Please welcome former director of AI Tesla Andre Carpathy. [Music] Hello. [Music] Wow, a lot of people here. Hello. Um, okay. Yeah. So I'm excited to be here today to talk to you about software in the era of AI. And I'm told that many of you are students like bachelors, masters, PhD and so on. And you're about to enter the industry. And I think it's actually like an extremely unique and very interesting time to enter the industry right now. And I think fundamentally the reason for that is that um software is changing uh again. And I say again because I actually gave this talk already. Um but the problem is that software keeps changing. So I actually have a lot of material to create new talks and I think it's changing quite fundamentally. I think roughly speaking software has not changed much on such a fundamental level for 70 years. And then it's changed I think about twice quite rapidly in the last few years. And so there's just a huge amount of work to do a huge amount of software to write and rewrite. So let's take a look at maybe the realm of software. So if we kind of think of this as like the map of software this is a really cool tool called map of GitHub. Um this is kind of like all the software that's written. Uh these are instructions to the computer for carrying out tasks in the digital space. So if you zoom in here, these are all different kinds of repositories and this is all the code that has been written. And a few years ago I kind of observed that um software was kind of changing and there was kind of like a new type of software around and I called this software 2.0 at the time and the idea here was that software 1.0 is the code you write for the computer. Software 2.0 know are basically neural networks and in particular the weights of a neural network and you're not writing this code directly you are most you are more kind of like tuning the data sets and then you're running an optimizer to create to create the parameters of this neural net and I think like at the time neural nets were kind of seen as like just a different kind of classifier like a decision tree or something like that and so I think it was kind of like um I think this framing was a lot more appropriate and now actually what we have is kind of like an equivalent of GitHub in the realm of software 2.0 And I think the hugging face is basically equivalent of GitHub in software 2.0. And there's also model atlas and you can visualize all the code written there. In case you're curious, by the way, the giant circle, the point in the middle, uh these are the parameters of flux, the image generator. And so anytime someone tunes a on top of a flux model, you basically create a git commit uh in this space and uh you create a different kind of a image generator. So basically what we have is software 1.0 is the computer code that programs a computer. Software 2.0 are the weights which program neural networks. Uh and here's an example of Alexet image recognizer neural network. Now so far all of the neural networks that we've been familiar with until recently where kind of like fixed function computers image to categories or something like that. And I think what's changed and I think is a quite fundamental change is that neural networks became programmable with large language models. And so I I see this as quite new, unique. It's a new kind of a computer and uh so in my mind it's uh worth giving it a new designation of software 3.0. And basically your prompts are now programs that program the LLM. And uh remarkably uh these uh prompts are written in English. So it's kind of a very interesting programming language. Um so maybe uh to summarize the difference if you're doing sentiment classification for example you can imagine writing some uh amount of Python to to basically do sentiment classification or you can train a neural net or you can prompt a large language model. Uh so here this is a few short prompt and you can imagine changing it and programming the computer in a slightly different way. So basically we have software 1.0 software 2.0 and I think we're seeing maybe you've seen a lot of GitHub code is not just like code anymore. there's a bunch of like English interspersed with code and so I think kind of there's a growing category of new kind of code. So not only is it a new programming paradigm, it's also remarkable to me that it's in our native language of English. And so when this blew my mind a few uh I guess years ago now I tweeted this and um I think it captured the attention of a lot of people and this is my currently pinned tweet uh is that remarkably we're now programming computers in English. Now, when I was at uh Tesla, um we were working on the uh autopilot and uh we were trying to get the car to drive and I sort of showed this slide at the time where you can imagine that the inputs to the car are on the bottom and they're going through a software stack to produce the steering and acceleration and I made the observation at the time that there was a ton of C++ code around in the autopilot which was the software 1.0 code and then there was some neural nets in there doing image recognition and uh I kind of observed that over time 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 and kind of like was um and a lot of the kind of capabilities and

