**Mathew Lodge Guest** 00:00

In reinforcement learning, you're going for accuracy. You're conducting this search, you're trying to find the best possible answer you can, the most accurate answer you can. In case of unit testing, it's how close to 100% coverage can be get, and in the case of Alpha Dev, it's how fast can we make this algorithm? Is it faster than the previous algorithm that I've come up with? And so you're very focused on a goal, In contrast with large language models, which don't have a goal at all. Their goal is to be very general. So you've gone from a very specific task in reinforcement learning to a more general task with large language models, where you're going to predict what comes next and it's OK if it's not exactly correct. You're trading accuracy for generality.

00:40

Large language models the code is usually syntactically correct. We've never seen it generate code that doesn't have the right syntax. What it gets wrong is semantics, because it doesn't have a model for semantics. It doesn't have a model of the language. It just knows what it's seen before and its end patterns. That it's seen before and that's what it tends to get wrong.

**Craig Smith Host** 00:59

I'm Craig Smith and this is I on AI. This week I talked to Mathew Lodge , CEO of Diff Blue, a company that uses reinforcement learning to automate the writing of unit tests for Java codes. Matthew talks about how reinforcement learning has been lost in the shuffle around generative AI, but remains one of the most powerful kinds of artificial intelligence in any machine learning developers toolbox, and, in fact, how reinforcement learning is the power behind much of generative AI. I found the conversation with Matthew fascinating, and I hope you do too. Now here's Matthew. Matthew, it's great to have you. Why don't you start by introducing yourself and then we'll get to some questions?

**Mathew Lodge Guest** 01:59

Great, great Thanks for having me on your podcast. So my name is Mathew Lodge . I am CEO of Diff Blue. We're a generative AI for code company. We're a spin out from Oxford University in the UK. In terms of my background, I started out as a software developer in real time systems and safety critical, so I worked on code that flew on the space station in the Boeing 777. And later on moved to Silicon Valley and been in B2B product management for a long time, worked a couple of years like Cisco and Symantec and VMware, as well as startups. You worked on a startup that did software, defined networking 10 years too early, which was the same as being wrong. I was also a SPP product to Anaconda. So the machine learning, data science, the Python distribution, very popular in the open source world before being CEO at Diff Blue, where we are using reinforcement learning to do generative AI for code. So we have a product our first product that automatically and autonomously writes unit tests for Java applications.

**Craig Smith Host** 03:02

Yeah, and we were talking yesterday about how reinforcement learning, or code generation through reinforcement learning, is kind of lost in the, in the, in the buzz around transformer based large language models which in fact have a lot of problems when it comes to things like code generation. And I was describing to you how I work sort of many hours with auto GPT and with you know, going back and forth between auto GPT and GPT, for you know, every time I'd hit an error then I'd plug it in GPT for it would tell me what the error is and suggest some changes, and go back and forth, and back and forth and I just never get anywhere. So can you talk about that difference you were giving a talk yesterday, I think about about that difference and why people don't talk about reinforcement learning as a solution for code generation.

**Mathew Lodge Guest** 04:19

Yeah, I mean large language models have this huge general appeal, I mean. So GPT has really made this kind of technology accessible to a general audience, and that's really the genius of large language models is that it's very easy for everybody to understand what they can do because they can just see it. And reinforcement learning has traditionally been used in AI for a game playing. So that's where really where it showed its forte. So in the early days of open AI, they were building a game playing AI very successful, you know, a Dota engine that adversarially trained, and that technology essentially is you're going on a search, you're trying to find the best move in game playing. So AlphaGo is a really great example of reinforcement learning led approach to game playing. And so the place where reinforcement learning is really useful is where you have a space of possible answers that's too big to check all of the answers. So people think about deep blue IBM's product from what 30 years ago now, as being this sort of breakthrough chess playing product. But essentially all they did was brute force search the move space and it just looked at every possible move and they picked the best one, and and that's how modern chess software works. There's no neural networks involved, it's they just look at every move when you get to go. You can't do that because number of moves is greater than the number of atoms in the universe Right. So you have to take a different strategy, and that's essentially what reinforcement learning is about is about identifying the areas of the solution space where the best solutions are likely to be found and spending more time searching there and basically ignoring the rest. So you ignore the areas of the space where you're not likely to find the answer. So you're not guaranteed to find the best answer, but you can find a very good one, and essentially that's what AlphaGo does. It has a prediction of who's going to win the game. It uses that to try different moves. It uses Monte Carlo methods, randomness to you and me to suggest additional moves and it tries them, and it makes a prediction about who's going to win the game if it makes that move, and it essentially conducts the search that way. So it's always asking who's going to win, who's going to win, who's going to win, and from that it can predict a try, a move, and then it can essentially follow a sequence of moves and build a playbook for playing the game. And you can do the same with code. So Google has recently done this with AlphaDev.

