SDS PODCAST EPISODE 927: AUTOMATING CODE REVIEW WITH AI, FEAT. CODERABBIT'S DAVID LOKER

Jon Krohn:	00:00:00	Welcome to another episode of the SuperDataScience podcast. I'm your host, Jon Krohn. Today I'm delighted to welcome the cool, well-spoken and super intelligent. David Loker is my guest on the show. David is director of AI at CodeRabbit, a bay area based company that dramatically accelerates and improves code reviews through. Yes, of course, ai. In today's extremely interesting episode, David explains how CodeRabbit AI agents work within their platform and the staggering implications of this for how software is created. At the end of the episode, we also dig into why he's impressed by GPT-5 and we go big by opining on how intelligent machines will transform society in the coming decades. This is a great one. Enjoy.
	00:00:44	This episode of Super Data Science is made possible by Anthropic, Dell, Intel and Gurobi.
	00:00:53	David, welcome to the SuperDataScience Podcast. It's a treat to have you here. Where are you calling in from today?
David Loker:	00:00:58	I'm calling in from Los Gatos, California
Jon Krohn:	00:01:01	Los Gatos. And just before we started recording, I tested my tiny bit of Spanish. I thought for sure I'd be wrong because it's a really silly town name. It stands for the cats.
David Loker:	00:01:13	It does. And we have little statues littered throughout the town of cats. Yeah.
Jon Krohn:	00:01:17	That's so funny. Well, we're not here to talk about that. We're actually here to talk about another animal, the CodeRabbit. So earlier this year you joined CodeRabbit, which is an AI driven code review platform, your director of AI there. And the idea behind CodeRabbit is to improve developer productivity by harnessing a Gentech AI to

provide context aware, expert like feedback to those code reviews. David, tell us a bit more about CodeRabbit.

David Loker: 00:01:49 Yeah, so I mean if you think about just sort of the problem in general as a quality control mechanism, prs are there to stop bad code from going into production. And historically speaking, humans do this on mass. More senior people are usually involved in this process. It takes a long time. It's a big part of your job. It's a big part of the enterprise process into making sure that code meets certain standards before it goes into production. And as we get into this new realm of AI generated code and people are producing more and more prs every single day, the human lift of going through all of that and making sure that the quality bar remains high is becoming an increasingly large problem. And so CodeRabbit is there to help with that, right? We're there to speed up the time from PR being opened to being merged.

00:02:44 We're trying to make sure that code that makes it into production is of the highest standards possible, even if you're having all this code being generated by ai. And so that's what CodeRabbit, we're here to solve that problem. We're trying to make sure that people can build things because ultimately that's what engineers want to do. They don't just want to sit here and read through line and line and lines of code and making sure that there are no little bugs and no little op by one errors in your algorithm. They want to build things. And so we're trying to make sure that that sort of more tedious aspect of the engineering tasks while still super important, we're helping as much as we can.

Jon Krohn: 00:03:23 I have no doubt that there's a big opportunity here. A lot of us have experience using Claude or Gemini or ChatGPT for assistance with writing code and those tools do pretty good jobs of giving us feedback on the code that we write. But there's a huge amount of opportunity to build a tool

that is built into your workflow and that is fine tuned to be really excelling at this task. And so we'll talk about all kinds of things in this episode that make that work well. A lot of our listeners on this show are developers, software engineers, but we also have a lot of listeners who are data scientists or AI engineers and they might be working more in a scripting environment, things like Jupiter Notebooks and how does this kind of tool, how does CodeRabbit work in that kind of context?

David Loker: 00:04:20 So at the end of the day, regardless of whether myself am working with Jupiter Notebook, whether I'm doing Python scripts, whether I'm making a machine learning pipeline, all these different aspects, I usually check that code into some sort of platform, usually GitHub in my case, and I'm usually making a pull request. I'm changing this over time. I'm updating my workflow, I'm making changes to my Jupyter Notebook. Now there are lots of things that can go wrong. There's lots of things that I could do better. There's lots of things I can make little mistakes. And so again, I want to have some level of quality control and finding those issues. So I'm not maybe even going down all the way down to the model building level with some mistakes in my data preparation. We all know as sort of machine learning people, data prep is pretty much 80% of your job.

00:05:07 I mean, getting the data from the source, cleaning it up, getting it to the point where I can put it into a model is a lot of work. And making sure that pipeline is reproducible is a big part of this. You want to make sure you can reproduce your results and so you're using source control, you're making sure that those systems are in there and that I can use those. And so still the quality control is super important. Otherwise you're spending maybe potentially millions of dollars training a model on something where the data wasn't what you actually intended it to be because you had some bugs in the

process. And so even for those people, even if it's not just building backend engineering systems in a different sense, quality control is still super important. Finding those bugs early on in the process so we don't end up training a model on bad data. These are all still super important problems to solve.

Jon Krohn: 00:05:59 For sure. Yeah, sounds really useful for our data science oriented listeners out there, but I wanted to have you highlight that for all of them out there. You recently wrote an article for the CodeRabbit blog, which I'll have in the show notes called Pipeline AI versus Age Agentic AI for code reviews, let the Model Reason Within Reason. And in that article you contrast two different AI architecture patterns. So tell me about those two patterns, pipeline AI versus age agentic ai. Probably a lot of, I expect most listeners have now drinking the agent AI Kool-Aid or at least are aware of it, but we could probably do with kind of your definition of age agent AI anyway since it can be so ambiguous. And then contrast that for us with pipeline ai.

David Loker: 00:06:50 Yeah, so I mean just for anyone who doesn't know, obviously Agentic systems, you're giving an LLM access to tools. So rather than being a brain out a jar, it has hands and it can do things. So that's a very crude definition. But ultimately an agentic flow would then allow me to give a command or a desired outcome to a system that uses LLMs and it can go in a loop and it's basically a glorified while loop, right? We're looping around, we're allowing the system to think about what it wants to do, plan out some sort of actions, take actions on tools, get the output from those tools, feed it back in, and then it just cycles and it cycles until it achieves its goal, whatever goal I set for it. And the problem with that is that LLMs do hallucinate and there's error, right? There's variant in the system.

00:07:42 It's not a perfect system, it's not deterministic. And so even if you have a 1% error rate, if you have a chain of actions which goes up to a hundred, you're guaranteed to have an error. I mean at least from a probabilistic perspective. And so that is a problem. Ultimately it's this little error is going, is chaining up. So at the end of the day, the agentic flow can take you to places and take you down paths that you may not have been able to think up yourself. And so it can do a lot of really, really cool things and I don't want to downplay that by any means. It is an ongoing area of research and optimization in terms of reducing the number of tool calls to achieve an outcome is an act of area research that's very much worth pursuing and will get better and better over time.

