

Andrej Karpathy: Software Is Changing (Again)

Software is changing again, and now it's programmed in English

“Remarkably, we're now programming computers in English.” That is Andrej Karpathy, quoting his own pinned tweet. The claim underneath it: software barely moved for 70 years, and then changed twice in the space of a few.

Karpathy was introduced to the room as the former director of AI at Tesla, and he called the talk “software in the era of AI.” He was told many in the room were students, “bachelors, masters, PhD and so on,” people about to enter the industry. He has given this talk before, and says so: “I actually gave this talk already. But the problem is that software keeps changing. So I actually have a lot of material to create new talks.” His argument is that the students are arriving at an unusually good moment. “It's actually like an extremely unique and very interesting time to enter the industry right now,” he says, because “there's just a huge amount of work to do, a huge amount of software to write and rewrite.”

Software 1.0, 2.0, and now 3.0

Karpathy opens with a map. “This is a really cool tool called map of GitHub,” he says, showing the sprawl of everything people have written for computers to run. That is software 1.0: code you write to instruct a machine.

A few years ago he noticed a second category and named it software 2.0. “Software 2.0 are basically neural networks, and in particular the weights of a neural network,” he says. “You're not writing this code directly. You are more kind of like tuning the datasets, and then you're running an optimizer to create the parameters of this neural net.” The label was needed, he says, because at the time neural nets “were kind of seen as just a different kind of classifier, like a decision tree or something like that.”

That category now has its own infrastructure. “Hugging Face is basically equivalent of GitHub in software 2.0,” he says, alongside model atlas, which visualizes the same space. He points at a giant circle in the middle of it: the parameters of Flux, the image generator. “Anytime someone tunes on top of a Flux model, you basically create a git commit in this space, and you create a different kind of an image generator.”

For the older kind, his example is AlexNet, the image recognizer: neural networks as “kind of like fixed function computers,” mapping images to categories. Then something changed underneath that picture. “What’s changed, and I think is a quite fundamental change, is that neural networks became programmable with large language models.”

That earns a new number. “It’s a new kind of a computer, and so in my mind it’s worth giving it a new designation of software 3.0. And basically your prompts are now programs that program the LLM.”

The example he uses is sentiment classification. You can write Python for it. You can train a neural net for it. Or you can write a few-shot prompt and change the program by changing the English. He points out that this is already visible in public code. “You’ve seen a lot of GitHub code is not just like code anymore. There’s a bunch of English interspersed with code,” he says, “and so I think there’s a growing category of new kind of code.” What he keeps returning to is the language itself. “Not only is it a new programming paradigm, it’s also remarkable to me that it’s in our native language of English.”

His advice to the students follows directly from having three paradigms instead of one. “If you’re entering the industry it’s a very good idea to be fluent in all of them, because they all have slight pros and cons,” he says. “Are you going to train a neural net? Are you going to just prompt an LLM? Should this be a piece of code that’s explicit?” You answer that per feature, which is why he wants all three in the toolkit.

The C++ that disappeared from Tesla’s autopilot

Karpathy has watched one of these transitions happen inside a product he worked on. At Tesla, the autopilot stack took input at the bottom and produced steering and acceleration at the top. In between sat a lot of C++, with some neural networks doing image recognition.

Then the proportions moved. “As we made the autopilot better, basically the neural network grew in capability and size, and in addition to that all the C++ code was being deleted,” he says. The network took over work engineers had written by hand. The stitching together of information across the different cameras and across time, for instance, moved into a neural net, “and we were able to delete a lot of code, and so the software 2.0 stack quite literally ate through the software stack of the autopilot.”

His read on the present moment is that the same thing is happening one layer up. “We have a new kind of software and it’s eating through the stack.”

LLMs have the properties of utilities

Karpathy works out what kind of object an LLM is by running analogies and seeing which ones hold. The first comes from Andrew Ng, who he points out is speaking right after him. “AI is the new electricity,” Karpathy quotes, and he thinks it captures something real. The labs, OpenAI and Google and Anthropic, spend capex to train models, which is the equivalent of building out a grid, then opex to serve that intelligence over APIs. The labs meter access and price it per million tokens. What customers want from it sounds like what they want from a utility: “we demand low latency, high uptime, consistent quality.”