06:46

So the DeepMind team at Google A couple of weeks ago came out with AlphaDev, which is an approach that they took reinforcement learning and they applied it to very common computer science algorithms. So they picked sorting and hashing algorithms, which are every software program today uses those things multiple times a day. So if you can squeeze and optimize, squeeze some time out of those algorithms, the effect is enormous because they are so frequently used. And so essentially what they did is use reinforcement learning to try different moves. They say tried different sequences, different implementations of the quicksort algorithm, and this is down at the assembly language level. So they were just trying different instructions like what happens if we delete this instruction, what happens if we simplify or change, modify the code, essentially mutate the code. And in that search they were able to find more efficient implementations. So for quicksort the implementation is about 1.7% faster for a large array of numbers, which doesn't sound like much until you remember that this is a function that you get called millions of times and so the cumulative saving is enormous. And for small data sets the improvement is 70% and much bigger for small data sets, which is probably a more common use of the function. So essentially they were searching for better code. And DiffBloose product does the same thing.

08:09

We search for unit tests, we look, we make a guess at what a unit test, a good unit test, would look like for a particular piece of code. So a unit test, the idea is you're isolating a unit of software and what you want to do is find regressions using these tests. So these tests you put in the input to the function. You check the output to make sure it's doing the right thing. You put in enough input, so you cover all the different branches inside of the code. You check the answers for all of those to make sure that they're correct. And what that means then is that when you run the unit test later, after you've modified the code, you can find regressions, so changes in behavior, because a regression could be a bug or it could be a deliberate change to the software. You've changed the way the software works, and so you would expect the results to be different.

08:59

But in the case of unit tests, what the search we're doing is we write the test, we look at the signature of the method, we have some analysis of the program as well to make. So we can make a good guess and we try it. We compile that code, we run it against the method on the test and we see how it did. What kind of coverage did it get, did it exercise the branches, all of those things. Based on that we can predict what a better test might look like. So we compile that, we run it, we try against the code, see how it does, and we iterate through that approximately 10,000 times and at the end of that process we've got a set of tests.

09:29

So it's a very different method to large language models. Large language models are based around the notion of text patterns. When you're training a large language model, it is learning, it's building a statistical model of text and it's using the context of that it knows. In the case of natural language, it knows the role that a token plays. A token is essentially a word in large language model. So it knows that this is a noun or it's a verb, a part of speech it forms. So they're using that information in order to construct the statistical model. And so what you see with large language models is that from the prompt it can do a very good job of figuring out what comes next, and so that kind of technology is kind of like a autocomplete turbocharged autocomplete. Autocomplete's been an IDE feature for years, since the first IDEs came out, but it's able to go a lot further because it's much better at guessing what comes next.

10:34

Essentially, what is happening in the two different techniques is that in reinforcement learning, you're going for accuracy, you're conducting this search, you're trying to find the best possible answer you can, the most accurate answer you can.

10:44

In the case of unit testing, it's how close to 100% coverage can we get, and in the case of AlphaDev, it's how fast can we make this algorithm?

10:53

Is it faster than the previous algorithm that I've come up with? And so you're very focused on a goal, in contrast with large language models, which don't have a goal at all. Their goal is to be very general. So you've gone from a very specific task in reinforcement learning to a more general task with large language models, where you're going to predict what comes next, and it's okay if it's not exactly correct. So you're trading accuracy for generality.