functionality that was originally written in 1.0 was migrated to 2.0. So as an example, a lot of the stitching up of information across images from the different cameras and across time was done by a neural network 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. So I thought this was really remarkable at the time and I think we're seeing the same thing again where uh basically we have a new kind of software and it's eating through the stack. We have three completely different programming paradigms and I think 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 and you may want to program some functionality in 1.0 or 2.0 or 3.0. Are you going to train neurallet? Are you going to just prompt an LLM? Should this be a piece of code that's explicit etc. So we all have to make these decisions and actually potentially uh fluidly trans transition between these paradigms. So what I wanted to get into now is first I want to in the first part talk about LLMs and how to kind of like think of this new paradigm and the ecosystem and what that looks like. Uh like what are what is this new computer? What does it look like and what does the ecosystem look like? Um I was struck by this quote from Anduring actually uh many years ago now I think and I think Andrew is going to be speaking right after me. Uh but he said at the time AI is the new electricity and I do think that it um kind of captures something very interesting in that LLMs certainly feel like they have properties of utilities right now. So um LLM labs like OpenAI, Gemini, Enthropic etc. They spend capex to train the LLMs and this is kind of equivalent to building out a grid and then there's opex to serve that intelligence over APIs to all of us and this is done through metered access where we pay per million tokens or something like that and we have a lot of demands that are very utility- like demands out of this API we demand low latency high uptime consistent quality etc. In electricity, you would have a transfer switch. So you can transfer your electricity source from like grid and solar or battery or generator. In LLM, we have maybe open router and easily switch between the different types of LLMs that exist. Because the LLM 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, right? Because they don't compete in such a direct way. And I think what's also a little fascinating and we saw this in the last few days actually a lot of the LLMs went down and people were kind of like stuck and unable to work. And uh 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. It's kind of like when the voltage is unreliable in the grid and uh the planet just gets dumber the more reliance we have on these models, which already is like really dramatic and I think will continue to grow. But LLM's don't only have properties of utilities. I think it's also fair to say that they have some properties of fabs. And the reason for this is that the capex required for building LLM is actually quite large. Uh it's not just like building some uh power station or something like that, right? You're investing a huge amount of money and I think the tech tree and uh for the technology is growing quite rapidly. So we're in a world where we have sort of deep tech trees, research and development secrets that are centralizing inside the LLM labs. Um and but I think 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. And so um I think it's just an interesting kind of thing to think about potentially. There's many analogy analogies you can make like a 4 nanometer process node maybe is something like a cluster with certain max flops. You can think about when you're use 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. So I think there's some analogies here that make sense. But actually I think the analogy that makes the most sense perhaps is that in my mind LLM have very strong kind of analogies to operating systems. Uh in that this is not just electricity or water. It's not something that comes out of the tap as a commodity. uh this is these are now increasingly complex software ecosystems right so uh they're not just like simple commodities like electricity and it's kind of interesting to me that the ecosystem is shaping in a very similar kind of way where you have a few closed source providers like Windows or Mac OS and then you have an open source alternative like Linux and I think for u neural for LLMs as well we have a kind of a few competing closed source providers and then maybe the llama ecosystem is currently like maybe a close approximation to something that may grow into something like Linux. Again, I think 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 multiodalities and how all of that works. And so when I sort of had this realization a while back, I tried to sketch it out and it kind of seemed to me like LLMs are kind of like a new operating system, right? So the LLM is a new kind of a computer. It's sitting it's kind of like the CPU equivalent. uh the context windows are kind of like the memory and then the LLM is orchestrating memory and compute uh for problem solving um using all of these uh capabilities here and so definitely if you look at it looks very much like operating system from that perspective. Um, a few more analogies. For example, 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 or Mac in the same way as you can take an LLM app like cursor and you can run it on GPT or cloud or Gemini series, right? It's just a drop down. So, it's kind of like similar in that way as well. uh more analogies that I think strike me is that we're kind of like in this 1960sish era where LLM compute is still very expensive for this new kind of a computer and that forces