00:08:35 Our ability to verify those outputs as we go to tame and tamp down on the air to make sure that we are getting where we want to go is also being investigated and these things are going to make forward progress. The pipeline version of that is, I know to some degree which direction this should take. These are the tools that should get run in this order. I know for example, as part of maybe a code review, I'm going to run static analysis tools. I know I have to do that, so I'm just going to make it do that. I'm going to let it choose to do it. I'm just going to do it and I'm going to put the output from that in some form or another into an LLM at a different time. So I'm going to do that for it. And so I might have an entire pipeline that just does that, grabs a bunch of contexts, formats it in some way and gives that to an LLM to generate a final result.

00:09:25 So I don't allow it to go off and do its own thing. That's just purely me directing it entirely, maybe using my domain expertise or I have some sort of flow that I know exists, I'm going to just do it that way. That is very brittle. There's not a lot it can do outside of what I've directed it. And so it's very limited if there's a situation that comes up that doesn't follow that pipeline or needs some sort of

variability in there, it's not really good at that. It's not going to be able to handle that new situation. And so you have those two contrasting approaches. Do I create something that I know is going to do the same thing every time? It's very easy to debug it is got a lot of nice features to it or do I let it just go off and do whatever it wants and maybe it'll run static analysis tools, maybe it won't.

00:10:11 Maybe it'll grab the library definitions from package JSON, maybe it won't and which sort of blend of this do I do? And that's usually where we get into this hybrid approach. And I'd say most production systems lie in the middle. There are things they know should be run like static analysis tools in the case of CodeRabbit with code review. And there are things where we need to allow some level of exploration and that's where you might turn things over to an agentic system like do some web queries because we need to get documentation related to tools that are libraries that are being used that we don't have the documentation for it now currently. Or maybe we need to go and pull files from a different repo because this one is using something of yours that's changed recently. Maybe we need to pull down an issue and understand what exactly what problem are you trying to solve in this moment with this PR and all that stuff isn't always there. It isn't always present. It comes sometimes, sometimes it doesn't. And sometimes you need to pull down multiple pieces and really do an exploration within reason. Again, you might put some boundaries on that. I'm allowed to do this many tool calls, I'm allowed to iterate this many times, I'm allowed to build up this much context before I stop. So guardrails are still super important in that context and that's what we do. We use a hybrid approach to build a context window that we think is optimized for the code review process.

Jon Krohn: 00:11:36 Beautiful. I think I understand all this now, it reminds me it might not be exactly the same and you can kind of

correct me on this, but late last year Anthropic published a very popular blog post kind of defining different kinds of agentic systems and I think they might've called what you call pipeline AI a workflow. And so they basically define that there are two different kinds of agentic systems workflows and real agents. And it seems like it's a pretty similar, if not the same definition to you where in the workflows like your pipeline ai, we are defining a particular path that they flow down and you are limiting to a great extent what an agent can be doing which might bring you feelings of more safety, more security, more control, but also you're limiting the potential for what could be happening in an application. And so it makes a lot of sense to me that you're doing this hybrid approach between the two and trying to get the best of both worlds. And that ends up creating what you described as the code reviews then feel like they came from your best engineer on their best day every time is the way that you phrased it.

David Loker: 00:12:49 Yes, I think that's the goal of any of these systems we want to make sure that we're building. So if you think about it just from a large perspective, I give you a PR to review and I don't give you half of the files in the PR or I don't give you the issue that it came from or I don't give you the documentation related to standards or you don't have any context of the code base at large, you're probably going to not do a great job even if you're a senior, even if you're a staff principal engineer, it doesn't matter if you don't have context, you can't do your job. So for us, I think the toy, the term is now coined, context engineering has been coined. I don't know when exactly that happened, but it happened fairly recently and the reality is most systems I think have been doing this right?

00:13:40 Very few systems are thin prompts with a few one-shot examples that then you just interact with. Most systems of any real value are doing some level of rag or some level

of context gathering in order to make sure that the information that goes into the LM to make that final output has what it needs. And the main thing that we're trying to tamp down is you also want to make sure it doesn't have anything else. And that's a tricky part because you can just shove stuff in there and especially as we get larger and larger context windows, it is tempting to just say, I'm just going to throw everything in there. Imagine we have 1 million, 2 million, 10 million contact size windows. There are other issues there. There's the ability for the LLM to actually pay attention to that entire context window. It can't pay attention to it equally, it tends to have a U-shaped pattern where it pays attention to the beginning and the end far more than the middle. And there have been multiple studies at this point that show that when you put in information, even if it's accurate, the LLM performance will degrade on a task the more you put in there. So you don't want just arbitrary things in there, even if they're factually accurate. And so this context engineering is extremely important.

Jon Krohn: 00:14:55 I'm glad that you brought up context engineering because that is actually the next topic that I had, but really quickly before we get to context engineering in a bit more detail, it occurs to me that there is a term that we've been using a lot and I know that our listeners are a technical audience and so for most people they are going to know what a PR is, but since we're using that term so much in this episode, we should probably define it. David, do you want to tell us about pull requests?

David Loker: 00:15:20 Yes. So yes, prs, what are they? Yeah, pull request is the idea that in a get platform specifically, if I am going to make a change and I'm ready to potentially release that change or at least have that change reviewed by somebody, I'm going to try and merge my branch, which has this feature say which a bunch of new code, a bunch of changes to code, I'm going to push that up somewhere

so that someone can look at it. And so that pull request is basically me saying, okay, I'm requesting that somebody look at this. I want to merge it from my branch into say the main branch or dev branch to sort of eventually roll out to production and it's going to show you all the different lines that are changed, all the additions. I'm going to have a summary in there. I'm going to have a bunch of information that I've put in and so someone can go in and sort of decide are there problems, should I do this differently and make comments?

Jon Krohn: 00:16:11 That was a great definition. There's something really funny in the way that you described it that I'm going to dig into a little bit more, which is that before the pandemic I was teaching an intro to deep learning course in person in New York and somebody who was new to it said to me, can you explain this pull request thing to me? It seems like it's actually a push and you actually use the word push in your definition are you kind of pushing code up? So what's up with that? Why is it called a pull request? Not a push request?

David Loker: 00:16:43 Yeah, that's an interesting term. I think the flow of it, if you think of it, the reason why I use that term is because when the commands that I enter is I'm just so used to it, I push my

00:16:53 Branch up to the cloud in this case. And so that's the pushing aspect of it. And I think pull requests you're talking about you're pulling your code from one branch to another. It's kind of an antiquated term I guess at this particular point, but it is kind stuck around. You're not really pulling anything per se. But yeah, I'm essentially, if I'm thinking of myself then as Maine let's say, or dev, I'm then pulling code from a different branch into my branch I guess. But yeah, merge emerge is another way to think of it. I guess it's just eventually that's the final command that gets run.

Jon Krohn: 00:17:24 Yeah, yeah. Anyway, thank you for indulging me with both my pull and push request questions. Let's talk more about context engineering, which is it's one of the biggest, most talked about topics in AI today. Ed Donner who's someone that I've done agentic AI workshops alongside, we were putting together another workshop recently and he said, we've got to have a big section on context engineering pretty early on. This is the new big thing that everyone's talking about. And so the CodeRabbit website describes context engineering in your case as feeding LLMs not just with code but also with intent dependencies and linked issues. Tell us more about this context engineering and what it means for you at CodeRabbit in particular.