11:22

And in tools like ChatGPT, if you have something to write code or get a co-pilot, or Tab9, tab9's product essentially it's making its best guess at what the next piece of code is. It knows the code that came before, it knows the code after and that's all it knows and it guesses, and that's okay, because you have a developer sitting there looking at the completion and deciding what to do. Does this make sense? Maybe I can. It's not quite correct but I can fix it. So there's very different techniques than it's large language models, because of ChatGPT, have I've been taking all the limelight because that's the thing that people can see and understand.

**Craig Smith Host** 12:01

Yeah, and before AlphaDev, deepmind had AlphaGo. Alphadev is not a product, it's a research project. Is that right?

**Mathew Lodge Guest** 12:13

That's correct. Yes, so they released a paper about AlphaDev, but I don't think it's been released outside of Google.

**Craig Smith Host** 12:20

Yeah, and how does that relieve to? I'm sorry not. I said AlphaGo, I meant AlphaCode, oh yeah.

**Mathew Lodge Guest** 12:31

AlphaCode.

12:33

Yeah yeah, alphacode also a research paper, and AlphaCode was designed to win programming competitions, and so AlphaCode is an example of where you're taking a large language model and asking it to generate lots of different alternative programs to match a specification.

12:54

So the thing that you have that is just completely unlike the real world in the programming competition is you have a very good, unambiguous, clear description of exactly what the kept should do and what the output should be, and so they take that and they use it to synthesize many different versions of the program, because the other thing with large language models is this stochastic component, the randomization that you can dial that up and down in most of most LLMs, and so it will can generate different alternatives essentially, and so the idea is that you have a better chance of coming up with the best alternative that way, and so in AlphaCode they take that description, they generate a whole bunch of candidate programs and then they run cluster analysis on those programs because they want to find out.

13:47

A lot of those programs effectively do the same thing. The code might not be exactly the same, but they do the same thing. They're semantically equivalent, maybe not syntactically equivalent, so the code doesn't match word for word, but it does the same thing. So they cluster all of those to find out to eliminate duplicates, and then they essentially cull it down to a couple of candidates and they try those and that's how it produces solutions. So it's it's very good at winning programming competitions, but it's not really will a real world thing.

**Craig Smith Host** 14:19

Yeah, but it sounds as though for code generation, you could write us a system or build a system that does generate accurate code using reinforcement learning. And why hasn't that been done?

**Mathew Lodge Guest** 14:41

Well, I think it has been done. It's like it's like the, the old phrase that's like in the future is unevenly distributed. So you know, so we've definitely. We've been doing this, for you know, since 2016,. There's an open source project called Evo Suite. They started in 2012,. They built a reinforcement learning research project, as originally for a number of academics at Sheffield University and elsewhere. They built Evo Suite and they use reinforcement learning to find unit tests and they've been doing that for a very long time. So it's a it's. You're starting to see more products based on reinforcement learning, you know, while LLMs have, you know, have grabbed all the headlines because of their broad appeal.

**Craig Smith Host** 15:32

Yeah, we talked about the threat discussion with that's. That's again grabbing all the headlines in the last month or so and you were telling me that this is actually a problem that's existed for a long time and has been dealt with for a long time. Can you, can you, go back over?

**Mathew Lodge Guest** 15:57

that? Yeah, it's so. It reminds me of the discussion that was had very early on around software controlled what are called safety critical systems. So you can think of a safety critical system as well. Or if it gets it wrong, people die, that's the easiest way to think about it. So we're talking about the software that flies aircraft and other drones, things like that signaling control for things like, you know, train networks, railroad networks. That software has to be absolutely correct.

16:27

And you know Boeing, some of the very early stuff that I worked on, so both Airbus and Boeing at a time working on flyby wire systems. Airbus did it, did it first and Boeing followed along later, and essentially those systems the you know the effect of getting it wrong is that people die, and so they, those systems are very highly verified and so the behavior of those systems is very well understood and they have this notion of the envelope sort of in the think of the flight envelope of an aircraft. If you exceed the envelope in some way, then bad things happen. So you know, maybe you stress the aircraft so much that things you know the wings break off, or or you know, stabilize or something like comes off the aircraft. You can't, you can't exceed the parameters of the flight for a particular airframe, and so a lot of work goes into verifying those systems, and essentially what they do is they proved that they can't exceed that flight envelope, so they there's a lot of work around making their systems deterministic in operation so that you can apply these proofs to it and you can constrain what the algorithm will do.