the LLMs to be centralized in the cloud and we're all just uh sort of thing 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 you know a dimension of the batch when they're running the computer in the cloud. And this is very much what computers used to look like at during this time. The operating systems were in the cloud. Everything was streamed around and there was batching. And so the p the personal computing revolution hasn't happened yet because it's just not economical. It doesn't make sense. But I think some people are trying. And it turns out that Mac minis, for example, are a very good fit for some of the LLMs because it's all if you're doing batch one inference, this is all super memory bound. So this actually works. And uh I think these are some early indications maybe of personal computing. Uh but this hasn't really happened yet. It's not clear what this looks like. Maybe some of you get to invent what what this is or how it works or uh what this should what this should be. Maybe one more analogy that I'll mention is whenever I talk to Chach or some LLM directly in text, I feel like I'm talking to an operating system through the terminal. Like it's just it's it's text. It's direct access to the operating system. And I think a guey hasn't yet really been invented in like a general way like should chatt have a guey like different than just a tech bubbles. Uh certainly some of the apps that we're going to go into in a bit have guey but there's no like guey across all the tasks if that makes sense. Um there are some ways in which LLMs are different from kind of operating systems in some fairly unique way and from early computing. And I wrote about uh this one particular property that strikes me as very different uh this time around. It's that LLMs like flip they flip the direction of technology diffusion uh that is usually uh present in technology. So for example with electricity, cryptography, computing, flight, internet, GPS, lots of new transformative technologies that have not been around. Typically it is the government and corporations that are the first users because it's new and expensive etc. and it only later diffuses to consumer. Uh, but I feel like LLMs are kind of like flipped around. So 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 or some special technology. Indeed, corporations are governments are lagging behind the adoption of all of us, of all of these technologies. So, it's just backwards and I think it informs maybe some of the uses of how we want to use this technology or like where are some of the first apps and so on. So, in summary so far, LLM labs LLMs. I think it's accurate language to use, but LLMs are complicated operating systems. They're circa 1960s in computing and we're redoing computing all over again. and they're currently available via time sharing and distributed like a utility. What is new and unprecedented is that 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 Chaship was beamed down to our computers like billions of people like instantly and overnight and this is insane. Uh and it's kind of insane to me that this is the case and now it is our time to enter the industry and program these computers. This is crazy. So I think this is quite remarkable. Before we program LLMs, we have to kind of like spend some time to think about what these things are. And I especially like to kind of talk about their psychology. So the way I like to think about LLMs is that they're kind of like people spirits. Um they are stoastic simulations of people. Um and the simulator in this case happens to be an auto reggressive transformer. So transformer is a neural net. Uh it's and it just kind of like is goes on the level of tokens. It goes chunk

chunk chunk chunk chunk. And there's an almost equal amount of compute for every single chunk. Um and um this simulator of course is is just is basically there's some weights involved and we fit it to all of text that we have on the internet and so on. And you end up with this kind of a simulator and because it is trained on humans, it's got this emergent psychology that is humanlike. So the first thing you'll notice is of course uh LLM have encyclopedic knowledge and memory. uh and they can remember lots of things, a lot more than any single individual human can because they read so many things. It's it actually kind of reminds me of this movie Rainman, which I actually really recommend people watch. It's an amazing movie. I love this movie. Um and Dustin Hoffman here is an autistic savant who has almost perfect memory. So, he can read a he can read like a phone book and remember all of the names and phone numbers. And I kind of feel like LM are kind of like very similar. They can remember Shaw hashes and lots of different kinds of things very very easily. So they certainly have superpowers in some set in some respects. But they also have a bunch of I would say cognitive deficits. So they hallucinate quite a bit. Um and they kind of make up stuff and don't have a very good uh sort of internal model of self-nowledge, not sufficient at least. And this has gotten better but not perfect. They display jagged intelligence. So they're going to be superhuman in some problems solving domains. And then they're going to make mistakes that basically no human will make. like you know they will insist that 9.11 is greater than 9.9 or that there are two Rs in strawberry these are some famous examples but basically there are rough edges that you can trip