David Loker: 00:18:15 So the context in this case, if I just want to give a brief definition of what I mean by that is the information that I'm eventually going to feed into an LLM to then decide the answer to my query. In this case, let's figure out what the bugs are, let's give me some comments about this pull request. And in that case, imagine I just give it the, that context is there and it might give you some interesting insights and from a certain perspective, very low level perspective. But if I don't understand what this PR is about, if I have no description as a human and I'm looking at it, it's going to take me a while to try and reason out what was this person trying to do and how does this fit into the larger picture of this code base? And so the context window, we need to start adding more information.

00:19:06 That context engineering is how you go about doing that. How do I decide what information I should get from where, and then when I grab that information, if it has too much, how do I decide what I need to do to that to get what is necessary into that context window, which is the context for the LLM. And so that code, the engineering task about how you do that, there's lots of different things

you could do, right? There's lots of different areas that I could grab information from. I could get you pointed out the issue, I can go and I can get the description of the issue and it could potentially lay out an entire PRD or product requirement and say, this needs to implement the following things, or maybe it's a bug and here's the bug, here's the output of the bug and this is where it came up.

00:19:53 It might be an image in there and stuff like that. I need to potentially bring this information to understand the context of this code to better look at it and say, oh, this was supposed to do X, but it's kind of doing a little bit different than what it was intended to do. And in some cases maybe there was a decision made that changed that and maybe there's other contexts somewhere else like a notion document related to a design of what this PR is supposed to do that I could pull in where I'd be able to say this was the decision that was made after this issue was formed and the issue might actually link to that and I can pull that information in too. And then it comes down to, okay, now I need to understand where does this PR sit? Where does the code that I've changed sit in a relationship to the rest of the code base and is there information I need from there?

00:20:40 So for example, if I change the a I of a public facing function that gets used elsewhere, I need to know that where it gets used and in what context that gets used. I may have just broke something. It's not even in the pr, it has nothing to do with this. I need to understand that that gets used elsewhere. I need to bring that in order to make sure that the LLM knows that. And so we do something called a code graph, which we connect pieces up together to be able to build an understanding of how the pieces of your code interact so that we can do that and we can bring that information in. We also have verification loops which go in and can search the code

base to understand where are these pieces coming from, how do I find the relevant code snippets that might exist elsewhere so that I can bring in that context and understand to a large degree what is happening inside this pr, bringing all that information in. If I was to give that to a human, they would finally go, okay, I get it now. Right? This makes sense to me now I don't have to think about it as much, but that information needs to be there.

Jon Krohn: 00:21:41 Awesome. This sounds really powerful to be able to have all that extra context. And it sounds like it enables your tool, it enables CodeRabbit to go from a what to kind of a why behind a code change, which would be invaluable in code reviews allow us to catch more bugs and maybe move away from just catching bugs to shaping better design decisions for the code, for the whole code database.

David Loker: 00:22:15 So there's a few things there when it comes to finding bugs. I was talking about the API example, you could find a bug there. There could also be things that we've discovered over time in the way that you typically do something like the way authentication is handled, where that's a higher level piece of information that might be the way that you've done it doesn't align with the standards documentation that you have. This is the way you should handle user information. You're doing things not quite the right way or maybe there's a way that you are supposed to deal with secrets and with API keys and all these other things and you don't do things quite right, but that documentation that we've pulled in says that this is the standard, this is where things live and you've changed that. We need to think about fixing that. And we have things called refactor suggestions, which typically are around those things which are not necessarily bugs specifically, but could be in the way that you've architected your solution. And that could come from other things. Learnings is an example. If you are chatting

with a human and human says, we don't do things this way, we do this, CodeRabbit could pick up on that and essentially bring that information and store that for future use so that we make sure that someone else doesn't make that same mistake. Right?

Jon Krohn: 00:23:23 Very cool. I like that a lot. It sounds like this extra context, all this context engineering might also be helpful in reducing hallucinations.

David Loker: 00:23:32 So being able to back onto something. So rather than letting the LLM fill in the missing details, which is where you start to get hallucinations, it fills in sort of from the mean or the median of its information and makes assumptions. We provide it with this context and then with that context we can do a verification. Does this comment actually reference something that's real? Does it have evidence to back it up? And that verification allows us to significantly prune out hallucinations.

Jon Krohn: 00:24:03 Love it. Another great reason for people to be exploring context engineering for whatever application they're involved with. Let's move on to another really important part of agents, which is tools. So you've pointed to research in blog posts about tools like Retool and I don't know if I'm pronouncing this right, Leret.

David Loker: 00:24:26 I don't know the pronunciation of that one either. So yeah, I'll let you take a stab at that.

Jon Krohn: 00:24:31 Well, I can spell it. L-E-R-E-T. And so retool loret, they show agents can be trained to call tools more intelligently. So tell us about this, these kinds of research advancements and what it would take to build the kind of high quality domain specific data sets that could actually teach agents to reason like experts in code reviews taking advantage of these tools.

David Loker: 00:24:57 So this has actually been a big area of research, not just from the open source community, but in general it's been a large effort both for philanthropic and OpenAI as part of this recent wave of age agentic systems. So even in chat GBT now, you can hit that little button and give it access to the web or give it access to other sources of data. And the way that they do this is they want to make sure that one, which tools are available, but also you're calling those tools correctly. This is again, and this is an LLM, this is not me programming in an API call. This is a system trying to design the API call on the fly. And again, there are errors in that, right? And so you can use reinforcement learning to bias it towards correct understanding of how you build an API call in this case an MCP call for example, or using a tool.

00:25:51 And you can also use these techniques. So imagine I have an outcome that I want and I have access to let's say read, write a bunch of different list commands and things like that. And I'm trying to find something on your computer. There are lots of different avenues that could lead to the right answer. If I want to find in a bunch of logs inside some large system, the logs which have specific values, I could just do a grab, right? Do that's a simple tool call. I could also list the files, go through the one by one cat everything out and start repping the cat. And so there's efficient ways of doing stuff and there are inefficient ways both of them lead to the right outcome, but one of them wasted a lot of tokens and wasted a lot of time. And so you have a lot of opportunity there to not only train how to use the tools, but maybe how to use the minimum number of tools required to get the answer that I need.

00:26:46 And so that's a really important area of exploration right now as we've realized that giving agents tools is necessary for them to do anything really meaningful to be able to interact with the world, to interact with us, to collect

information. All this stuff is we need tools and we don't want to tell them like a pipeline system. We don't want to tell them exactly what to do. We want them to decide what to do, in which case they need to get good at it, they need to figure out what I should call, when I should call it, how do I use it and what do I do after I get that information? What do I do with that? And so all this stuff retool, we pronounced it L-E-R-E-T, both of those are avenues that are investigating that optimization problem. I think we need to keep going down that route. We need these things to get better and better. So again, we don't pay as much money as we're trying to use these tools.

