Sens-AI Framework Is a Bridge, Not a Flowchart

You don’t cross the bridge one plank at a time. **You build it up as you go.**

These aren’t steps to follow. They’re **habits** you build and **techniques** you reach for when you need them.

You’ll loop back, skip ahead, combine parts—whatever gets the job done.

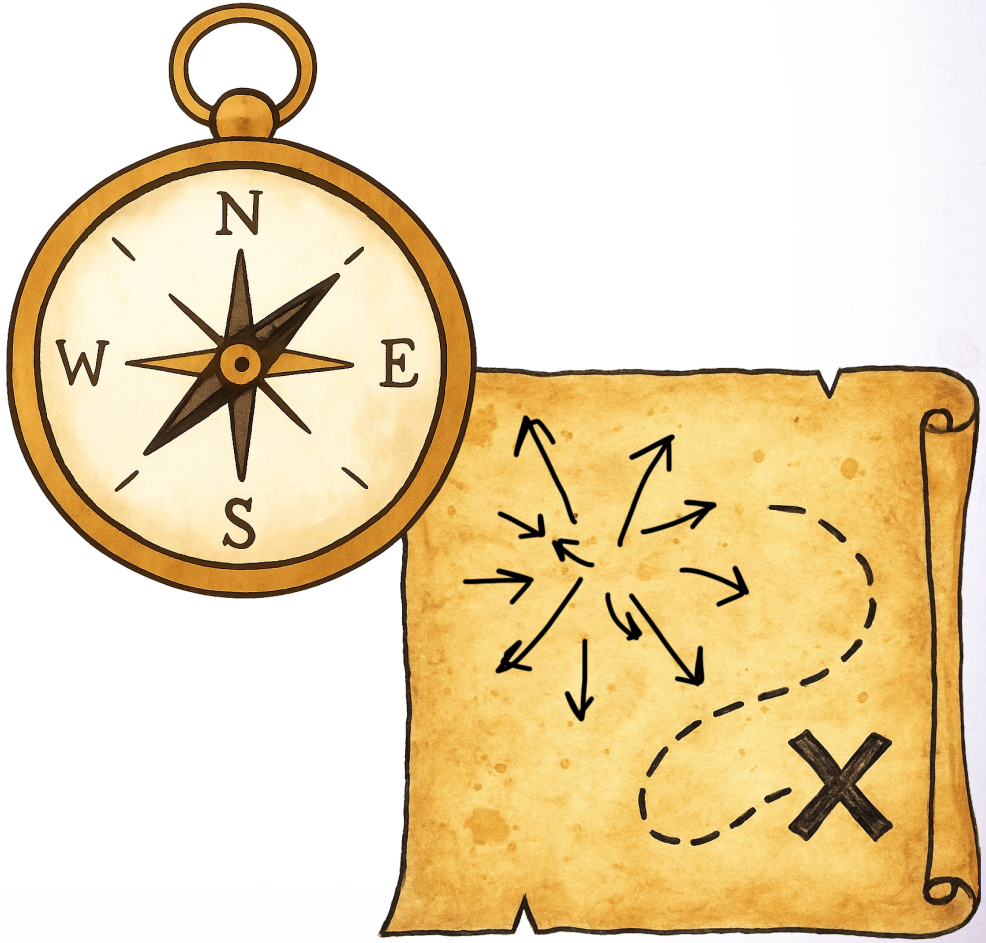
What makes the whole thing work is how all the pieces **reinforce each other**.



Refining means adjusting course as you go

Use the AI's response to steer it.

- **Refining** is the habit of correcting course, one prompt at a time.
- You don't always get it right the first time, and neither does the AI.
- The model won't always interpret your prompt the way you meant.
- Every response gives you clues about how it "read" your intent.



The prompt is a starting point, not the destination

Prompt

Write a Java method that filters numbers in a list, ignoring any invalid ones.

AI Response

```
public static List<Double> filterList(List<String> list) {
    List<Double> numbers = new ArrayList<>();
    for (String s : list)
        if (s.matches("-?\\d+(\\.\\d+)?"))
            numbers.add(Double.parseDouble(s));
    return numbers;
}
```

The model gave us a technically correct answer. That's how we realized we hadn't been specific about what "invalid" means, or what's in the list we're providing.