on so that's kind of I think also kind of unique um they also kind of suffer from entrograde amnesia um so uh and I think I'm alluding to the fact that if you have a co-orker who joins your organization this co-orker will over time learn your organization and uh they will understand and gain like a huge amount of context on the organization and they go home and they sleep and they consolidate knowledge and they develop expertise over time. LLMs don't natively do this and this is not something that has really been solved in the R&D; of LLM. I think um and so context windows are really kind of like working memory and you have to sort of program the working memory quite directly because they don't just kind of like get smarter by uh by default and I think a lot of people get tripped up by the analogies uh in this way. Uh in popular culture I recommend people watch these two movies uh Momento and 51st dates. In both of these movies, the protagonists, their weights are fixed and their context windows gets wiped every single morning and it's really problematic to go to work or have relationships when this happens and this happens to all the time. I guess one more thing I would point to is security kind of related limitations of the use of LLM. So for example, LLMs are quite gullible. Uh they are susceptible to prompt injection risks. They might leak your data etc. And so um and there's many other considerations uh security related. So, so basically long story short, you have to load your you have to load your you have to simultaneously think through this superhuman thing that has a bunch of cognitive deficits and issues. How do we and yet they are extremely like useful and so how do we program them and how do we work around their deficits and enjoy their superhuman powers. So what I want to switch to now is talk about the opportunities of how do we use these models and what are some of the biggest opportunities. This is not a comprehensive list just some of the things that I thought were interesting for this talk. The first thing I'm kind of excited about is what I would call partial autonomy apps. So for example, let's work with the example of coding. You can certainly go to chacht directly and you can start copy pasting code around and copyping bug reports and stuff around and getting code and copy pasting everything around. Why would you why would you do that? Why would you go directly to the operating system? It makes a lot more sense to have an app dedicated for this. And so I think many of you uh use uh cursor. I do as well. And uh cursor is kind of like the thing you want instead. You don't want to just directly go to the chash apt. And I think cursor is a very good example of an early LLM app that has a bunch of properties that I think are um useful across all the LLM apps. So in particular, you will notice that 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. And so some of the properties of LLM apps that I think are shared and useful to point out. Number one, the LLMs basically do a ton of the context management. Um, number two, they orchestrate multiple calls to LLMs, right? So in the case of 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. A really big one that uh I think also maybe not fully appreciated always is application specific uh GUI and the importance of it. Um because you don't just want to talk to

the operating system directly in text. Text is very hard to read, interpret, understand and also like you don't want to take some of these actions natively in text. So it's much better to just see a diff as like red and green change and you can see what's being added is subtracted. 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 guey allows a human to audit the work of these fallible systems and to go faster. I'm going to come back to this point a little bit uh later as well. And the last kind of feature I want to point out is that there's what I call the autonomy slider. So, for example, in cursor, you can just do tap completion. You're mostly in charge. You can select a chunk of code and command K to change just that chunk of code. You can do command L to change the entire file. Or you can do command I which just you know let it rip do whatever you want in the entire repo and that's the sort of full autonomy agent agentic version and so you are in charge of the autonomy slider and depending on the complexity of the task at hand you can uh tune the amount of autonomy that you're willing to give up uh for that task maybe to show one more example of a fairly successful LLM app uh perplexity um it also has very similar features to what I've just pointed out to in cursor uh it packages up a lot of the information. It orchestrates multiple LLMs. It's got a GUI that allows you to audit some of its work. So, for example, it will site sources and you can imagine inspecting them. And it's got an autonomy 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. So, this is all just varying levels of autonomy that you give up to the tool. So, I guess my question is I feel like a lot of software will become partially autonomous. I'm trying to think through like what does that look like? And for many of you who maintain products and services, 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? Because again, these are fallible systems that aren't yet perfect. And what does a diff look like in Photoshop or something like that? You know, and also a lot of the traditional software right now, it has all these