Jon Krohn: 00:27:40 I looked up the LT paper while you were speaking and so I'll have that for sure in the show notes as well as the link to retool. And I do think LT is correct because it's an abbreviation of language empowered retentive network, so presumably lore and also of course it's a play on Lynette from Jan Laun. So there you go. I mean it has to be, it can't be a coincidence. And then also really important question for you here, David, when you're catting out a file in your part of the world, you call it gatting out

David Loker: 00:28:19 In Los Gatos, if you don't do that, they catch you. That's the real issue.

Jon Krohn: 00:28:23 Exactly. So onto the next question, the CodeRabit website features a case study for something called plane like airplane of all the different homonyms or homophones out there I guess for plane. So plane is a project management platform and they share improvements with developer productivity. If AI handles more of the repetitive review cycles, including things like tedious refactorings, what new dimensions of productivity should organizations start measuring beyond lines of code, which has always been an annoying one for me. Velocity bug counts. How can we be measuring productivity if we have machines generating most of our lines of code,

determining most of our velocity and handling most of our bugs?

David Loker: 00:29:16 Yeah, I think it's an interesting problem. Ultimately in terms of return on investment of tools that get used, there's a lot of different things that you could measure or one is just satisfaction overall of developers, but if you're measuring developer productivity, you cannot use lines of code at this particular point. If you're involving AI generated code tools into their workflows, you're allowing that, it just becomes meaningless. Even number of PRS opened is not necessarily indicative. I think we're going to get started to get into this realm of how do you measure, for example, the productivity of someone who's in the data science machine learning area. For me, when I was working at Netflix and stuff like this, it was more around did we have a hypothesis execute on that hypothesis and get a result? And so that was our measurement because of the fact that the number of lines generated in this example don't necessarily lead to a good outcome.

00:30:13 And it's really hard to figure out whether or not somebody's doing something when it's lines of code. So it's going to get down to I think the idea of features. Are you building things? Are you coming up with those ideas and executing on them? Because at the end of the day, prototyping, rapid prototyping is going to become increasingly and increasingly a solved problem. And so are you bringing that? Do you have an idea? Are you building that thing in a prototype and then are you then bringing that to production? So that's going to be the thing. It's not going to be about code lines being generated and as we get better and better with our AI code review, we're going to be preventing then those big bugs from going out and we're going to be preventing long-term bugs from happening. So we can't measure the same degree we used to the number of bugs in the platform, we could measure that before, but as we do

this, that's not going to be an issue. And so it's going to be this creativity, I think, and this ability to bring things out into the world. So that level of velocity becomes a little bit different.

Jon Krohn: 00:31:16 As all of your answers have been. You're an outstanding podcast guest. It's great having you have great answers, lots of information. No umming and aing. Pretty impressive so far in this episode, we've largely been talking about software development in general. I'd like to move towards some more AI engineering specific questions now for a bit, which builds on your experience as an AI leader, for example, as director of AI at codera. So CodeRabbit provides flexibility to both large enterprises, small teams supporting self-hosted deployments, multiple LLM providers, and highlighting that companies can stay in control of their AI infrastructure while maintaining full data privacy. How should somebody who's responsible for the AI in an organization like you are resolve this kind of tension between a bring your own AI culture, which boosts individual developer productivity relative to a coherent governable AI dev tool stack that ensures enterprise wide consistency, security control?

00:32:26 It's kind of interesting, actually. There was, I recently read a stat that something like 90% of people, and this is going beyond people just working on in ai, but in enterprises in general, 90% of employees use a personal ChatGPT subscription or maybe not even as a subscription, they're just putting it into the free tool and using that augmenting themselves without there being any kind of official support in the organization for ChatGPT. So yeah, I mean that's kind of giving generally some kind of stats on how common this is and I'm sure the same kind of thing is happening with AI engineering people, whether you're aware of it or not throwing things into Claude code and getting great results, but that might

not meet the security requirements of your organization. So yeah, how do you resolve this tension?

David Loker: 00:33:15 Yeah, so for us, we have zero data retention policies with the organizations and then we promote people to use then the systems internally. So we have API keys and we have subscriptions to these systems that we give to our people so that they can have access to the latest tools and do as much work as they need to with them and leverage them in the best way possible. I do think there's a learning curve when it comes to using these systems, so having them there so they can interact with them and figure out which ones work and which ones don't. Also ends up leading, I think to better outcomes in our own tools as we're thinking about the prompts that we engineer and the way that we interact with LLMs, the experience that you gain from using them in all these different aspects of your work actually I think really, really help building the product itself.

00:34:04 And when I think about how people would think about using CodeRabbit and how we are very, very protective of people's data. I mean we're talking about code here, this is people's ip, it's the bread and butter, it's the soul of their company. And at the end of the day, we're very, very careful around that kind of stuff. So everything gets spun up exactly once during the code review into a completely isolated environment in sandbox and gets immediately torn down the second that it's done. And we don't store any of that code there. And the reason we don't do that is because we understand the level of importance there and getting the compliance that we have, having that zero data retention policy with the various LLM providers. That way people who use our system know that none of that code that goes through there, none of the documentation that we pull in for context engineering, none of that stuff is going to end up inside someone's system and is going to be stored there.

00:34:57 It's all gone. And so they don't have to worry about their stuff being leaked out through these LLMs. And I think having that, if your developers are using AI and they should be to a degree, I think at this point in time you need to take a step forward and be like, I'm going to embrace this. I'm going to make sure that I have these things in place with the LLM providers that my developers need to use and want to use to make themselves more productive. And I'm going to have those policies in place of this is making sure that none of that data gets leaked out. If for whatever reason your system requires even more stringent guidelines and using LLMs maybe on your own infrastructure, provide that. So take some time to get that set up so that people can then use the same underlying ideas behind making themselves more productive, asking LLMs to do certain things for them and it remains entirely within the boundaries of your own infrastructure. So that's sort of something from our self-hosted perspective that CodeRabbit does. We allow you to go on and install it on your own infrastructure if that is a requirement, but our SaaS system is extremely secure and none of your code will leak out into any of these outside systems and nothing will ever get stored there.

Jon Krohn: 00:36:17 Nice. Great answer. Another tricky thing that comes up for a lot of us developing AI systems is how to allow our AI systems to learn from user behavior but still feel confident that as a result of that learning, as the AI systems have more power to learn to do things themselves based on some context or based on interactions with a particular user, it's harder and harder to test that. And so for example, CodeRabbit, one of its features is Agentic Chat, which is not only reviewing code but also generating tests, creating issues, resolving feedback. And this incorporates learning from conversation from chat with the users. And so developers are completely effortlessly teaching the system what kind

of feedback they want, but for all of us designing AI systems, this can be a scary thing because we can't test that particular version of the model that's now learned these preferences. So how do you handle those challenges and what are the opportunities?