Refining helps the model adjust and helps us think more clearly

Prompt

The list contains strings like '87', '94.2', 'error', or 'NaN'. Write a method that parses the numbers and returns a list of values between 0 and 1000. Skip anything invalid.

Response

```java
public List<Double> filterList(List<String> list) {
    List<Double> numbers = new ArrayList<>();
    for (String s : list) {
        double value = Double.parseDouble(s);
        if (value ≥ 0 && value ≤ 1000)
            numbers.add(value);
    }
    return numbers;
}
```

Seeing the first answer helped us to learn more about the problem, so we could guide the AI more accurately.

Each revision brings you closer—to the solution *and* the problem

Prompt

This is for a sensor reading system. Valid values are between 0 and 1000. Negative values mean errors. Over 1000 is noise. Parse the list and return only valid numbers.

Response

```
public List<Double> filterReadings(List<String> list) {
    List<Double> validReadings = new ArrayList<>();
    for (String s : list) {
        double value = Double.parseDouble(s);
        if (value >= 0 && value <= 1000)
            validReadings.add(value);
    }
    return validReadings;
}
```

Adding the use case improved the output—and helped us better understand the problem we're actually solving.

A few tips for refining your prompts

- **Read the model's response like a teammate's first draft**
  → What's missing? What did it assume?
- **Clarify your intent before adding more detail**
  → Are you asking for structure, behavior, or a pattern?
- **Use examples to disambiguate vague language**
  → Phrases like "invalid" or "edge case" often need context
- **Explain what makes the task tricky**
  → Highlight the thing that's most likely to go wrong
- **If you're not sure what you need, say so**
  → Use the model to explore, not just execute

Exercise

Use AI to research something new about your programming language. You'll explore a topic, refine your understanding, and uncover the questions you didn't know you needed to ask.

Research a programming topic

You won't always find the answer you need in the docs. That's where AI can help—but only if you treat it like a research tool.

- **Start with a real question.**
  Start by deciding on a topic you're actually trying to learn. Be specific enough to search—but open enough to explore.
- **Try the same question in different ways.**
  "Explain like I'm a junior dev." → "Pretend you're reviewing this code."
- **Follow up and ask for context you're missing.**
  "Why would someone use this in a real system?" is often better than "What does this do?"

Frame the problem for a better answer

Remember, you're just working with the AI not just to find the answer to your question, but to figure out what question you're asking.

- **Choose a persona that matches how you think.**
  A casual tone with silly examples might help some devs. Others do better with tight, serious code.
- **Try setting constraints on how it answers.**
  "Give me a short bullet list." → "Write full paragraphs." Both can clarify your thinking.
- **Clarify your intent before diving into detail.**
  Are you asking for structure? Behavior? Tradeoffs? You won't get a great answer if you don't know what you want yet.

Keep refining until you're satisfied

Every revision helps you learn more: about the model, the task, and the question you're actually trying to ask.

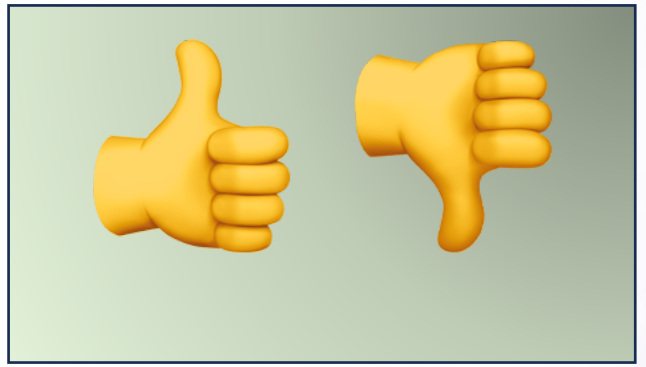
- **Ask follow-up questions to explore edge cases.**
  "What if the list contains nulls?" → "What happens with very large inputs?"
- **Use examples to pin down vague terms.**
  If you say "invalid," show what you mean. If you say "slow," give a number.
- **Refining isn't just about fixing the model's answer.**
  It's a way to understand the problem better—one step at a time.