switches and all this kind of stuff that's all designed for human. All of this has to change and become accessible to LLMs. So, one thing I want to stress with a lot of these LLM apps that I'm not sure gets as much attention as it should is um we we're now kind of like cooperating with AIS and usually they are doing the generation and we as humans are doing the verification. It is in our interest to make this loop go as fast as possible. So, we're getting a lot of work done. There are two major ways that I think uh this can be done. Number one, you can speed up verification a lot. Um, and I think guies, for example, are extremely important to this because a guey 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 it's just a kind of like a highway to your brain. So, I think guies are very useful for auditing systems and visual representations in general. And number two, I would say is we have to keep the AI on the leash. We I think a lot of people are getting way over excited with AI agents and uh it's not useful to me to get a diff of 10,000 lines of code to my repo. Like I have to I'm still the bottleneck, right? Even though that 10,00 lines come out instantly, I have to make sure that this thing is not introducing bugs. It's just like and that it's doing the correct thing, right? And that there's no security issues and so on. So um I think that um yeah basically you we have to sort of like it's in our interest to make the the flow of these two go very very fast and we have to somehow keep the AI on the leash because it gets way too overreactive. It's uh it's kind of like this. This is how I feel when I do AI assisted coding. If I'm just bite 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 uh agent doing all this kind of stuff. So this slide is not very good. I'm sorry, but I guess I'm trying to develop like many of you some ways of utilizing these agents in my coding workflow and to do AI assisted coding. And 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 and um I sort of work on small chunks of single concrete thing. Uh and so I think many of you probably are developing similar ways of working with the with LLMs. Um, I also saw a number of blog posts that try to develop these best practices for working with LLMs. And here's one that I read recently and I thought was quite good. And it kind of discussed some techniques and some of them have to do with how you keep the AI on the leash. And so, as an example, if you are prompting, if your prompt is vague, then uh 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. So it makes a lot

more sense to spend a bit more time to be more concrete in your prompts which increases the probability of successful verification and you can move forward. And so I think a lot of us are going to end up finding um kind of techniques like this. I think in my own work as well I'm currently interested in uh what education looks like in um together with kind of like now that we have AI uh and LLMs what does education look like? And I think a a large amount of thought for me goes into how we keep AI on the leash. I don't think it just works to go to chat and be like, "Hey, teach me physics." I don't think this works because the AI is like gets lost in the woods. And so 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. And 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. and the AI is kept on the leash with respect to a certain syllabus, a certain like um progression of projects and so on. And so this is one way of keeping the AI on leash and I think has a much higher likelihood of working and the AI is not getting lost in the woods. One more kind of analogy I wanted to sort of allude to is I'm not I'm no stranger to partial autonomy and I kind of worked on this I think for five years at Tesla and this is also a partial autonomy product and shares a lot of the features like for example right there in the instrument panel is the GUI of the autopilot so it's showing me what the what the neural network sees and so on and we have the autonomy slider where over the course of my tenure there we did more and more autonomous tasks for the user and maybe the story that I wanted to tell very briefly is uh actually the first time I drove a self-driving vehicle was in 2013 and I had a friend who worked at Whimo and uh he offered to give me a drive around Palo Alto. I took this picture using Google Glass at the time and many of you are so young that you might not even know what that is. Uh but uh yeah, this was like all the rage at the time. And we got into this car and we went for about a 30-minute drive around Palo Alto highways uh streets and so on. And this drive was perfect. There was zero interventions and this was 2013 which is now 12 years ago. And it kind of struck me because at the time when I had this perfect drive, this perfect demo, I felt like, wow, self-driving is imminent because this just worked. This is incredible. Um, but here we are 12 years later and we are still working on autonomy. Um, we are still working on driving agents and even now we haven't actually like really solved the problem. like you may see Whimos going around and they look driverless but you know there's still a lot of teleoperation and a lot of human in the loop of a lot of this driving so we still haven't even like declared success but I think it's definitely like going to succeed at this point but it just took a long time and so I think like like this is software is really tricky I think in the