17:39

It strikes me that's a good analogy for what some of the requests offer around large language models. If you start hooking these things up to real world systems, then you need to understand the envelope of solutions, and that is one of the big challenges with anything that's based on your network is the inherent unpredictability. You have a giant statistical model that humans can't understand. It's too complex, too complicated. There are starting to be companies out there that are segmenting the input space and looking at how you can test these algorithms and understand this envelope more, but essentially, I think that's the next challenge for the AI industry is think about it the way safety-critic people do or have done for decades.

**Craig Smith Host** 18:28

But why then this near hysteria about unleashing large language models or large transformer-based models and their potential for catastrophic outcomes? Is it because they cannot be controlled through these different constraints? I'm really puzzled by this debate.

**Mathew Lodge Guest** 19:08

Yeah, I don't pretend to understand everybody's point of view in this debate because, as you say, there are wide disagreements between the various participants. I can tell you how I think about it. I think there are really a couple of different, distinct things that people worry about. One is the misinformation challenge.

**Craig Smith Host** 19:31

Yeah.

**Mathew Lodge Guest** 19:32

Yeah, you can ask GPT to write something in the style of somebody else. You can ask it to write in the style of Gordon Ramsay and it'll sound like he does on Hell's Kitchen or something like that. So you can see the potential for misuse in those areas. I think the more general challenge is the idea that if you start hooking this up to things that impact the real world, I think that's the bigger concern, because we just don't really understand them very well. We don't understand how they work. We can't predict them. I think that's where a lot of this comes from. I don't buy into the world. It's going to end hysteria and this is a new form of intelligence. It's not a new form of intelligence.

20:16

It's a statistical model of text and the work that we do at DiffBlue around with large language models in using them to generate code, what we find is that large language models, the code is usually syntactically correct. We've never seen it generate code that doesn't have the right syntax. What it gets wrong is semantics, because it doesn't have a model for semantics. It doesn't have a model of the language, it just knows what it's seen before and patterns that it's seen before and that's what it tends to get wrong. And in your example, with auto GPT style, it produces some code that maybe doesn't compile or doesn't run, and so you ask it to go fix the code that it generates.

21:03

It has no idea what the code does. It is just looking at patterns in that code and trying to problem solve by suggesting a different pattern. And that's the inherent challenge with large language models. So there's limited understanding as large language models and if you don't have a way to validate the result of what it's producing and you just take that and use it verbatim and just assume that it's correct, then you can get into this very quickly, spiral into completely off the rails, because the model is not trying to be correct, it's trying to do the best it can with a single shot. These algorithms are deliberately trading accuracy for generality.

**Craig Smith Host** 21:50

Yeah, and going back to code generation in particular, since code generation is so useful for everybody in that there's so much software that needs to be written as the economy continues to digitize. Why haven't, for example, why hasn't, from your point of view, hasn't Google put more focus on alpha code that writes complete, albeit simple, programs, so that we can use natural language to instruct a reinforcement learning system to write code for us that compiles and executes?

**Mathew Lodge Guest** 22:52

Yeah Well, I don't know the answer to that, but I do know that they've followed suit around a co-pilot style product. So there are now three of those You've got a co-pilot, you've got Code Whisperer from Amazon AWS and now there's a version of Bard that does coding, and the advantage of those approaches is that they're highly accessible for developers, because you can just put something in and it will generate some code inside of your IDE, and that's great. It's very simple from an interaction model, and I think it's what we've seen in the case of OpenAI. I don't know about how Google thinks about this, but in case of OpenAI, they clearly believe that large language models are on the road to general artificial intelligence, agi, artificial general intelligence, and so they're trying to push that technology as far as it can go, and they're not interested in things that are less general as a result.

23:55

So I don't know if that's true at Google or not.

**Craig Smith Host** 23:58

But certainly it would be useful if there were systems that could generate code that were not general, that were more accurate.

