# How to tackle unreliability of coding assistants – Sinclair Ibe

One of the trade-offs to the usefulness of coding assistants is their unreliability. The underlying models are quite generic and based on a huge amount of training data, relevant and irrelevant to the task at hand. Also, Large Language Models make things up, they "hallucinate" as it's commonly called. (Side note: There is a lot of discourse about the term "hallucination", about how it is not actually the right psychology metaphor to describe this, but also about using psychology terms in the first place, as it anthropomorphizes the models.)

That unreliability creates two main risks: It can affect the quality of my code negatively, and it can waste my time. Given these risks, quickly and effectively assessing my confidence in the coding assistant's input is crucial.

## How I determine my confidence in the assistant's input

The following are some of the questions that typically go through my head when I try to gauge the reliability and risk of using a suggestion. This applies to "auto complete" suggestions while typing code as well as to answers from the chat.

### Do I have a quick feedback loop?

The quicker I can find out if the answer or the generated information works, the lower the risk that the assistant is wasting my time.

- Can my IDE help me with the feedback loop? Do I have syntax highlighting, compiler or transpiler integration, linting plugins?
- Do I have a test, or a quick way to run the suggested code manually? In one case, I was using the coding assistant chat to help me research how to best display a collapsible JSON data structure in a HTML page. The chat told me about an HTML element I had never heard about, so I was not sure if it existed. But it was easy enough to put it into an HTML file and load that in the browser, to confirm. To give a counterexample, the feedback loop for verifying a piece of infrastructure code I have never heard about is usually a lot longer.

### Do I have a reliable feedback loop?

As well as the speed of the feedback loop for the AI input, I also reflect on the reliability of that feedback loop.

- If I have a test, how confident am I in that test?
- Did I write the test myself, or did I also generate it with the AI assistant?
- If the AI generated the test(s), how confident am I in my ability to review the efficacy of those tests? If the functionality I'm writing is relatively simple and routine, and in a language I'm familiar with, then I'm of course a lot more confident than with a more complex or larger piece of functionality.

- Am I pairing with somebody while using the assistant? They will give additional input and review for the AI input, and increase my confidence.
- If I’m unsure of my test coverage, I can even use the assistant itself to raise my confidence, and ask it for more edge cases to test. This is how I could have found the crucial missing test scenario for the median function I described in a previous memo.

**What is the margin of error?**

I also reflect on what my margin of error is for what I’m doing. The lower the margin for error, the more critical I will be of the AI input.

- When I’m introducing a new pattern, I consider that a larger blast radius for the overall design of the codebase. Other developers on the team will pick up that pattern, and the coding assistant will reproduce that pattern across the team as well, once it is in the code. For example, I have noticed that in CSS, GitHub Copilot suggests flexbox layout to me a lot. Choosing a layouting approach is a big decision though, so I would want to consult with a frontend expert and other members of my team before I use this.
- Anything related to security has of course a low margin of error. For example, I was working on a web application and needed to set a “Content-Security-Policy” header. I didn’t know anything about this particular header, and I first asked Copilot chat. But because of the subject matter, I did not want to rely on its answer, and instead went to a trusted source of security information on the internet.
- How long-lived will this code be? If I’m working on a prototype, or a throwaway piece of code, I’m more likely to use the AI input without much questioning than if I’m working on a production system.

**Do I need very recent information?**

The more recent and the more specific (e.g. to a version of a framework) I need the answer to be, the higher the risk that it is wrong, because the probability is higher that the information I’m looking for is not available or not distinguishable to the AI. For this assessment it’s also good to know if the AI tool at hand has access to more information than just the training data. If I’m using a chat, I want to be aware if it has the ability to take online searches into account, or if it is limited to the training data.

## Give the assistant a timebox

To mitigate the risk of wasting my time, one approach I take is to give it a kind of ultimatum. If the suggestion doesn’t bring me value with little additional effort, I move on. If an input is not helping me quick enough, I always assume the worst about the assistant, rather than giving it the benefit of the doubt and spending 20 more minutes on making it work.

The example that comes to mind is when I was using an AI chat to help me generate a mermaid.js class diagram. I’m not very familiar with the mermaid.js syntax, and I kept trying to make the suggestion work, and thought I had maybe included it in my markdown file in the wrong way. Turns out, the syntax was totally wrong, which I found out when I finally went to the online documentation after 10 minutes or so.

## Make up a persona for the assistant

When preparing this memo, I started wondering if making up a persona for the assistant could help with how to use it responsibly, and with as little waste of time as possible. Maybe anthropomorphizing the AI could actually help in this case?

Thinking about the types of unreliability, I'd imagine the AI persona with these traits:

- eager to help
- stubborn
- very well-read, but inexperienced (for Dungeons and Dragons fans: high intelligence, low wisdom)
- won't admit when it doesn't "know" something

I tried a few prompts with an image generator, asking it for variations of eager beavers and stubborn donkeys. Here's the one I liked the best ("eager stubborn donkey happy books computer; cartoon, vector based, flat areas of color" in Midjourney):