same way that driving is tricky and so 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 and this is going to be quite some time. We need humans in the loop. We need to do this carefully. This is software. Let's be serious here. One more kind of analogy that I always think through is the Iron Man suit. Uh I think this is I always love Iron Man. I think it's like so um correct in a bunch of ways with respect to technology and how it will play out. And 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. And in some of the movies, the Iron Man suit is quite autonomous and can fly around and find Tony and all this kind of stuff. And so this is the autonomy slider is we can be we can build augmentations or we can build agents and we kind of want to do a bit of both. But at this stage I would say working with fallible LLMs and so on. I would say you know 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. And these products have custom gueies and UIUX. And we're trying to um and this is done so that the generation verification loop of the human is very very fast. But we are not losing the sight of the fact that it is in principle possible to automate this work. And 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 uh sort of um more autonomous over time. But this is kind of how I think there's lots of opportunities in these kinds of products. I want to now switch gears a little bit and talk about one other dimension that I think is very unique. Not only is there a new type of programming language that allows for autonomy in software but also as I mentioned it's programmed in English which is this natural interface and suddenly everyone is a programmer because everyone speaks natural language like English. So this is extremely bullish and very interesting to me and also completely unprecedented. I would say it 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. So, I don't know if by any chance anyone has heard of vibe coding. Uh, this this is the tweet that kind of like introduced this, but I'm told that this is now like a major meme. Um, fun story about this is that 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 like fizzles and no one cares. And I thought that this tweet was going to be the latter. I don't know. It was just like a shower of thoughts. But this became like a total meme and I really just can't tell. But I guess like it struck a chord and 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. This is like [Applause] yeah this is like a major contribution now or something like that. So, um, so Tom Wolf from HuggingFace shared this beautiful video that I really love. Um, these are kids vibe coding. And I find that this is such a wholesome video. Like, I love this video. Like, how can you look at this video and feel bad about the future? The future is great. I think this will end up being like a gateway drug to software development. Um, I'm not a doomer about the future of the generation and I think yeah, I love this video. So, I tried by coding a little bit uh as well because it's so fun. Uh, so bike coding is so great 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 or something like that. So, I built this uh iOS app and I don't 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 uh dumb, but uh I kind of like this was just like a day of work and this was running on my phone like later that day and I was like, "Wow, this is amazing." I didn't have to like read through Swift for like five days or something like that to like get started. I also vipcoded this app called Menu Genen. And this is live. You can try it in menu.app. And I basically had this problem where 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 this doesn't exist. So I was like, "Hey, I'm going to bite code it." So, um, this is what it looks like. You go to menu.app, um, and, uh, you take a picture of a of a menu and then menu generates the images and everyone gets $5 in credits for free when you sign up. And therefore, this is a major cost center in my life. So, this is a negative negative uh, revenue app for me right now. I've lost a huge amount of money on menu. Okay. But the fascinating thing about menu genen for me is that the code of the v the vite coding part the code was actually the easy part of v of v coding menu and most of it actually was when I tried to make it real so that you can actually have authentication and payments and the domain name and averal deployment. This was really hard and all of this was not code. All of this devops stuff was in me in the browser clicking stuff and this was extreme slo and took another week. So it was really fascinating that I had the menu genen um basically demo working on my laptop in a few hours and then it took me a week because I was trying to make it real and the reason for this is this was just really annoying. Um, so for example, if you try to add Google login to your web page, I know this is very small, but just a huge amount of instructions of this clerk library telling me how to integrate this. And this is crazy. Like it's telling me go to this URL, click on this dropdown, choose this, go to this, and click on that. And it's like 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. So I think the last part of my talk therefore focuses on can we just build for agents? I don't want to do this work. Can agents do this? Thank you. Okay. So roughly speaking, I think 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 and agents are they're computers but they