**Mathew Lodge Guest** 24:11

Yeah, absolutely. I think there's plenty of room for that. I mean, you can see this very quickly being useful in today's low-code products. The problem out there is there's more software to bring in than there are people to do it. So you've got several different approaches to solving that problem. You've got the no-code things, where stuff that really doesn't have to have code and could have like a flow chart. You can just build those things. That's a very well-established market's been around well over 10 years. You've got the low-code things where you're trying to just simplify it. People are out. Systems have been doing this for over a decade, but I think there's more modern versions of those products that try and make it even simpler. It seems like there's a lot of opportunity to you to marry the two together and get something that can produce useful results with even lower code. I don't know what you would call that.

**Craig Smith Host** 25:07

Have you used co-pilot, or do you guys use co-pilot in your work?

**Mathew Lodge Guest** 25:12

We don't use co-pilot in our work. Well, it's very controlled. So part of the issue is we have a lot of code that our customers give us and we agree to keep that confidential, not share that with anyone. So we're incredibly careful about how we use any kind of tool where we would be sending code outside of our organization. So we don't tend to use it for that reason and essentially it's on an exceptional basis.

**Craig Smith Host** 25:39

Can we talk about intelligence? You dismiss this idea of intelligence as opposed to statistical models. It's something that I've talked to a lot of people about, and of course it gets into a semantic debate about what intelligence is. But can you give your view on how much large language models really exhibit intelligence? And then this goes on to the whole discussion of sentience and consciousness and that sort of thing. Yeah, I was telling you, I had a call with Noam Chomsky and he's very dismissive of what large language models do and does not believe that there is any real intelligence being exhibited there.

**Mathew Lodge Guest** 26:54

Yes. So I mean, nobody knows the answer it's a fun thing to debate is like, what is consciousness? Some kind of electrochemical process that's going on in our brains, but we don't really know how that works. And although neural networks are inspired by the way the brain works, but they're really not a model of how the brain works at all. They're not close. It's like a Hollywood movie that's based on a true story.

27:28

So the thing that what you tend to see is blog posts where they said, well, I asked GPT this thing and it seemed to have a theory of mind. Right, it seemed to do these things. And then you can, somebody else comes up with a counter example. So they get one example and they stop and it's like, well, you need more than one example and you need to be able to relate that back to why is it doing this and what is the mechanism that's at work here, that? Why do you think it has a theory of mind? You have to be able to explain that. You can't just like assert it because you've got a good result in one particular situation.

28:04

It's not science, it's cherry picking, essentially, and so there have been number of claims that have been made. There's this claim of emergent behavior. So emergent behavior is things like flocking. So when birds fly in flocks and essentially there's some very simple rules for flocking that birds seem to be following, which is if two neighboring birds are moving this direction, I should move in that direction and that's how. And you can build cellular automata. This is from 30 years ago. Explore this whole idea where you could build these very simple automata, very simple rule sets, and then show that they would exhibit these emergent behaviors like flocking. The only problem with this is that nobody's actually been able to demonstrate emergent behavior. It looks like it might be emergent behavior, but there's actually a really good Stanford paper like how it's not. And so it's this thing where, as humans, we want to see these patterns. It reminds me a lot of Talib's book, fooled by Randomness. He sort of talks about trading and the idea that you could be a successful trader for 10 years on pure luck, and if you're in that situation, you think you're a genius. I'm a genius trader. I've made all these fantastic returns for the last 10 years and it could all just be randomness, and we see the same sort of thing going on.

29:24

It's like GPT is very easy to understand when you see these examples, it's like, oh, this must be. It's very easy to get a leap to the conclusion that something must be happening in there that is intelligent. But again it's like, great, you need to have a theory of where is that intelligence in this process? We do know how it works at the macro level. We understand how it's trained. We understand the perceptrons, the neurons that are in there, how they work, the layers, how they're connected, how it's all of those things. Where is the intelligence in that? And you've got plenty of counter examples.

30:02

And this is made worse by the fact that the more modern large models like GPT-4, are also trained by humans. It learns human answers. So humans will come up with some of the answers that you're getting here and you're like, wow, that's a brilliant answer. And things that GPT-3.5 maybe didn't get quite right. It has been trained on the right answer for those. So a human being has gone and taken that example, written the correct answer, and it has been trained up, fine-tuned on that as part of the process. That's what they call reinforcement learning with human feedback. By the way, it's not reinforcement learning, it's an additional training step where they're fine-tuning the model. So that, to me, is the best explanation of why people think it's intelligent. I don't think it's intelligent because I know how it works.