Pulse Check – How did that feel?

You combined habits from Research, Context, Problem Framing, and Refining… whether you realized it or not.

Did the pieces feel like they worked together?

Answer 👍 if it helped to approach the problem from multiple angles.



Answer 👎 if it still felt a little disconnected or awkward (which is okay!).

Tips for breaking problems into AI-friendly chunks

Not every prompt needs to solve everything. Break it down.

- **Start with just one part of the task.**
  Method, class, use case—smaller asks make cleaner answers.
- **Highlight what's tricky.**
  The edge cases or ambiguous logic are usually where things fall apart.
- **Use the model's response to decide what to ask next.**
  Your goal isn't simply getting the problem "right" up front, the first time. As you interact with the AI, you learn what's missing.
- **Build context as you go.**
  Each response teaches the AI—and you!—more about your problem.

Critical thinking

Critical thinking is what turns AI from a shortcut into a real tool. It helps you keep control and question the output so you can design better solutions.

Critical Thinking Is Both the Easiest and Hardest Part of This Framework

- Critical thinking is the *most natural skill for developers*—you already do it every day when you write, review, and debug code.
- But it's also the **hardest part to apply with AI**, because the vibe coding loop can switch you out of critical thinking mode.
- AI makes it easy to accept outputs without questioning them, offloading decisions you'd normally inspect and keeping you in the "vibe" flow.
- In fact, some studies[1,2] suggest that over-reliance on AI can reduce engagement, make us less reflective, and weaken independent problem-solving. ***That's what happens when you stay in the vibe coding loop.***

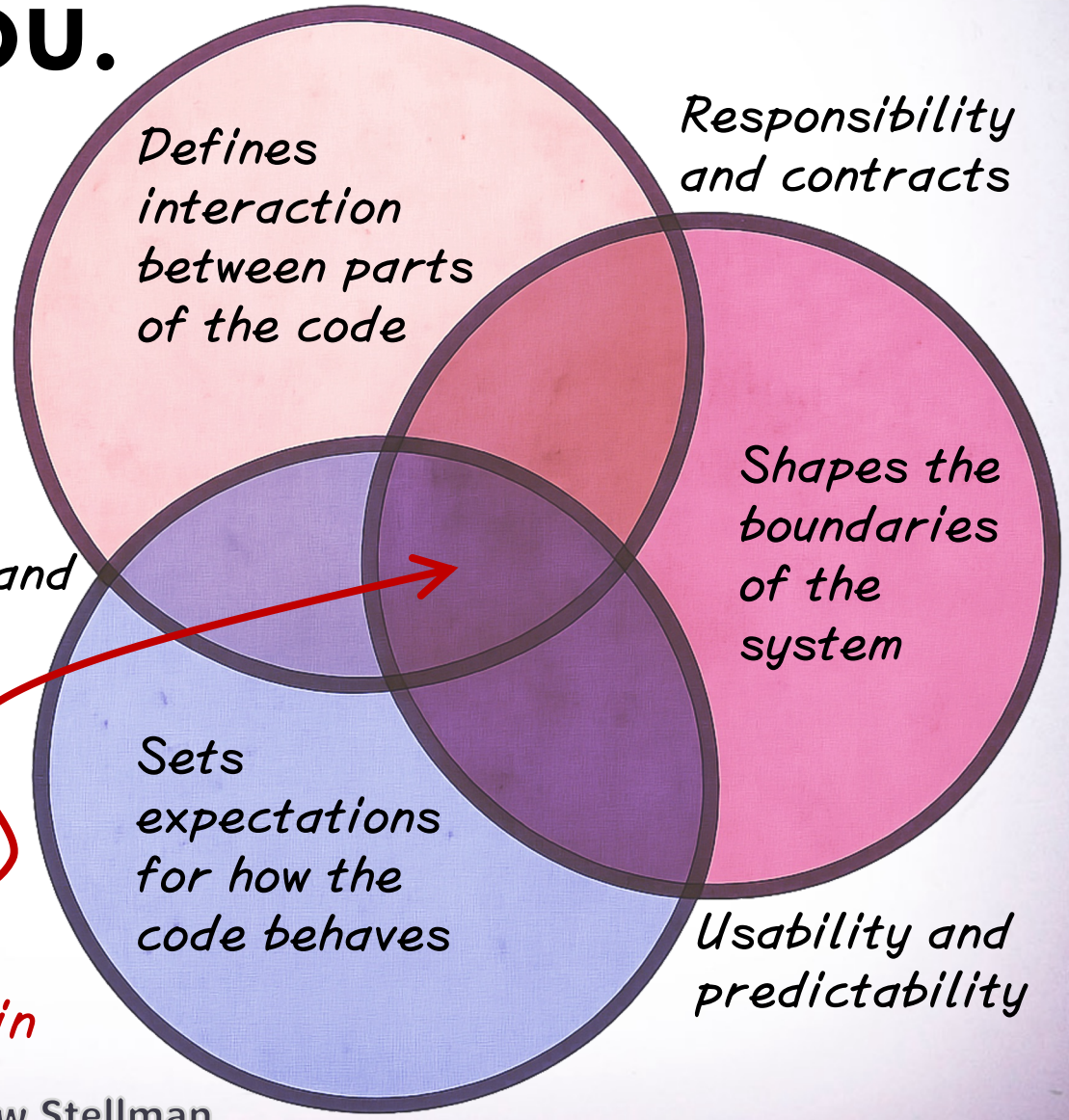
[1] Gerlich, Michael. "AI Tools in Society: Impacts on Cognitive Offloading and the Future of Critical Thinking." *Societies*, vol. 15, no. 1, 2025, article 6, MDPI, doi:10.3390/soc15010006 (https://www.mdpi.com/2075-4698/15/1/6)