The analogy extends to the switching hardware. “In electricity, you would have a transfer switch, so you can transfer your electricity source from grid and solar or battery or generator. In LLMs, we have maybe OpenRouter.”

Software differs in one way. “Because the LLMs are software, they don’t compete for physical space. So it’s okay to have basically like six electricity providers and you can switch between them.”

The analogy also holds when things break. In the days before the talk, a lot of the LLMs went down and people were “kind of like stuck and unable to work. And I think it’s kind of fascinating to me that when the state-of-the-art LLMs go down, it’s actually kind of like an intelligence brownout in the world,” he says. “It’s kind of like when the voltage is unreliable in the grid, and the planet just gets dumber the more reliance we have on these models.”

The fab analogy, and where it stops working

Utilities do not require the kind of money that LLMs require, so Karpathy tries a second comparison: semiconductor fabs.

“The capex required for building an LLM is actually quite large,” he says. “It’s not just like building some power station.” Alongside the money there is a deep tech tree, and research and development secrets that concentrate inside a small number of labs.

Then he qualifies it. “The analogy muddies a little bit also because, as I mentioned, this is software, and software is a bit less defensible because it is so malleable.”

He maps the pieces across anyway. A 4 nanometer process node is something like a cluster with a certain max flops. “When you’re using Nvidia GPUs and you’re only doing the software and you’re not doing the hardware, that’s kind of like the fabless model. But if you’re actually also building your own hardware and you’re training on TPUs, if you’re Google, that’s kind of like the Intel model where you own your fab.” He is not throwing the comparison out: “I think there’s some analogies here that make sense.”

Why the operating system is the analogy that fits

Neither of the first two analogies is where he lands. “This is not just electricity or water. It’s not something that comes out of the tap as a commodity,” he says. “These are now increasingly complex software ecosystems.”

The market has arranged itself along the same lines. There are a few closed source providers, the Windows and macOS of the arrangement, and an open source alternative that could grow into Linux. “Maybe the Llama ecosystem is currently a close approximation to something that may grow into something like Linux.”

He is careful about how early all of this is. “It’s still very early because these are just simple LLMs, but we’re starting to see that these are going to get a lot more complicated. It’s not just about the LLM itself. It’s about all the tool use and the multimodalities and how all of that works.”

He also thinks the internals line up. “The LLM is a new kind of a computer. It’s kind of like the CPU equivalent. The context windows are kind of like the memory, and then the LLM is orchestrating memory and compute for problem solving.”

The app layer behaves the same way too. “If you want to download an app, say I go to VS Code and I go to download, you can download VS Code and you can run it on Windows, Linux or Mac, in the same way as you can take an LLM app like Cursor and you can run it on GPT or Claude or Gemini series. It’s just a drop down.”

We are in the 1960s, and nobody has invented the GUI yet

If LLMs are operating systems, Karpathy wants to know which decade of operating systems we are living in. His answer is the 1960s.

Compute for this new computer is expensive, which forces it into the cloud. “We’re all just sort of thin clients that interact with it over the network, and none of us have full utilization of these computers, and therefore it makes sense to use time sharing, where we’re all just a dimension of the batch.” That is what computers looked like then, he says: operating systems in the cloud, everything streamed around, batching.

Which means the personal computing moment is still ahead. “The personal computing revolution hasn’t happened yet, because it’s just not economical.” Some people are pushing at it anyway, and one machine turns out to suit the workload. “Mac minis, for example, are a very good fit for some of the LLMs, because if you’re doing batch one inference, this is all super memory bound. So this actually works.”

He leaves the question open for the room. “Maybe some of you get to invent what this is or how it works.”

The interface is the other thing missing. “Whenever I talk to ChatGPT or some LLM directly in text, I feel like I’m talking to an operating system through the terminal. It’s just text. It’s direct access to the operating system. And I think a GUI hasn’t yet really been invented in a general way. Should ChatGPT have a GUI different than just text bubbles?” Individual apps have their own. “There’s no GUI across all the tasks, if that makes sense.”

LLMs reached consumers before they reached governments

One property of LLMs breaks the pattern Karpathy expects from new technology, and he wrote about it separately.