David Loker: 00:37:35 We listen to customers a lot, so we talk to a lot of customers all the time and we are ultimately receiving a lot of feedback all the time. And when it comes to things that we can do, we listen to them and we also watch open source and see what happens there. So there's a lot of open source. Also at the end of the day, we use our own product, so we've interacted with the learning system, we've seen where it works and where it breaks down and we've tweaked it and we are constantly using our own system in all the different ways that we have. We are integrating, we're an MCP client now, so we've hooked up our MCP servers, we've seen the exact details of how it gets used, the learning system, we use that all the time. If something comes out and sort of a library mismatch happens, we're like at Code Revit, you should know this library is being used, remember this.

00:38:26 And then the learning get added and we'll see how that impacts future reviews. So we use our own system a lot. We watch open source, we get feedback and we're constantly just making sure that we are listening to the customer. And I think that's the super, at the end of the day, that's what we're trying to do. We're trying to provide value to the customer and if we're not listening to them, we're not doing our jobs. And so I think the be all end of it is that when you're not in a place where you can just take that information and use reinforcement learning, we're not going to be able to do that In this case, it makes perfect sense. We don't want to store people's code so we're not training on people's code and that's a trade off that we're willing to make to ensure that people's security and people feel like they can trust what's happening.

Jon Krohn: 00:39:13 Nice. Speaking of security, a big complaint that I see so much in social media around using Gen AI for code generation specifically, but you can see how that ties pretty closely to what we're doing here. We have code reviews happening with gen AI systems and agent systems. One of the big complaints is people will say, oh, it's not using best practices all the time. There's all kinds of security holes that end up getting picked up from Stack Overflow just by spitting out some result that works, but has all kinds of security holes in it. I come across this all the time and it seems to be, it's especially one of the things that as we've gone from GPT two to three, to four to five and the code generation capabilities have become more and more threatening to software engineers, it seems like I'm seeing this kind of like, oh, well obviously when it's GT two, the code is so bad, there's no threat.

David Loker: 00:40:22 Yes, that's true.

Jon Krohn: 00:40:23 And then GPT-4, you're starting to see, okay, well there's this wide range of things that this generative tool can do, but look at all these places where you still absolutely need a human in the loop. I can't be replaced. And now that we're kind of at G BT five, the security thing comes up a lot. I don't really buy it and I wonder if you have any thoughts on that in particular, CodeRabbit emphasizes reducing alert fatigue by providing actionable, prioritized security insights. So yeah, it seems like CodeRabbit has kind of caught onto what I see is that actually machines can be way more vigilant than humans in spotting issues and could probably create a more secure system than a human anyway.

David Loker: 00:41:07 Yes, I agree with that statement because of the fact that machines don't need sleep, they don't need food, they don't lose attention, they just sit there and they stare at this thing and they're just going to keep staring at it until they find whatever that they need to find. And the more

we teach them, and the more we get better at this from a context engineering perspective, the more and more unlikely it is that a human's going to find some security issue that we missed. And I think what people are coming from when they talk about code generators learning from Stack Overflow and some security issue, is the assumption essentially that the training data gets replicated and to a certain degree there is, we have to understand that these are probabilistic machines and so at the end of the day, they are picking and choosing things based on what they see very frequently and they're trying to mold that into the surrounding context of whatever your code is right now so they can output things that are novel.

00:42:07 They do output things that are novel. They're not a database, it's a probabilistic machine the same way that our brains are probabilistic machines. Now, will they make mistakes? Yes. That's why we need things like Code Revit. They're going to make mistakes. They're getting better and better all the time because I can take that initially trained probabilistic machine and I can do a lot of stuff to it after the fact. I can make sure that when I output code, I run it through some system looking for security issues and if it finds it, I can rate that low and one that didn't have that problem, I can rate up and guess what, this reinforcement learning technique over time is going to remove these issues and they're putting a lot of effort into this, a lot of effort, a lot of money, a lot of human effort into this process of labeling and getting this feedback and iterating on it.

00:42:56 These systems are going to get to the point where they're significantly better than people at most of these tasks. My hope is, I watched this talk, I think it was about a month maybe two ago from Andrew ing where he said he brought up a really interesting point, okay, coding has shifted dramatically from the seventies. You think about going

back punch cards and everybody's like, okay, this is very tedious. There are very few programmers at that particular point in time, and then we go into symbolic computing. You're talking about doing things like just doing machine level code, right? Again, super tedious compared to what we do now. More programmers came around, but it's significantly easier. People are doing punch cards, this is way too easy. Then you get things like cobol, right? And then all right, now it's way easier. People can do this high level representational language to be able to get things done on a machine and the people who used to code in machine language are like, this is way too easy.

00:43:52 These are not coders we're coders. It constantly has this progression. You get simpler and simpler, higher and higher order languages and you get not less coders, you get more engineers. So we just changed the definition of what being a software developer actually is. And so I think we need to take a little bit of a step back and it's a frightening moment. I get it. I really do. I get it. But if we take a step back and we think, what is this going to do? Most likely it's going to allow a lot of people who previously would never have engaged with the idea of building software to suddenly engage with building software. And so if we allow for that, if we allow for that to expand our definition of what it means to be a software developer, if we just allow for that for a moment and we let these people stumble through into this new world, we get to greatly expand the amount of things that are going to come out.

00:44:45 The imagination that we get to now engage with through software is going to be greatly expanded and I think we will benefit from that as a society, as other software engineers. We are now going to be engaging with this on a deeper level, and I think we are going to see people move towards over the next five, 10 years to can I talk to an AI

system in a way that leads to the outcome that I want? And we still might need the understanding of large scale systems and when this gets deployed, I need to make sure because do I use Kubernetes? Do I use Cloud Run? Do I use Redis as a cache in this instance, do I not? Some of these questions, there's multiple right answers, and choosing those can be difficult and maybe those expertise levels will stick around a little bit longer. But I do think this is a good thing, generally speaking,

Jon Krohn: 00:45:36 Yeah, we stand more and more on the shoulders of more and more giants and we abstract away more and more of the complexity. I'm glad that I don't have to be worried about punch cards for example. And I have a long question now that listeners are going to have to bear with me on this. I guess it gives you a chance to get some water or something while I go through this long question, but a lot of what you've been saying leads us into this discussion of vibe coding. So Vibe Coding was coined I guess about a year ago by Andre Carpathy. I can actually look up the exact tweet. So it was February, 2025. This year he popularized this term vibe coating to refer to coding entirely via prompt and basically forgetting that the code even exists underneath. So kind of another layer of abstraction on top of all the layers that you've been talking about, punch cards, cobalt machine code, and since then the term has been embraced by some people.

