VIEWPOINT

The End of Programming

By Matt Welsh
Communications of the ACM, January 2023, Vol. 66 No. 1, Pages 34-35
10.1145/3570220
Comments (10)

MORE NEWS & OPINIONS

Apple Says Your iPhone's Usage Data is Anonymous, but Tests Say That's Not True
Gizmodo

Data Platforms and Network Effects
Michael A. Cusumano

Data Loss Prevention vs. Data-Centric Audit and Protection: What Is the Difference?
Alex Vakulov

Credit: Ursa Major

The End of Programming
CACM
03:44

I came of age in the 1980s, programming personal computers such as the Commodore VIC-20 and Apple][e at home. Going on to study computer science (CS) in college and ultimately getting a Ph.D. at Berkeley, the bulk of my professional training was rooted in what I will call "classical" CS: programming, algorithms, data structures, systems, programming languages. In Classical Computer Science, the ultimate goal is to reduce an idea to a program written by a human—source code in a language like Java or C++ or Python. Every idea in Classical CS—no matter how complex or sophisticated, from a database join algorithm to the mind-bogglingly obtuse Paxos consensus protocol—can be expressed as a human-readable, human-comprehendible program.

When I was in college in the early 1990s, we were still in the depths of the AI Winter, and AI as a field was likewise dominated by classical algorithms. My first research job at Cornell University was working with Dan Huttenlocher, a leader in the field of computer vision (and now Dean of the MIT Schwarzman College of Computing). In Huttenlocher's Ph.D.-level computer vision course in 1995 or so, we never once discussed anything resembling deep learning or neural networks—it was all classical algorithms like Canny edge detection, optical flow, and Hausdorff distances. Deep learning was in its infancy, not yet considered mainstream AI, let alone mainstream CS.

Of course, this was 30 years ago, and a lot has changed since then, but one thing that has not really changed is that CS is taught as a discipline with data structures, algorithms, and programming at its core. I am going to be amazed if in 30 years, or even 10 years, we are still approaching CS in this way. Indeed, I think CS as a field is in for a pretty major upheaval few of us are really prepared for.

Programming will be obsolete. I believe the conventional idea of "writing a program" is headed for extinction, and indeed, for all but very specialized applications, most software, as we know it, will be replaced by AI systems that are *trained* rather than *programmed*. In situations where one needs a "simple" program (after all, not everything should require a model of hundreds of billions of parameters running on a cluster of GPUs), those programs will, themselves, be generated by an AI rather than coded by hand.

I do not think this idea is crazy. No doubt the earliest pioneers of computer science, emerging from the (relatively) primitive cave of electrical engineering, stridently believed that all future computer scientists would need to command a deep understanding of semiconductors, binary arithmetic, and microprocessor design to understand software. Fast-forward to today, and I am willing to bet good money that 99% of people who are writing software have almost no clue how a CPU actually works, let alone the physics underlying transistor design. By extension, I believe the computer scientists of the future will be so far removed from the classic definitions of "software" that they would be hard-pressed to reverse a linked list or implement Quicksort. (I am not sure I remember how to implement Quicksort myself.)

AI coding assistants such as CoPilot are only scratching the surface of what I am describing. It seems totally obvious to me that *of course* all programs in the future will ultimately be written by AIs, with humans relegated to, at best, a supervisory role. Anyone who doubts this prediction need only look at the very rapid progress being made in other aspects of AI content generation, such as image generation. The difference in quality and complexity between DALL-E v1 and DALL-E v2—announced only 15 months later—is staggering. If I have learned anything over the last few years working in AI, it is that it is *very* easy to underestimate the power of increasingly large AI models. Things that seemed like science fiction only a few months ago are rapidly becoming reality.

So I am not just talking about things like Github's CoPilot replacing programmers.[1] I am talking about *replacing the entire concept of writing programs with training models*. In the future, CS students are not going to need to learn such mundane skills as how to add a node to a binary tree or code in C++. That kind of education will be antiquated, like teaching engineering students how to use a slide rule.

The engineers of the future will, in a few keystrokes, fire up an instance of a four-quintillion-parameter model that already encodes the full extent of human knowledge (and then some), ready to be given any task required of the machine. The bulk of the intellectual work of getting the machine to do what one wants will be about coming up with the right examples, the right training data, and the right ways to evaluate the training process. Suitably powerful models capable of generalizing via few-shot learning will require only a few good examples of the task to be performed. Massive, human-curated datasets will no longer be necessary in most cases, and most people "training" an AI model will not be running gradient descent loops in PyTorch, or anything like it. They will be teaching by example, and the machine will do the rest.