“LLMs flip the direction of technology diffusion that is usually present in technology,” he says. Electricity, cryptography, computing, flight, the internet, GPS: typically the first users were governments and corporations, because the technology was new and expensive, and consumers came later.

This one arrived the other way around. “Maybe with early computers, it was all about ballistics and military use, but with LLMs, it’s all about how do you boil an egg or something like that. This is certainly like a lot of my use. And so it’s really fascinating to me that we have a new magical computer and it’s like helping me boil an egg. It’s not helping the government do something really crazy like some military ballistics.” Institutions, he says, are the laggards here: “corporations and governments are lagging behind the adoption of all of us.”

He closes the first half by pulling the analogies together. LLMs are complicated operating systems, circa the 1960s of computing, “and we’re redoing computing all over again,” available on a time sharing model and distributed like a utility. What has no precedent is who holds them. “They’re not in the hands of a few governments and corporations. They’re in the hands of all of us, because we all have a computer and it’s all just software, and ChatGPT was beamed down to our computers, like billions of people, instantly and overnight. And this is insane.” Then he turns it back to the room: “now it is our time to enter the industry and program these computers. This is crazy.”

People spirits, and their cognitive deficits

Before programming these computers, Karpathy wants the room to think about what is on the other side of the prompt.

“The way I like to think about LLMs is that they’re kind of like people spirits,” he says. “They are stochastic simulations of people.” The simulator is an autoregressive transformer that works through tokens, “chunk chunk chunk chunk chunk,” with roughly equal compute spent on each one. Because it was fit to the text humans wrote, “it’s got this emergent psychology that is humanlike.”

The superpower is memory. LLMs “have encyclopedic knowledge and memory,” he says, far past what any individual could hold, “because they read so many things.” He reaches for Rain Man, “which I actually really recommend people watch. It’s an amazing movie. I love this movie.” Dustin Hoffman plays an autistic savant with almost perfect memory, who can read a phone book and remember all the names and phone numbers. “LLMs are kind of like very similar. They can remember SHA hashes and lots of different kinds of things very, very easily.”

Then the deficits. They hallucinate, and “don’t have a very good internal model of self-knowledge, not sufficient at least,” though “this has gotten better but not perfect.” They have what he calls jagged intelligence: “they’re going to be superhuman in some problem solving domains, and then they’re going to make mistakes that basically no human will make.” Two get quoted more than the rest. “They will insist that 9.11 is greater than 9.9, or that there are two R’s in strawberry.”

The deficit he dwells on longest is memory of a different kind. A new colleague joins your company, learns the place over time, goes home and sleeps, consolidates what they learned, and turns into an expert. “LLMs don’t natively do this, and this is not something that has really been solved in the R&D of LLMs.” He calls it anterograde amnesia. What you get instead is a working memory you have to fill yourself. “Context windows are really kind of like working memory, and you have to program the working memory quite directly, because they don’t just get smarter by default.”

For anyone who wants the feeling of it, he prescribes two films. “I recommend people watch these two movies, Memento and 50 First Dates. In both of these movies, the protagonists, their weights are fixed and their context windows get wiped every single morning, and it’s really problematic to go to work or have relationships when this happens.”

The last item on the list is security. “LLMs are quite gullible. They are susceptible to prompt injection risks. They might leak your data.”

Which leaves you holding both at once. “You have to simultaneously think through this superhuman thing that has a bunch of cognitive deficits and issues,” he says. “How do we program them and how do we work around their deficits and enjoy their superhuman powers?”

Partial autonomy apps, and the slider that runs them

Karpathy flags what follows as “not a comprehensive list, just some of the things that I thought were interesting for this talk.” The first of the opportunities he picks out is what he calls partial autonomy apps, and his example is coding.

“You can certainly go to ChatGPT directly and you can start copy pasting code around, and copy pasting bug reports and stuff around, and getting code and copy pasting everything around,” he says. “Why would you do that? Why would you go directly to the operating system?” The better answer is a dedicated app, and the one he uses is Cursor. He wants the room to look at its shape. “We have a traditional interface that allows a human to go in and do all the work manually just as before. But in addition to that, we now have this LLM integration that allows us to go in bigger chunks.”

He pulls out four properties that he thinks generalize to every app of this kind.