**Craig Smith Host** 30:48

Yeah, yeah. And where do you think that research is going to go? Do you have any opinion? I mean, either you keep expanding the size of these models and looking for increased capability, or maybe you combine them with reinforcement learning. I mean, there already is some reinforcement learning in there to make it more accurate. Yeah, a lot of people are just using the large language model as a generator of syntactically correct language, but calling a vector database with ground truth knowledge that. Then the language model formulates into coherent responses. And then there's the browsing models that everyone's come out with, so that you're going out to the internet and citing your sources. I mean, I'm curious what your view is on the future of that.

**Mathew Lodge Guest** 32:15

Yeah. So I do think it's going to be. It's not just large language models, I think it is going to be some kind of ensemble approach. Jeff Hinton has this really great quote from actually from the interview that you did in IEEE Spectrum Craig. When he talks about he's like what about all the non-language tasks that the brain does? So he gives basketball, says you learn to play basketball by throwing the ball, so it goes through the hoop. You don't learn to play basketball by reading a book. It's got nothing to do with language. So that whole, that part of learning is completely missing from large language models, and that, I think, is where there is a difference of opinion.

32:53

You see the folks at OpenAI saying these are not the problems you're looking for. They sort of well, yes, languages the root of everything. Well, chomsky argued that language was innate in the brain. That was one of his famous assertions that he's made. Lots of people disagree with that, and so you've got that kind of disagreement going on. In my case, I don't think reinforcement learning is the example of throwing the ball so it goes through the hoop. That's exactly what that is, and so I think the answer is not large language models on their own. The answer is not reinforcement learning. I think it's something else. We don't know what that is, and that's what makes this space fun and exciting.

**Craig Smith Host** 33:36

Yeah, and in terms of applications, do you think that reinforcement learning I mean presumably there are a lot of people working on reinforcement learning applications that are as useful as diff blues application Do you think that? I mean right now we're in a period where there's this tsunami of products coming to market or that will be coming to market in the next year, based on pre-trained transformer models Are there, is there a parallel space where products are being developed based on reinforcement learning, or do you think that there will be if, in fact, those produce more accurate results?

**Mathew Lodge Guest** 34:34

Yeah, I think so. I mean, there already are. O'reilly published a book on reinforcement learning for developers a couple of years ago now, and that sort of gives you an idea of that. So the author of that has a website where he collects examples of reinforcement learning in the real world. The LLM hype machine is in full gear right now. I mean, if you saw Mistral AI, the French AI startup, the people quit Metta and Google four weeks ago and they've raised 100 million euros seed funding. That's the kind of craziness we're in right now, and so there's a lot of noise.

**Craig Smith Host** 35:19

Yeah, but you think that there is. There are other people continuing the work on reinforcement learning for applications? Yeah, absolutely yeah.

**Mathew Lodge Guest** 35:35

So Phil Winder read the O'Reilly book on reinforcement learning a couple of years ago and he maintains a website with lots of examples of reinforcement learning, and you follow his Twitter feed and all kinds of different applications for reinforcement learning. I think the fact that they work so well for a particular problem is what makes them less popular, because they're not as general, and so it's a question of you know what are the things that come to our attention and what are the things that we can relate to, and anything that is easy to relate to is going to be much easier, is going to have a much better job of going viral and attracting attention, versus more specific solutions. I think that's part of the challenge you have with anything that is a more specific solution. I mean, most general audiences don't understand what AI for code products do and don't really understand that, and that's fine. It's not their domain, and the great thing about large language models is that they can write you a haiku or a poem or a funny story, and that's incredibly relatable.

**Craig Smith Host** 36:39

Yeah, I mean you were saying one of the challenges for Diff Blue is just raising awareness, because I was asking if it works so well and people spend so much of their time. Developers spend so much of their time writing unit tests and you're focused on Java right, you guys only do Java, but there's certainly a that's a massive space. I would think it would spread like prairie fire through the Java community. Everyone would say, oh my God, thank God, look, there's this tool that we don't have to write unit tests.