[2] Royce, Christine Anne, and Valerie Bennett. "To Think or Not to Think: The Impact of AI on Critical-Thinking Skills." *NSTA Blog*, National Science Teaching Association, 10 Mar. 2025 NSTA (www.nsta.org/blog/think-or-not-think-impact-ai-critical-thinking-skills)

You don't have to read every line of code the AI gives you.

AI generates a lot of code (and you end up throwing out a lot of it). Trying to read it all can lead to ***cognitive overload***.

But there are parts of the code that are **especially important to read**: code that sets boundaries, determines the design, shapes the architecture of the system. That's where design decisions hide and technical debt grows.



Unit tests can reduce cognitive overload

When you generate a new class, have the AI create a unit test for it.

- AI generates a lot of code, and it's easy to get overloaded deciding what to pay attention to. (That’s why the vibe coding cycle works.)
- Unit tests are about more than quality and finding bugs. They also help you **create decoupled code with clear boundaries**, make design assumptions explicit, and reveal hidden dependencies.
- Unit tests help you figure out which parts you can safely set aside.

You don't have to keep every line in your head if you know a good test protects it from rippling changes that can break other parts of the code.

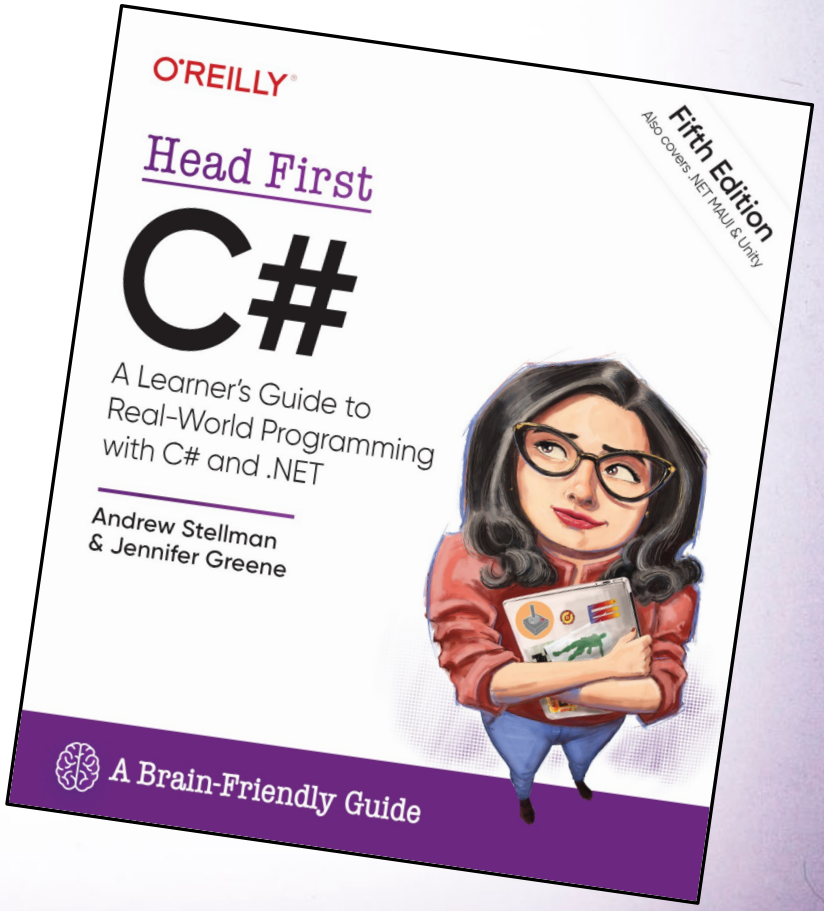
If tests are hard for the AI to generate, that's a signal to *stop and think*

Adding unit tests to your vibe code cycle creates a signal to stop just accepting what the AI gives you and start critical thinking.

- When you ask the AI to help write unit tests for generated code, first have it **generate a plan** for the tests it's going to write.
- Watch for signs of trouble: lots of mocking, complex setup, too many dependencies—**especially** needing to modify other parts of the code.
- Those are clues **the design is too coupled or unclear**.
- When you see those signs, **stop vibe coding and read the code**.
- Ask the AI to explain it. Run it in the debugger. Stay in critical thinking mode ***until you're satisfied with the design*.**