The app handles context: “the LLMs basically do a ton of the context management.” It also orchestrates models rather than calling just one. In Cursor, “there’s under the hood embedding models for all your files, the actual chat models, models that apply diffs to the code, and this is all orchestrated for you.”

The interface is purpose-built, and he thinks it is undervalued. “You don’t just want to talk to the operating system directly in text. Text is very hard to read, interpret, understand.” A diff shown in red and green is legible in a way a paragraph is not, and accepting it should be a keystroke. “It’s much easier to just do command Y to accept or command N to reject. I shouldn’t have to type it in text, right? So a GUI allows a human to audit the work of these fallible systems and to go faster.”

Last, the dial. “There’s what I call the autonomy slider.” In Cursor that runs from tab completion, where you are mostly in charge, to command K for a selected chunk, to command L for a whole file, to command I, “which just let it rip, do whatever you want in the entire repo,” he says. “Depending on the complexity of the task at hand, you can tune the amount of autonomy that you’re willing to give up.”

Perplexity has the same four properties. It packages information, orchestrates several models, cites sources you can go and inspect, and offers its own slider: “you can either just do a quick search, or you can do research, or you can do deep research and come back 10 minutes later.”

Karpathy turns this into a question for everyone in the room who ships a product. “How are you going to make your products and services partially autonomous? Can an LLM see everything that a human can see? Can an LLM act in all the ways that a human could act? And can humans supervise and stay in the loop of this activity?” He leaves it open. “What

does a diff look like in Photoshop or something like that?” Existing software is full of controls built for hands and eyes. “All of this has to change and become accessible to LLMs.”

Keeping the AI on the leash

Underneath the product advice is a loop that Karpathy thinks people underrate. The AI generates, the human verifies. “It is in our interest to make this loop go as fast as possible.”

There are two ways to speed it up, and the first is to make verification cheap. This is what interfaces are for. “A GUI utilizes your computer vision GPU in all of our head. Reading text is effortful and it’s not fun, but looking at stuff is fun, and it’s kind of like a highway to your brain.”

The second is to stop the model from producing more than you can check. “We have to keep the AI on the leash,” he says. “I think a lot of people are getting way over excited with AI agents.” He has run into the limit himself. “It’s not useful to me to get a diff of 10,000 lines of code to my repo. I’m still the bottleneck. Even though that 10,000 lines come out instantly, I have to make sure that this thing is not introducing bugs, and that it’s doing the correct thing, and that there’s no security issues.”

He draws a line between two ways of working, one of which he named himself. “If I’m just vibe coding, everything is nice and great. But if I’m actually trying to get work done, it’s not so great to have an overreactive agent doing all this kind of stuff.”

He is not pretending he has this solved. “So this slide is not very good, I’m sorry,” he says, “but I guess I’m trying to develop, like many of you, some ways of utilizing these agents in my coding workflow.” What he has landed on is small. “In my own work, I’m always scared to get way too big diffs. I always go in small incremental chunks. I want to make sure that everything is good. I want to spin this loop very, very fast,” working on “small chunks of single concrete thing.”

A blog post he read recently made the same point from the prompting side. “If your prompt is vague, then the AI might not do exactly what you wanted, and in that case verification will fail. You’re going to ask for something else. If a verification fails, then you’re going to start spinning.” Spending longer on a concrete prompt “increases the probability of successful verification and you can move forward.”

Education is what Karpathy says he is currently thinking about, and it shows what the leash looks like inside a product. The naive version does not work. “I don’t think it just works to go to ChatGPT and be like, hey, teach me physics. I don’t think this works, because the AI gets lost in the woods.”

The design he describes splits the problem in two. “For me, this is actually two separate apps. For example, there’s an app for a teacher that creates courses, and then there’s an app that takes courses and serves them to students.” What that buys him is something to inspect between the model and the student. “In both cases, we now have this intermediate artifact of a course that is auditable, and we can make sure it’s good. We can make sure it’s consistent.” The syllabus becomes the constraint: “the AI is kept on the leash with respect to a certain syllabus, a certain progression of projects.”

Twelve years of driving, and the decade of agents

Karpathy has done partial autonomy before. He spent five years on it at Tesla, and the autopilot has the same anatomy he has been describing: a GUI in the instrument panel “showing me what the neural network sees,” and a slider that moved further right over his tenure.

