Context Is All You Need

If you're at all familiar with AI, you'll know that the title is a pun. You'll also know that the original paper, *Attention is All You Need,* had a seismic impact on machine learning. I chose this title because I think some of the ideas in this paper could have an equally seismic effect, albeit in a much smaller niche.

In October Google's CEO, Sundar Pichai, said that 25% of the company's code is now written by AI.[1] That number will continue to grow at a rapid rate. For the first time in my life, I too, am writing code. And now, in the last few weeks, I'm writing software.

Writing code and writing software are not the same. Writing code is the act of producing correct and logical syntax. Writing software is producing that code while maintaining a working knowledge of the other code in a system. They both require writing code, but building software requires context. This matters because it motivates a different problem. AI is good at coding, but not nearly as good at software engineering.

Entry-level SWE positions are becoming increasingly vulnerable. These positions entail more coding than software engineering. This is because when you start at a company, you don't yet have a knowledge of the codebase. So, by virtue of circumstance, the entry-level SWE writes code. AI is very good at writing code, so this SWE is not needed. High-level SWEs also have to write code. Their jobs are comparatively much safer, yet AI is still better than they are at writing code. So the difference in the value added by entry-level and high-level SWEs must be something other than writing code. Much of the difference is codebase knowledge. Context.

One problem with using AI to write software is limited context windows. I know this because I write software with AI. A context window is the amount of information a model can recall before it begins to forget it. It's memory. In isolation, AI writes pieces of code flawlessly. It doesn't work as well when you start writing a bunch of code for the same system. If you do this, the AI will start to forget some of the code it already wrote. This is bad because if you want to connect one chunk of code to another, it won't remember the other one. There are ways around this. One is simply reminding the model by showing it some code you deem relevant. This sort of works, but it forgets quickly, and you usually leave out some other piece context that would be important. It's especially bad if you're building a larger system, where one change likely cascades into many. So it's a problem. Many of the issues I have programming with AI would be solved if it had a working memory of my codebase. Better context.

How might one fix this? One way is to wait. Context windows are getting much better, and at some point this won't be a problem anymore. But waiting is boring, and usually doesn't make you a ton of money. I've thought of a different way. For now we can't make context windows better, but we can change the way we think about them. I will now torture you with a metaphor.

I've played soccer since I was very young. To be a good soccer player, you need to do a lot of things. You need to shoot. You need to pass. You need to run. Defend. Etc. If you don't learn to do these things, you probably aren't going to be a very good soccer player.
Now let's say that when I was little I was obsessed with the bicycle kick. The bicycle kick is a specific move where you throw yourself in the air, and acrobatically kick the ball mid movement. It's effectively useless in an actual game of soccer, so nobody practices them. But again, I was obsessed, so for 10 years I only did bicycle kicks. No shooting. No passing. Nothing but bicycle kicks.

At the end of the 10 years I was very good at bicycle kicks. I was also very bad at everything else. So, it turns out that if you want to be a good soccer player, you shouldn't just practice bicycle kicks.

It's the same way when we train AI models. If you want to train a model to be good at something, you can't only train it on a specific subset of that thing. Take a model trained to classify images of birds. If I train the model on only images of Blue Jays, it's going to classify everything as a Blue Jay. It's hard to imagine a world in which this is useful.

We say this is bad because the model fails to generalize beyond its training data. This lack of generalization is called overfitting, and it's caused by training on too narrow a dataset. We don't want overfitting, so we say it's bad.

Let's return now to our soccer metaphor. All of a sudden I've entered to compete at the bicycle kick world championship. Suddenly, that overtraining is exactly what I needed. I might be terrible at soccer, but I'm really good at bicycle kicks, so I'm going to beat all the soccer players I'm up against. So now overfitting is useful, because we've aligned it with the goal. It's basically just specialization.

It's the same with software. Companies aren't usually building *all software*, they're just building specific software for specific things. Yet, the AI they use to help them write that code isn't specific. It's been trained on a ton of code all over the place. So, when a senior developer is coding with AI, why do they need it to understand *all* software? They don't. It just happens to, because that's how it's been trained.

Here's my solution: intentional overfitting. Overfitting is almost always a bad thing. I've just shown you why it might not be. If a developer is only using AI on one codebase, then all it needs is a working knowledge of *that* codebase. So, given a codebase or a repository, one could theoretically fine-tune a model on just *that* code. The model would only work well with code in that codebase, but for a software company, that's exactly enough.

Fine-tuning an open-source model exclusively on that code would make it extremely fluent in that system's structure, naming conventions, and dependencies. It would perform poorly on unrelated code, but for the company in question, that is not a drawback. It's an advantage.

The potential value for this is highest for large organizations. A small startup with a few thousand lines of code can fit its entire codebase into a modern context window, so general-purpose tools like Cursor work fine. But a company with millions of lines of code has no way to fit it all into a single prompt, and while retrieval-augmented approaches help, they will always miss some important pieces. In such cases, a model that has been deliberately overfit to the codebase could outperform even the largest context windows. Overtraining, in this sense, turns the generalist model into a domain expert for one particular environment. It allows the AI to carry not just the current working file in its "mind," but also the deep understanding that comes from having learned the entire codebase over time. That kind of specialization is exactly what large organizations rely on from their most senior engineers.

Context is not a luxury in software development. It is the foundation. By intentionally overfitting models to massive, specific codebases, we could give AI the same contextual advantage that human experts develop over years. It is an unorthodox approach, but I think it could work.

1. https://fortune.com/2024/10/30/googles-code-ai-sundar-pichai/