Critical thinking was the driving force behind the Sens-AI Framework

- My goal was to add AI to *Head First C#* in a way that actually helped people learn.
- I couldn't assume readers already had the critical thinking skills that come from writing and debugging a lot of code.
- If I just had them generate a lot of code with AI, they'd risk skipping that thinking entirely.
- I needed a way to help them *build* those skills instead of replacing them.



Bringing It All Together: Using the Sens-AI Framework in the Real World

- AI can be an incredible tool for developers—if we use it thoughtfully.
- The habits in the Sens-AI Framework are what help you stay in control of your work, even when you're moving fast.
- They let you bring the same care and thinking to AI-assisted code that you do to anything you write yourself.

If there's one thing I hope you take away from this course, it's that you can use AI to build better software *and* become a better developer.

Exercise

Doing the Sens-AI exercises from *Head First C#* helps you practice the habits together, so you'll get better at guiding the AI and knowing when to slow down and think.

⚠️ Generating code is uncertain! ⚠️

- **This exercise may take longer than the remaining course time. It's a take-home project for you to practice AI skills.**
- **Always review AI-generated code carefully.** Copilot and other AI tools may sometimes produce code that doesn't compile or work as expected.
- **You're in charge!** Apply the best practices and critical thinking skills we've discussed throughout the course.
- **I'm still here to help.** If you encounter difficulties or want to see the process in action, a great way to get in touch with me is by raising an issue on the GitHub page for the course: https://bit.ly/coding-ai-course
- **Every AI experience is different.** Each interaction with AI can produce unique results. Don't be discouraged if your AI doesn't work the way you want it to. Just remember the Sens-AI habits—and keep refining!

Take-Home Exercise: Practice Using the Sens-AI Framework

Now that you've seen how the Sens-AI habits work together, it's time to apply them on your own.