are humanlike kind of right they're people spirits there's people spirits on the internet and they need to interact with our software infrastructure like can we build for them it's a new thing so as an example you can have robots.txt on your domain and you can instruct uh or like advise I suppose um uh web crawlers on how to behave on your website in the same way you can have maybe lm.txt txt file which is just a simple markdown that's telling LLMs what this domain is about and this is very readable to a to an LLM. If it had to instead get the HTML of your web page and try to parse it, this is very errorprone and difficult and will screw it up and it's not going to work. So we can just directly speak to the LLM. It's worth it. Um a huge amount of documentation is currently written for people. So you will see things like lists and bold and pictures and this is not directly accessible by an LLM. So I see some of the services now are transitioning a lot of the their docs to be specifically for LLMs. So Versell and Stripe as an example are early movers here but there are a few more that I've seen already and they offer their documentation in

markdown. Markdown is super easy for LMS to understand. This is great. Um maybe one simple example from from uh my experience as well. Maybe some of you know three blue one brown. He makes beautiful animation videos on YouTube. [Applause] Yeah, I love this library. So that he wrote uh Manon and I wanted to make my own and uh there's extensive documentations on how to use manon and so I didn't want to actually read through it. So I copy pasted the whole thing to an LLM and I described what I wanted and it just worked out of the box like LLM just bcoded me an animation exactly what I wanted and I was like wow this is amazing. So if we can make docs legible to LLMs, it's going to unlock a huge amount of um kind of use and um I think this is wonderful and should should happen more. The other thing I wanted to point out is that you do unfortunately have to it's not just about taking your docs and making them appear in markdown. That's the easy part. 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, Verscell, for example, is replacing every occurrence of click with an equivalent curl command that your LM agent could take on your behalf. Um, and so I think this is very interesting. And then, of course, there's a model context protocol from Enthropic. And this is also another way, it's a protocol of speaking directly to agents as this new consumer and manipulator of digital information. So, I'm very bullish on these ideas. The other thing I really like is a number of little tools here and there that are helping ingest data that in like very LLM friendly formats. So for example, when I go to a GitHub repo like my nanoGPT repo, I can't feed this to an LLM and ask questions about it uh because it's you know this is a human interface on GitHub. So when you just change the URL from GitHub to get ingest then uh this will actually concatenate all the files into a single giant text and it will create a directory structure etc. And this is ready to be copy pasted into your favorite LLM and you can do stuff. Maybe even more dramatic example of this is deep wiki where it's not just the raw content of these files. uh this is from Devon but also like they have Devon basically do analysis of the GitHub repo and Devon basically builds up a whole docs uh pages just for your repo and you can imagine that this is even more helpful to copy paste into your LLM. So I love all the little tools that basically where you just change the URL and it makes something accessible to an LLM. So this is all well and great and u I think there should be a lot more of it. One more note I wanted to make is that it is absolutely possible that in the future LLMs will be able to this is not even future this is today they'll be able to go around and they'll be able to click stuff and so on but I still think it's very worth u basically meeting LLM halfway LLM's halfway and making it easier for them to access all this information uh because this is still fairly expensive I would say to use and uh a lot more difficult and so I do think that lots of software there will be a long tail where it won't like adapt apps because these are not like live player sort of repositories or digital infrastructure and we will need these tools. Uh 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 if that makes sense. So in summary, 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. These LLMs are kind of like utilities, kind of like fabs, but they're kind of especially like operating systems. But it's so early. It's like 1960s of operating systems and uh and I think a lot of the analogies cross over. Um and these LMS are kind of like these fallible uh you know people spirits that we have to learn to work with. And in order to do that properly, we need to adjust our infrastructure towards it. So when you're building these LLM apps, I describe some of the ways of working effectively with these LLMs and some of the tools that make that uh kind of possible and how you can spin this loop very very quickly and basically create partial tunneling products and then um yeah, a lot of code has to also be written for the agents more directly. But in any case, 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. And I'm very interesting. It's going to be very interesting to see what that looks like. And I can't wait to build it with all of you. Thank you.