**Mathew Lodge Guest** 37:22

Yeah, Well, there's a lot of skepticism. So your program is, in general, a skeptical lot and they don't like hype, and so they are skeptical. The biggest issue we had prior to GPT coming along is that people didn't believe that our product did what we said it did. That's an incredible claim. How can you possibly claim that? Where's your evidence? You know very skeptical and that's how they are. That kind of goes with the territory really, as a software developer, and so we've seen that in terms of how our process plays out, how we gain customers, first of all, they don't believe we can do it. Then we show them that we can do it and they're like well, that's great, we need to show that it works on my code, and so it's a brand new area. Most developers have never seen a tool like this before, and so they really want to understand what it does and they want to convince themselves, and so we give them, obviously give them every opportunity to do that. We have a community edition. You can just install it into IntelliJ and just try it.

38:30

Back in 2020, we did a Wall Street Journal op-ed. We basically said you know, it was 10 years since Mark Huntryson's software is eating the world op-ed, and we said software is ate the world. But now software is not going to eat itself. This software will write software, and the Wall Street Journal editors were also very skeptical and they gave us a really hard time. We had to work incredibly hard to prove to them that this one was real and two was going to be an important trend in the software world and therefore the business world, because business runs on software and, to their credit, they did run it in the end. But it's like that. People really need to believe that this is something that is worth their time and would make a significant difference. Also, I think frankly, because it's been tried before. So certainly over 20 years ago there was what we call computer-aided software engineering tools case tools that tried to do this kind of stuff and automatically generate code. They did a really bad job. Code was awful, developers hated it, so it kind of got a bad reputation.

**Craig Smith Host** 39:37

Yeah, Again, you know I'm not a coder but I'm fascinated by software and one of the reasons I got excited about these large language models is the prospect that I could write my prompt and the model would write my code and I could build my own software. People have done that at a certain level with GPD4. But again, why isn't there a product where I can write my prompt for a reinforcement learning program that then would go out and search the space and come up with the most viable code and then I could be on my way with whatever app I want to build? Yeah, I believe this application on Steam keeps us gifted in gaving an easy and fulfilling yeah.

40:48

Is there a reason?

**Mathew Lodge Guest** 40:50

Yes, it's really hard for anyone to write down exactly what it is they want their program to do. That's, that's what the specification problem. So this is a very old problem in computer science. This was the whole what the idea of waterfall development in the early days of software was that you know you'd write this full and complete specification for exactly what the software was going to do at the beginning, and then developers could then take that you know fantastic specification without everything in it that they needed to design and, you know, build the software according to the spec. And the problem is, nobody can write the spec. And part of the reason for this is that you're essentially what you're doing is you're asking the people who want the software to do something for them. You're you're essentially asking them to break it down into how it should be done and they don't know. Right, there's and that's why you need developers in the first place, because you know the uses of the product, the people who are describing what it should do. They're experts on their problem. They know what their problem is. They are not experts on how to solve it. They just know that, like, here's the problem, and this is what I want to is what I want to solve. They don't know how it should be solved. That's a very different intellectual exercise to figure out. Okay, so how do you do that? So I mean, so that's the joke about you know, ai generated code from English text. It's like all it requires is a complete and exact specification for that to happen, which means every developer is safe, because that almost never happens. There are exceptions.

42:21

So back to the safety critical world. If you're going to do verification, then you do need to have a very you need a formal specification of what the software should do, and that's why it tends to be used for very small areas of code that are safety critical, right, the real core stuff. What's interesting is, you know, we do a lot of work with Amazon Web Services. They do a lot of formal verification of their code across their code base. For really, you know, mission critical parts of AWS service, like the code that runs on the hypervisor secure boot all of that code is verified right for when the power on the server and all of the microcode that runs before you even start the BIOS or start the operating system All of that stuff is verified at AWS. So it's interesting AWS is somewhat of an outlier in that they're doing that because they're not in the safety critical world, but they see the benefit of formally verifying all that software.