“The first time I drove a self-driving vehicle was in 2013,” Karpathy recalls. A friend at Waymo offered him a ride around Palo Alto, and he took a photo on Google Glass. “Many of you are so young that you might not even know what that is,” he notes. “But yeah, this was like all the rage at the time.” The drive lasted about 30 minutes, over highways and streets. “This drive was perfect. There was zero interventions. And this was 2013, which is now 12 years ago.”

At the time, that settled it. “I felt like, wow, self-driving is imminent, because this just worked. This is incredible.”

Then he counts forward from that demo. “Here we are 12 years later and we are still working on autonomy. We are still working on driving agents, and even now we haven’t actually really solved the problem.” The cars on the road look further along than they are. “You may see Waymos going around and they look driverless, but there’s still a lot of teleoperation and a lot of human in the loop of a lot of this driving.”

“We still haven’t even declared success, but I think it’s definitely going to succeed at this point. It just took a long time.” Then the transfer: “Software is really tricky, I think, in the same way that driving is tricky.” So he is wary when people put a year on it. “When I see things like, oh, 2025 is the year of agents, I get very concerned, and I kind of feel like, you know, this is the decade of agents.”

He puts it plainly. “We need humans in the loop. We need to do this carefully. This is software. Let’s be serious here.”

What the Iron Man suit is for

The image Karpathy keeps returning to is the Iron Man suit, because it works two ways at once. “What I love about the Iron Man suit is that it’s both an augmentation, and Tony Stark can drive it, and it’s also an agent,” he says. In some of the films the suit flies around on its own and goes looking for Tony.

That doubleness is the autonomy slider again. “We can build augmentations or we can build agents, and we kind of want to do a bit of both.” But at this stage, working with fallible LLMs, he picks a side. “It’s less Iron Man robots and more Iron Man suits that you want to build. It’s less like building flashy demos of autonomous agents and more building partial autonomy products.” He is not ruling out full automation. “We are not losing sight of the fact that it is in principle possible to automate this work.”

Everyone is a programmer now

The other thing he calls “completely unprecedented” is who gets to write software at all, and it follows from the programming language being English.

“Suddenly everyone is a programmer, because everyone speaks natural language like English,” he says. “It used to be the case that you need to spend five to 10 years studying something to be able to do something in software. This is not the case anymore.”

He is responsible for what people now call this. The tweet that coined vibe coding is his. “I’ve been on Twitter for like 15 years or something like that at this point, and I still have no clue which tweet will become viral and which tweet fizzles and no one cares,” he says. He expected this one to sink. “It was just like a shower of thoughts. But this became like a

total meme and I really just can't tell." His explanation is that the phrase was already needed: "it gave a name to something that everyone was feeling but couldn't quite say in words. So now there's a Wikipedia page and everything." The room applauds. "Yeah, this is like a major contribution now or something like that."

He shows a clip Tom Wolf of Hugging Face had shared, of children vibe coding. "I find that this is such a wholesome video. Like, I love this video. How can you look at this video and feel bad about the future? The future is great." He expects it to be an on-ramp rather than a dead end. "I think this will end up being like a gateway drug to software development. I'm not a doomer about the future of the generation."

He tried it himself, "because it's so fun." Vibe coding suits a particular kind of job: "when you want to build something super duper custom that doesn't appear to exist and you just want to wing it, because it's a Saturday." He built an iOS app in a language he does not know. "I can't actually program in Swift, but I was really shocked that I was able to build like a super basic app. And I'm not going to explain it. It's really dumb." A day of work, and it was running on his phone later that day. "I was like, wow, this is amazing. I didn't have to read through Swift for like five days or something like that to get started."

The demo took a few hours, making it real took a week

The second thing he vibe coded is live. "I show up at a restaurant, I read through the menu, and I have no idea what any of the things are. And I need pictures." So he built MenuGen, at menugen.app: photograph a menu, get images of the dishes back.

"Everyone gets \$5 in credits for free when you sign up," he says, "and therefore this is a major cost center in my life. So this is a negative revenue app for me right now. I've lost a huge amount of money on MenuGen."