00:46:47 So many technologists, investors, media figures have embraced the term. Companies like Lovable, which allow you to go from a prompt to a working application has reached a nearly $2 billion valuation recently. Meanwhile, there are other folks who think that vibe coding think of it only as a negative term and are hesitant to jump on the bandwagon. Maybe this kind of relates to my security concerns question that maybe it's the same kind of people. This probably an overlap in the Venn diagram, but you've even noted in the CodeRabbit blog how studies

find that AI tools, AI coding tools can add up to 41% more bugs to your code. Yeah. So if AI generated code quality is so poor and Vibe coders aren't really engaged in the specifics of the creation of that code, wouldn't the code reviewed process be one of ai? So if you have one AI system generating code and then another AI system reviewing code, how do you see this kind of future playing out? What are the dynamics really like? What are the positives of that scenario and what are some of the risks?

David Loker: 00:48:06 Yeah, I think this is an interesting point in time. I mean, we can't deny that This is one of the more interesting points in time. Even if you hate the idea of vibe coding and you hate this idea of prompt only engineering, this is unique and we're not going to be seeing we're going to be in this point for a period of time and then suddenly things are going to shift again. So let's enjoy this to a degree. It's interesting. My sister-in-law built an app using Lovable in the afternoon, and ultimately it's purely based on an idea of something she wanted to build. She has no idea about anything coding related. Yes, vibe coding. I think maybe the term has had a negative impact on the overall progression of what could have been if maybe this term wasn't there for people to latch onto and have this sort of negative feedback towards.

00:48:51 But the idea behind being able to use AI to create something without knowledge that I have personally is an amazing thing. And so we need to take it for what it is. Does it introduce tons of problems? A hundred percent. That app that my sister-in-law generated was not production ready, it was a prototype. And prototyping with these systems is extremely valuable. As we all know. It takes usually a long time to get to a prototype. There's a lot of effort that goes into that just to check whether the idea is worth anything before I maybe spend a ton of time and a ton of money putting something out and making it production ready. Maybe I should test it first. And this is

kind of an entrepreneurship's dream in a way. You want to be able to test things and fail fast and iterate, listen to people iterate again.

00:49:40 And once you get something, then spend some effort making the choices that you need to make it production ready to get rid of all those bugs, make sure this thing is bulletproof or whatever you want to say. But we still need that initial period and that still brings in more people and creates more jobs and makes the need for software developers even more. The only reason why I don't like the term vibe coding as much is because it makes people think that the only way to use these systems is through that mechanism. I'm just going to prompt not going to care about the code. You can do really deep and meaningful work with AI generated code and you can be an engineer and looking at that code and making sure that the decisions, the high level decisions being made are accurate and checking it over and making sure that why are you doing this?

00:50:30 Why are you doing scope creep? That happens a lot to me with cloud code. The scope creep is real. And so you can engage with these systems in a very, very deep level that's very intellectually engaging and still requires a lot of software engineering prowess, but you can prototype with them just in the vibe co way. And that is very, very valuable. And that's why I think a lot of VCs have been latching onto it. And a lot of people are really excited. The idea, for example, of a non-engineer coming in and creating an app that skyrockets that was never even remotely possible at a certain point without a lot of effort from that person. So just think about again, the amount of human imagination that we're bringing to bear into our world, the number of things we could potentially be interacting with at some point that are just would maybe never have come to be. I still think that that's the reason to be excited about it and to ultimately embrace it.

Jon Krohn: 00:51:29 It's an exciting time for sure where we are reducing over just a few years. We've seen you talk about effort there. Going from some app idea to having a working prototype, we've reduced by thousands of times, maybe millions of times, the amount of time and effort required to get that up and running. It's crazy.

David Loker: 00:51:50 And this is ultimately CodeRabbit fits into this picture. You go from lovable generation, you put CodeRabbit in there, and you're making sure that your system at least has this bug checking going on. So if you don't understand what's happening, it can point things out to you. You feed that back into your AI system and it can fix some things, right? And so our system fits into this new model really well. And I think the reason why we are doing as well as we are and why our system is becoming more and more of just a standard requirement is because of the level of people that are now engaging with software engineering. They just need this. They don't necessarily have a whole team behind them either, right? They have one person vibe coding an app. Somebody's got to look at your code. And if you look at your own code, as we know, reviewing your own code is the biggest fo PA in software engineering. So having at least some level of third party review, I think our tool is absolutely essential as we move forward into this new sort of world.

Jon Krohn: 00:52:47 I agree with you on everything you've said and it was really interesting and entertaining and kind of gives us a bit of the, it brings back to me some of the awe around what we're going through. I think it's easy to feel stressed and intimidated by all that's going on, but really you can kind of reinterpret that same emotion as awe and excitement. Speaking of awe and excitement, I like you, I'm a big fan of GPT-5. I think that it's an impressive leap forward. So I did an episode on GPT-5 recently, episode number 916, and it, it's a pretty positive review. A lot of people have been pretty negative about the GPT-5 release.

The cover story on The Economist the week that you and I are recording is about how LLMs are plateauing and what does this mean for the trillions of dollars that have flowed into the stock market and are being invested in AI systems.

00:53:51 The Economist article is talking about how there's no longer so much hype around God-like AI systems coming around, and I don't really get how GPT five has changed so many opinions around this because your own benchmarking, so on the CodeRabbit blog, you have a post which I'll have in the show notes called Benchmarking GPT-5, why it's a generational leap in reasoning. And you described G PT five in that article is delivering this significant leap forward. So what in your view is so great about G PT five and how is it a big leap forward and not just the kind of minor incremental gain or even step backward that so many critics have come up with?

David Loker: 00:54:39 So when I'm reviewing that, I'm reviewing it right from our perspective in our particular use case. And we use the reasoning aspect of it pretty heavily when it comes to the code review process that system is that ability to reason through is extremely important when it comes to trying to find errors that might cross a lot of different pieces of the context window. So the reason why I call it generational leap is because the gains in our evaluations were so significant that I hadn't anticipated it. So the system that we go through every time we're evaluating prompting and we're evaluating new models, we're trying out new ideas, we have a set of prs of pull requests that are designed to be difficult. And I was getting, when I first got a join CodeRabbit, we were getting maybe five or six of those really hard prs correct at that particular point in time.

00:55:36 And we've been steadily increasing that by doing better context engineering and doing better things with where

we're pulling data from and how we're doing our prompting and this and all kinds of different things. And so we get that up to maybe 10, right? And 10 things that we're getting correct. Out of 25 of these hardest prs, we have many more in terms of overall, but these are the hardest set that we could come up with and we sort of pruned it down to. And then we see Opus Opus come out and Sona four come out and it goes up to about 12. So this is a pretty big jump. And then we go to G PT five and we goes up to anywhere. But depending on, again, non-determinism goes between 18 to 21, correct out of this 25. And that's a massive jump. I mean, I can't deny that that's a massive jump.