- Download the **Take-Home Exercise PDF** from https://bit.ly/sens-ai-course.
- This exercise walks you through building a complete Math Quiz game using AI—from generating code to writing unit tests to refactoring.
- You can do it in any language (C#, Java, or Python), using whatever AI tool works best for you.
- Work through each step, applying the best practices and critical thinking skills from the course.
- Remember: AI-generated code is uncertain! Always review carefully, and iterate until you're satisfied.

How to get the most out of the take-home exercise (and why it matters)

After you do each part of the exercise, take notes on *how* you used the Sens-AI framework:

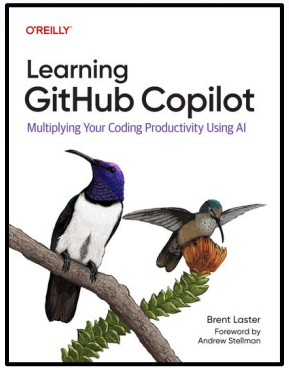
- **Context:** What did you tell the AI about what you need?
- **Research:** How did you learn more about the topic?
- **Problem Framing:** How did you make the prompt clear and specific?
- **Refining:** How did you adjust after seeing the output?
- **Critical Thinking:** Where did you stop and question the code?
- Don't rush it. Taking notes will help you! Your notes help you see *where* you're already applying the habits and *where* you can improve.

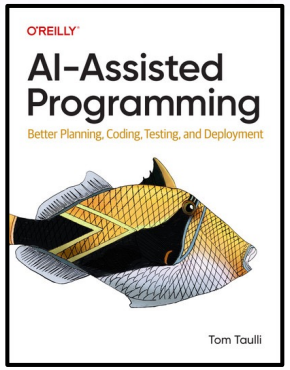
The more you practice recognizing and applying these habits, the more naturally you'll use them in real projects.

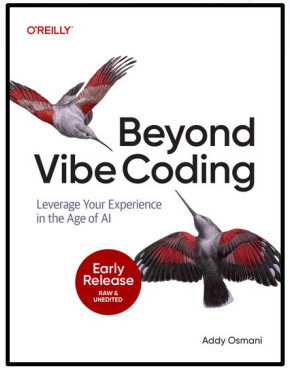
Q&A

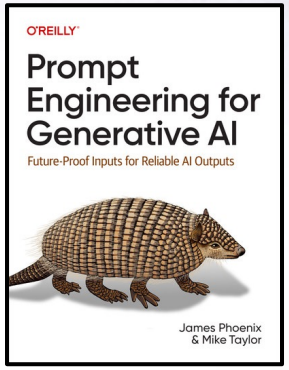
If you have any questions, ask them now. Make sure you enter them in the Q&A widget, because that makes it easier for me to keep track of them.

Additional resources









Want to dive deeper into AI-assisted coding and prompt engineering? Here are some great places to start.

- The material in this course is covered in my O'Reilly report, Critical Thinking Habits for Coding with AI (O'Reilly, 2025) and my O'Reilly Radar series on the Sens-AI Framework
- "Learning GitHub Copilot" by Brent Laster (O'Reilly, 2025) – this book is an amazing resource for learning to use Copilot (I wrote the foreword for it): https://learning.oreilly.com/library/view/learning-github-copilot/9781098164645/
- "AI-Assisted Programming" by Tom Taulli (O'Reilly, 2024) – another fantastic resource, and the chapter on prompt engineering is a phenomenal way to follow up on this course: https://learning.oreilly.com/library/view/ai-assisted-programming/9781098164553/
- "Beyond Vibe Coding" by Addy Osmani (O'Reilly, 2025) https://learning.oreilly.com/library/view/beyond-vibe-coding/9798341634749/
- "Prompt Engineering for Generative AI" by James Phoenix & Mike Taylor (O'Reilly, 2024) https://learning.oreilly.com/library/view/prompt-engineering-for/9781098153427/

linkedin.com/in/andrewstellman/ github.com/andrewstellman linktr.ee/andrewstellman

Thank you!

I hope you enjoyed my course! If you want to learn more, follow me on social media, and check out *Head First C#* and my other books on O'Reilly Learning.

Thanks so much for attending!