In this new computer science—if we even call it computer science at all—the machines will be so powerful and already know how to do so many things that the field will look like less of an engineering endeavor and more of an an educational one; that is, *how to best educate the machine*, not unlike the science of how to best educate children in school. Unlike (human) children, though, these AI systems will be flying our airplanes, running our power grids, and possibly even governing entire countries. I would argue that the vast majority of Classical CS becomes irrelevant when our focus turns to teaching intelligent machines rather than directly programming them. Programming, in the conventional sense, will in fact be dead.

I think CS as a field is in for a pretty major upheaval few of us are really prepared for.

How does all of this change how we think about the field of computer science? The new atomic unit of computation becomes not a processor, memory, and I/O system implementing a von Neumann machine, but rather a massive, pre-trained, highly adaptive AI model. This is a seismic shift in the way we think about computation—not as a predictable, static process, governed by instruction sets, type systems, and notions of decidability. AI-based computation has long since crossed the Rubicon of being amenable to static analysis and formal proof. We are rapidly moving toward a world where the fundamental building blocks of computation are temperamental, mysterious, adaptive agents.

This shift is underscored by the fact that *nobody actually understands how large AI models work*. People are publishing research papers[3,4,5] actually *discovering new behaviors* of existing large models, even though these systems have been "engineered" by humans. Large AI models are capable of doing things that they have not been explicitly trained to do, which should scare the living daylights out of Nick Bostrom[2] and anyone else worried (rightfully) about an superintelligent AI running amok. We currently have no way, apart from empirical study, to determine the limits of current AI systems. As for future AI models that are orders of magnitude larger and more complex—good luck!

The shift in focus from programs to models should be obvious to anyone who has read any modern machine learning papers. These papers barely mention the code or systems underlying their innovations; the building blocks of AI systems are much higher-level abstractions like attention layers, tokenizers, and datasets. A time traveler from even 20 years ago would have a hard time making sense of the *three sentences* in the (75-page!) GPT-3 paper[3] describing the actual software built for the model: "We use the same model and architecture as GPT-2, including the modified initialization, pre-normalization, and reversible tokenization described therein, with the exception that we use alternating dense and locally banded sparse attention patterns in the layers of the transformer, similar to the Sparse Transformer. To study the dependence of ML performance on model size, we train eight different sizes of model, ranging over three orders of magnitude from 125 million parameters to 175 billion parameters, with the last being the model we call GPT-3. Previous work suggests that with enough training data, scaling of validation loss should be approximately a smooth power law as a function of size; training models of many different sizes allows us to test this hypothesis both for validation loss and for downstream language tasks."

This shift in the underlying definition of computing presents a huge opportunity, and plenty of huge risks. Yet I think it is time to accept that this is a very likely future, and evolve our thinking accordingly, rather than just sit here waiting for the meteor to hit.

Figure. Watch the author discuss this work in the exclusive *Communications* video. https://cacm.acm.org/videos/end-of-programming

Back to Top

References

1. Berger, E. Coping with copilot. SIGPLAN *PL Perspectives* Blog, 2022; https://bit.ly/3XbJv5J

2. Bostrom, N. *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press, 2014.

3. Brown, T. et al. Language models are few-shot learners. 2020; https://bit.ly/3Eh1DT5

4. Kojima, T. et al. Large language models are zero-shot reasoners. 2022; https://bit.ly/3Ohmlqo

5. Nye, M. et al. Show your work: Scratchpads for intermediate computation with language models. 2021; https://bit.ly/3TLnfMY

Back to Top

Author

Matt Welsh (mdw@mdw.la) is the CEO and co-founder of Fixie.ai, a recently founded startup developing AI capabilities to support software development teams. He was previously a professor of computer science at Harvard University, a director of engineering at Google, an engineering lead at Apple, and the SVP of Engineering at OctoML. He received his Ph.D. from UC Berkeley back in the days when AI was still not playing chess very well.

Comments

Ken Kahn

December 22, 2022 06:17

Agree. But even "The bulk of the intellectual work of getting the machine to do what one wants will be about coming up with the right examples, the right training data, and the right ways to evaluate the training process." might be more than what will be needed. Why wouldn't future systems largely automate this?