The part he found instructive was which half of the project was hard. "The code was actually the easy part of vibe coding MenuGen. Most of it actually was when I tried to make it real." Authentication, payments, a domain name, deployment. "This was really hard, and all of this was not code. All of this DevOps stuff was me in the browser clicking stuff, and this was extremely slow and took another week."

He had the demo running on his laptop in a few hours. The week that followed went to a different kind of work, and one screen in particular set him off: the Clerk instructions for adding Google login. “It’s telling me go to this URL, click on this dropdown, choose this, go to this, and click on that. It’s telling me what to do. Like a computer is telling me the actions I should be taking. Like, you do it. Why am I doing this? What the hell? I had to follow all these instructions. This was crazy.”

llms.txt, markdown docs, and replacing click with curl

Which brings Karpathy to his question: “can we just build for agents? I don’t want to do this work. Can agents do this?”

He frames it as a new species of user. “There’s a new category of consumer and manipulator of digital information. It used to be just humans through GUIs or computers through APIs. And now we have a completely new thing.” Agents sit between the two. “They’re computers, but they are humanlike, kind of. They’re people spirits. There’s people spirits on the internet, and they need to interact with our software infrastructure.”

He starts with talking to them directly. We already have robots.txt to advise crawlers, so he wants the equivalent for models: “maybe an llms.txt file, which is just a simple markdown that’s telling LLMs what this domain is about.” The alternative is worse. Making a model read your HTML and work it out “is very error prone and difficult, and will screw it up and it’s not going to work.”

Docs are the bigger target, and almost all of them were written for people. “You will see things like lists and bold and pictures, and this is not directly accessible by an LLM.” Some companies have started converting. “Vercel and Stripe as an example are early movers here,” offering their docs in markdown, “super easy for LLMs to understand.”

He wanted to make animations with Manim, the library behind 3Blue1Brown’s videos. “I love this library,” he says. He did not want to read its documentation. “So I copy pasted the whole thing to an LLM and I described what I wanted, and it just worked out of the box. The LLM just vibe coded me an animation exactly what I wanted, and I was like, wow, this is amazing.”

Reformatting only gets you so far. “We actually have to change the docs, because anytime your docs say click, this is bad. An LLM will not be able to natively take this action right now.” So the fix has to reach the content. “Vercel, for example, is replacing every occurrence of click with an equivalent curl command that your LLM agent could take on your behalf.”

He puts Anthropic’s Model Context Protocol in the same category, “another way, it’s a protocol of speaking directly to agents as this new consumer and manipulator of digital information.”

A GitHub page is a human interface, so pasting one at a model gets you nowhere. Swap github for gitingest in the address “and this will actually concatenate all the files into a single giant text, and it will create a directory structure,” ready to paste. DeepWiki, from Devin, goes further and generates documentation pages for the repository first. “I love all the little tools that basically where you just change the URL and it makes something accessible to an LLM.”

Meeting the models halfway

Karpathy anticipates the objection to all of this, which is that models are already learning to click things themselves.

He grants it, and immediately says it is not the point. Agents can click around today, “but I still think it’s very worth basically meeting LLMs halfway and making it easier for them to access all this information, because this is still fairly expensive, I would say, to use and a lot more difficult.”

There is also a long tail that will never come to meet them. Those apps are “not like live player sort of repositories or digital infrastructure,” and for them the scrapers and converters are the only route in. “But I think for everyone else, I think it’s very worth kind of like meeting in some middle point. So I’m bullish on both.”

Taking the slider from left to right

He ends where he started, with the people about to enter the industry. “What an amazing time to get into the industry. We need to rewrite a ton of code. A ton of code will be written by professionals and by coders.” The models are a bit like utilities, a bit like fabs, and most of all like operating systems, “but it’s so early. It’s like 1960s of operating systems.” They are fallible people spirits.

He leaves them with the dial. “There should be an autonomy slider in your product, and you should be thinking about how you can slide that autonomy slider and make your product more autonomous over time.”

He closes on the same image. “Going back to the Iron Man suit analogy, I think what we’ll see over the next decade roughly is we’re going to take the slider from left to right. It’s going to be very interesting to see what that looks like.”

Then he hands it to the room. “And I can’t wait to build it with all of you.”