00:56:20 And so I see this and we start going like, okay, something wrong. That's our first interpretation. Is it something's wrong, right? We're not getting this right. And so we do a deep dive into it and ultimately we see some really interesting things when it comes to the way that it reasons and the way that it follows the path of code through multiple files across many, many different lines, pulling in context in different places. And really I saw some things. So I taught in college logic as a class when I was doing grad school, and I saw some things in there that human beings have a hard time doing when it comes to logical thought of you have a set of axioms and I'm following these logical rules to reach a conclusion. And I saw some things in there that I thought were interesting that I hadn't seen from other models before.

00:57:12 Being able to use a negative in a chain of thought I hadn't seen before. So the fact that this thing over here doesn't exist implies the following things to happen as a result of multiple levels of logic deep. And I was like, that is really cool. That was something that I thought stood out to me is almost right away with some of the examples. But also just being able to say, if I assume that you fix the bug that I reported earlier, then the following things have to

happen again. That sort of multi-layered assumption, changing your worldview by shifting your axioms is a difficult things even for a human being to do. And so I saw this happen again and again and again, and that's where I started to be like, this is really interesting. And I just saw things that I didn't see in other systems. The hallucination rate also dropped dramatically. The negative sentiment that we got in response to this dropped dramatically. And so all these things combined and I was just like, for us, this is a massive leap forward.

Jon Krohn: 00:58:15 I think that part of why people didn't have such the shift from GPT three to GPT four that allowed machines to go from being able to tackle tasks that took humans maybe like 10 seconds to tasks that took that take humans minutes and that for a huge number of simple queries that you could just put into a chat on regular day-to-day questions. I think that that felt like a really big leap GPT four to GPT five, I think it might not have felt to a lot of people like such a big leap if they're continuing to ask those same kinds of questions that take seconds or minutes for a human to answer. But as we get into the kinds of things that CodeRabbit is doing where you have these complex code reviews that could take hours, GPT five is powerful there in a way that GPT four,

David Loker: 00:59:06 Right? Yeah. So you do have to ask it much more complicated things if you're going to see that lift. I do think that outside of the code review, massive gain that we saw, I do think that LLMs are going to reach their height. I do think that architecturally we're going to have to come up with different ideas, not even just from a reasoning long-term perspective. How far can this go? I do think we're going to hit that. We don't have infinite data, the efficiency, there's lots of different things we could do to make it better. One is just our brains are super efficient. Just think about the stuff that we do with the minimum amount of energy and then the amount of

energy required to run LLMs. There's a reason there's something missing there, right?

Jon Krohn: 00:59:51 Training dataset sizes, an infant child learns or makes an inference with even just a whiff of an example, just an illusion to what could be happening here in terms of the real world, world physics or human intentions. Whereas the LLM is requiring millions of examples to get to the same conclusion. So vast differences in energy and data requirements for sure.

David Loker: 01:00:13 Yes, exactly. And so I think there is a different framework that we need to discover. And so some of that is going to require tremendous amounts of very smart people to think away from LLMs to spend their cycles thinking outside the box. And so do I feel like the money is wasted? No. We still have a long ways to go. We still have lots of things that we can do from a vertical perspective. We still have an exhausted fine tuning models on specific verticals, coding being one of them, code review being another. There's lots of verticals that we can still use the current architecture and really take it still a lot farther. And we have to have people thinking at these new things. We are going to reach a limit in terms of what LLMs can do. And there's no reason to stop at LLMs.

01:01:06 There's no reason. I'm sure I'm a hundred percent positively sure that there's a lot of very intelligent people thinking about new architectures. Why is the brain more efficient? How do I architect this in different way? How do I make better use of the energy being easier to make sure that I'm doing the right thing? All these problems need to be solved and people are working on them. And we'll see when that next thing happens, right? And it will be sudden and we'll suddenly be like, oh, I guess we're in a new realm again. And it's going to again be very exciting. And I think I look forward to it.

Jon Krohn: 01:01:39 Yeah, no question. Big, big, big things ahead. Even if we run into temporary plateaus or if it feels like that looking over many years, decades, crazy, crazy things are going to happen in terms of machine intelligence, no question. I want to get into one last kind of long question or topic area before we start wrapping up. And so before CodeRabbit, your career in software and machine learning span several decades. You've had roles doing supply chain optimization at Amazon. You used to automate testing at Rim research in motion maker of the Blackberry and now called Blackberry. So the Blackberry phone. And actually something interesting I mentioned to you before we started recording that, I went to high school in Waterloo where you did your undergrad and master's at the University of Waterloo. And in high school I had a summer job doing testing of Blackberries at Rim, which was a really, really tedious job.

01:02:42 I think the whole summer that I spent doing that, I didn't find a single real bug. I just found a lot of issues in the evaluation spreadsheets for a particular application for a particular type of Blackberry phone. Anyway, so yeah, you've done Amazon Rim, you did personalization and recommendation systems at Netflix, and you co-founded a unique AI based musical assistant called Wave AI where you helped elevate human creativity with generative AI tools. You have degrees in math and computer science specializing in combinatorics and optimization, and you're also a singer, a tuba player, and a guitarist. We've got guitars in the background, many of them. And so as a musician with a math competent Torx background and all your AI experience, what have you learned about the boundaries, about what's uniquely human creativity? Do you think there are any, or do you think that machines will be able to, in our lifetimes, kind of take on any creative pursuit that we limit to only humans today?

David Loker: 01:03:57 These are super interesting questions. So this is an area of passion of mine in particular, and as well as my wife. So Wave AI was co-founded between myself, my wife, and one of her old students, Chris, and this idea, can machines be creative? What is creativity? That's the first question we have to ask ourselves. What does it mean to be creative? Because it has had definitions that make it inherently something that only human beings can partake in. But we have to sit there and think, is there a definition that's somehow agnostic to humans? Because some animals can be creative. There are birds that are creative, there's lots of different, and I would argue nature is in and of itself a pretty creative entity or a concept. And so what does it mean to be creative? And so one of the definitions that comes up in literature is this idea of novelty and value.

01:04:56 Is it new and does someone find value in it? And so that definition is interesting because it doesn't preclude machines from being creative. It doesn't preclude anything from being creative, and it still satisfies. I think most people's reasonable definitions of one human being is creative. They something that's new and it has value to somebody because anybody can create something. But if it has no value, do we consider that art? Do we consider that creativity? It's hard to say if there's no value. And so I think that machines can be creative for that definition of creativity. The main thing is the question that I like to ask on top of this is should we allow machines to be creative? Should we allow them to create art and not have humans involved in the process? And from my perspective, the answer is probably no. I don't really want to consume purely AI generated art.

01:06:02 That's not of interest to me. I want to have it come from a person because I can identify with that person. Ultimately. Can I feel something from AI generated art? Sure, yeah. Potentially. It's just not what I would choose

to consume. So our company, one of the main focuses, and I think the most time we spent was what is the user experience? What do I want to do here? What is my goal? And our goal was to make human beings more creative or to enable humans to be creative who previously thought they couldn't be. And at the end of the process, we wanted people to walk away from that machine and be able to continue to create in a better way that they could previously. And that was our goal. And so that was antithetical to a lot of VCs. They wanted machines to replace.