43:17

And so, yeah, I think it's. I think we're for an interesting time. I think you can probably get close. I think people are. I am sure there are companies working on this problem right now and that you'll be able to use that. But, as you say, even today, you know I'm not a programmer anymore. Nobody would pay me to write code. But you know, just for things like running my own website or a project where I just need a little chunk of code to do a specific thing, I can go to chat GPT and it'll give me something that's pretty close. And so for sort of like dabblers like you and me because I would say that you know I'm at the dabbling stage now, I'm not a professional programmer it's an incredible benefit to have that available.

**Craig Smith Host** 44:00

Yeah, for somebody like you who can look at the code and see whether it's correct, semantically correct, and then correct it. For somebody like me who can't do that, yeah, I just end up in endless loops. Yes, what you were describing of the you know writing of the specifications, there's this you know the specificity of the code. It's a whole world of prompt engineering that's developing for working with large language models. I mean, that's really what you're talking about. Right, is prompt engineering being able to write the instructions for an AI model, whether it's an LLM or a reinforcement learning model, to follow?

**Mathew Lodge Guest** 44:53

Yeah, yeah, yeah. I really don't like the term prompt engineering, Craig, because engineering implies there's some kind of rigor there and there isn't. Because, I mean, the main reason that prompt engineering doesn't exist is not a thing is that the fact that small variations in the prompt can produce massive variations in the output and the because it's an unpredictable model. So how, exactly, in what way are you engineering the prompt? You have no predictive ability for you make a change in the prompt. You have no idea what that's going to do to the output. So it's not really engineering. There are good rules of thumb for a prompt and in one shot, systems like large language models, putting examples into the prompt really helps the model give you a better answer, and so by the more context you can give the large language model, the better it will do, and that's really that's the rule of thumb that you need. That's prompt engineering in a nutshell. Just give it as much as you possibly can.

**Craig Smith Host** 45:51

Yeah, and it will be worse than a top. Then you're constrained by the number of tokens you can input. Yes, the but on a reinforcement learning model is, if you can specify your intent with some precision, it could write correct code. So in that sense they're and in order to do that I mean we were talking yesterday I mean, at a certain point if you can specify a natural language that precisely, you might as well be writing code. Yes, but I can imagine that if the programmers may be trained in the future to write in that kind of in natural language, that kind of a spesus, Spesus yeah, I know what you mean. I'm not to edit this. Yeah, too early in the morning. But who are going to be able to write that specifically, that natural language? There could be an orchestration layer that then decides which programming language is most appropriate, or yeah, yeah, yes, but the skill would be in writing specifically in natural language.

**Mathew Lodge Guest** 47:28

Your intent, yes, that's right, and there was a really great Twitter post the other day. Essentially, that's what programming is. It is a way of thinking and a systematic way of thinking, breaking down problems and figuring out how to solve them. That's really what's going on in a program's mind. The typing of the code is sort of incidental to that. And he was responding to some of the Microsoft crazy claims that 61% of Java is now written by Copilot. And it clearly isn't, because the market for Java programmers would have collapsed if that was the case and Microsoft's own statistics. There's another paper from Microsoft Research came out last week and shows the exceptions rate varies quite considerably, from like 0.33 to 0.55 at the maximum level. So I don't know where the 61% figure came from. And his point was like look, saving you time typing is not really what increases a program of productivity. It's that breaking down of the problem and thinking analytically and understanding how to solve it. And so maybe you're right, maybe that's what programming will look like in the future.

48:32

It's like I'm old enough to remember this transition from a sampling language to a higher level languages. You know Algal 68 and Pascal and modular 2 and C and C++ and eventually Java all of those languages and assembly language programmers in the early days. Well, I can write better assembly code. It's tighter, it uses less instructions, it uses less memory All this great thing, but the productivity benefit of writing a high level language is just so overwhelming that very few people have to write assembly these days. Everyone writes in a higher level language and so maybe that's, it's the next level and evolution's the next step up in abstraction.

**Craig Smith Host** 49:08

That's it for this week's episode. I want to thank Matthew for his time. If you want to read a transcript of this conversation, you can find one, as always, on our website. I on AI, that's EYE-ONAI. We love to hear from listeners, so drop us a line and remember the singularity may not be near, but AI is about to change your world, so pay attention.