01:06:56 They wanted the idea of the ai, Spotify where humans aren't involved, there's no royalties, and you just generate music indefinitely. That's not for me. And so we spent a lot of time, what does that interface look like? How does a machine interact with the human so that when you're done the artifact that comes out of it, you feel it's yours? And to a large degree, it is. And that was a difficult problem to solve, and I think we did a pretty good job at it. Ultimately, there's a long way to go. And I think some of these systems that you'll see now, like Suno uio, whether it's AI image generations, they're moving in the direction of how do I give the human being more control? How do I make it so that at the end they're not like, this is cool, but I'm not going to use it again.

01:07:42 It's not really mine. And they walk away. We want to, as a human being, we want to be involved in the creative process. I think it's innately us somehow. We want to create music, we want to make art, we want to make dance, we want to do movies. This is innately, we spent so much money on this. It's a human trait. And so I think if we want to have AI be involved in creativity, I think it needs to enable, I think it needs to enhance. I think it needs to broaden the ability of human beings to be creative, but ultimately the human needs to be the creator and the machine needs to be in the backseat. And

that's something that I think not a lot of companies do right now. And I think that's what we try to do. And I think that's where it should go if we're going to allow AI to be involved in the creative pursuits.

Jon Krohn: 01:08:34 Big, big, big shifts coming. We've talked about a lot of them in today's episode for software developers, AI engineers, musicians, lots of big changes, opportunities coming, and I'm excited about it. It sounds like you're excited about it. I do. So basically, we're done with the interview now, but now I have two questions that I always end every interview with, and I forgot to prepare you for this. So most of the time I tell you before we start recording, but I forgot. So do you have a book recommendation for us by chance? David,

David Loker: 01:09:07 If you're interested in AI creativity, my wife has a book coming out that I would definitely recommend to Creative Machines. It's coming out in October. It tries to distill in a very approachable way where we are in this moment as it pertains to machines and creativity. And I think it does a good job encompassing the history of the field. Actually since the 1950s, this field has existed and it does a good job telling and understanding the moment that we're in. So I would definitely recommend it. I read it. I thought it was really well done. Not to push my wife's book, but just from the last part of our conversation, I would say that I think there's a lot of great material. I would more, if I'm going to get into this area, and I'm going to learn a lot about agentic systems and I'm going to learn a lot about AI coding, I don't know if I would go to a book per se.

01:10:05 I look at podcasts, I look at YouTube channels, I look at Coursera, deep, deep learning, ais, all those types of things. Those are the content that I consume in order to get my hands dirty. At the end of the day, I like to learn by doing these things as opposed to just reading them. And I think if you want to get into this moment and you

want to really understand what's going on, that's the route that I would take. Just build something. You can use these systems pretty easily. Try them out. Because a lot of times people are criticizing these systems and saying all this stuff about the moment, but you have to first try it. You have to understand where these limitations lie. Not as scary once you actually get in there and you start to fool around with it.

Jon Krohn: 01:10:46 Yeah, yeah, yeah, I like that. The two book recommendations there, one Creative Machines. It does sound great. What's your wife's name so that we can find that easily and have it in the show notes?

David Loker: 01:10:57 Yeah. Dr. Maya Ackerman. There we go. She's a professor of Santa Clara University.

Jon Krohn: 01:11:02 Alright, that sounds legit. It doesn't just sound like just a nepotistic recommendation. That sounds like a legit one. And then in addition, your second book recommendation is Build something. Yes. Just get out there, something,

David Loker: 01:11:18 Don't read about it, don't read about it. Go out there and build something. Yes,

Jon Krohn: 01:11:21 I agree a hundred percent. That's the way to do it, especially when there are so many tools out there that make it easy for us to be building. So get on it, see what's possible, find out where the boundaries are in what humans can do with machines. And David, I have loved this conversation. This has been such a great podcast episode. For listeners who want to get more of your thoughts after the episode, how can they follow you?

David Loker: 01:11:46 Well, I don't really partake in social media, so following me, if you want to, you can follow me on LinkedIn. I do post on there. I do also post on our blog, but ultimately, yeah, if you want to and you're curious, you can reach

out to me on there and I will do my best to respond to the people and ask and hopefully engage in some interesting conversations.

Jon Krohn: 01:12:05 Something that I sometimes recommend is if it's a question, basically, I guarantee that if somebody asks me a question as a public post, I'll answer it every time because that way lots of people can benefit.

David Loker: 01:12:18 That's a good point. Yes, that's right. If you have a really interesting question, definitely some people are scared to ask questions publicly and they don't want to be shamed for bad questions. I would say there are no bad questions, but either way, human beings are human beings, right?

Jon Krohn: 01:12:32 Yeah, yeah, yeah, exactly. Alright, so David, that's it. Thank you so much for taking all this time and giving us such wonderful insights. Looking forward to seeing how your journey and CodeRabbit journeys, how your journey and CodeRabbit's journey, how the both of those journeys unfold over the exciting years to come.

David Loker: 01:12:50 Awesome. Thanks very much. I really enjoyed it.

Jon Krohn: 01:12:56 I seriously love that episode with David Loker today. In it, he covered how CodeRabbit combines pipeline AI and a Gentech AI to provide expert level code reviews, how context engineering feeds LLM issues, dependencies, and code graphs alongside diffs to catch bugs that span multiple files and understand developer intent. How developer productivity metrics must now shift from things like lines of code to feature delivery and problem solving is AI handles more and more routine coding tasks. How GPT-5 doubled its performance on CodeRabbit's hardest test cases, jumping from 10 to about 20, correct out of 25, showing breakthrough reasoning and multi-layered logical chains. And he talked about the importance of keeping humans central to AI creativity tools so people

feel ownership over their creations and become more capable creators. As always, you can get all the show notes including the transcript for this episode, the video recording, any materials mentioned on the show, the URLs for David's social media profiles, as well as my own, at superdatascience.com/927.

01:14:00 Thanks to everyone on the SuperDataScience podcast team, our podcast manager, Sonja Brajovic, media editor, Mario Pombo, partnerships manager, Natalie Ziajski, researcher Serg Masís, writer Dr. Zara Karschay, and our founder Kirill Eremenko. Thanks to that awesome team for producing another excellent episode for us today and for enabling that super team to create this free podcast for you. We are so grateful to our sponsors. They make the show happen. You can support the show if you are interested in doing that by checking out our sponsor's links in the show notes, or you can find out how to sponsor an episode yourself by going to jonkrohn.com/podcast. Otherwise, you can help us out by sharing this episode with people who would like to listen to it or watch it. You can review this podcast on your favorite podcasting platform or YouTube subscribe. Obviously if you're not a subscriber already. But most importantly, just keep on listening. I'm so grateful to have you listening and I hope I can continue to make episodes you love for years and years to come. Till next time, keep on rocking it out there and I'm looking forward to enjoying another round of the SuperDataScience Podcast with you very soon.