Coding Agents are Dead: Long Live Coding Agents! with Nicolay Gerold (Amp Code)

65 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/LIVE/XW_XJ_ZCQO4?IS=J692JFJOFNBRX7Y4
https://www.youtube.com/live/xw_xj_zcQo4?is=j692jFjOFNbrx7Y4

SUMMARY

Nico Gerard and Hugo discuss the evolving landscape of coding agents and AI in the latest episode of "Vanishing Gradients." They explore the rapid advancements in AI models, the implications for software engineering, and the importance of adapting to new tools and methodologies in coding. The conversation highlights the balance between leveraging AI capabilities and maintaining human oversight in the development process.

- *Coding agents are evolving rapidly, necessitating constant adaptation from developers.*
- *The concept of "coding agents are dead" reflects a shift in how coding tools are used, rather than their complete obsolescence.*
- *AI models are becoming more capable, allowing for more delegation of tasks and reducing the need for constant human oversight.*
- *The importance of "taste" in coding is discussed, emphasizing the need for models to align with human preferences in code quality and structure.*
- *The conversation touches on the advantages of using TypeScript and Rust for building agents, highlighting their type systems and feedback mechanisms.*
- *Future coding agents are expected to operate more in cloud-based sandboxes, automating background tasks and improving efficiency.*
- *The role of open-source models is anticipated to grow, potentially catching up with proprietary models in capabilities.*
- *The upcoming workshop will focus on building coding agents, emphasizing context management and the evolution of features in coding tools.*

What is up, Nico Gerard, and welcome to Vanishing Gradients. >> Hey Hugo, nice to be here. >> So great to have you here, and we recently had you on the Show Us Your Agents live stream, where you gave us a teaser of the types of things you're working on and building at Amp, which was super fun. But it's nice to be here with you to have a like a a dedicated chat about everything, all the fun stuff you're up to.

>> Yeah, it's really fun. And it's like, like you said, like two non-Americans chatting about AI, it always feels It always feels a little bit wrong. >> Exactly. As you said, we're in the doghouse. I am Yeah, I So, everyone, before we we went live, I mentioned the State of AI Coding Podcast from Armen Ronacher and Ben Vinegar, which I I recommend everyone check out. I I I learn a lot in it, and I learn like a lot of, you know, One of my One of my metrics is insights per minute, and the insights per minute are through through the roof for for me.

So, definitely check it out, but it is nice hearing Armen from Europe and Ben from Canada talk about AI. It's

not, you know, the Silicon Valley Market Street, take my vector database money shenanigans, right? >> It's AI with a siesta, so a lot more family-oriented with a lot more breaks, especially >> >> Exactly. Exactly. So, everyone, welcome to Vanishing Gradients. We're here to talk about building coding harnesses and coding agents.

The Q&A will happen in Discord, so join us there, ask questions. Please introduce yourself there. Let us know who you are, what you're interested in in the podcast channel, and the link is in the description and in the chat here. Also, check out the description for some upcoming courses we have, and another live stream I'm doing with Nico next week on actually, you know, the ins and outs. We're going to build a coding harness with you, and we'll talk a bit about that. But a couple of workshops I'm running and a course I'm running on building production agent harnesses with none other than Doug Turnbull who used to lead search at Reddit and Shopify among other places. But Doug is who the wonderful guy who introduced you you and me Nico originally, right?

>> Yeah, great guy. And if you're interested in search, I can only recommend Doug's courses as well. >> Without a doubt. Without a doubt. So, Nico, software engineer, working on amp, founder of Iceberg Ice Iceberg Iceberg >> Iceberg >> Iceberg. writing about coding agents as well, software AI in production. one other thing I would like to share about Nico, let me share my screen.

I haven't shown Nico this. You may have seen it, but for everyone who comes on Show Us Your Agent Skills, I make them a little website. And have you seen this, Nico? >> No, I haven't. >> This is and I'll share it in Discord as well. This is a website I've I I I've built for you using a bunch of agent slop as well about everything you you you showed us. And you can click through to see what you're working on in your demos there. You know, you showed us your Google Cloud skill, your team up skills, your thread postmortem skill, which was absolutely fascinating. All your kind of philosophies around reproduce, inspect, verify, debug the instructions, not the code, looking at artifacts, all of all of these types of things. And the stack you use. So, you know, from amp to amp threads to Google Cloud Cloud Code Pi, all of these things. So, I'll share that with you as well after the fact.

>> Nice. Thanks. It looks awesome. >> so, Nico, we have called this podcast Coding Agents Are Dead, Long Live Coding Agents, which you know, is is a provocation in in some ways, but maybe we can start by you You me why why do we call it coding coding agents are dead and not coding agents are evolving? Because one could like clearly they're not dead, right? But calling it evolution doesn't really capture what we're talking about either, does it?

>> It's I think this is one of the like core philosophies or approaches we have at Anthropic that makes us a little bit different. It's like we constantly throw away large chunks of the code base and it's really interesting that when you because the space is moving so fast and AI is getting better and better, you always have to with like when these step changes happen like for example now Opus 4 A to fable, you have to rethink a good chunk of your product from scratch.

And this is like when your model is suddenly so much more capable and you can entrust it way way way more and not be in the loop. Then the how you handle a coding agent will change drastically because you don't have

to like look at all the code anymore. You don't have to be in the loop constantly. So you don't have to have it run on your own device at all the time. You can delegate way more which removes a lot of the restrictions or a lot of the features which makes some harnesses or some products really useful right now which is for example like work tree management and automation.

>> It makes a lot of sense and I wonder on top of this thinking through what's happening as the models I mean you've used the analogy of Kirby before, right? So what happens as the models start to Tell us a bit about Nintendo and Kirby and what happens as the models start to suck up the harnesses around them. >> Yeah, I think that Kirby for those who don't know it's like one of the like OG Nintendo games I would call it. And Kirby is basically this pink monster which is just eating other things and then acquiring the capabilities.

And this is very similar to I think like how models are evolving. and one example which we had is like we built handoff as one of the core features of Amp which was basically hey, you're in an existing thread but you want to continue work or do a related task. So, you trigger handoff with the next task description and it basically puts you in a new conversation, a new thread and you can continue in there automatically. And we built this originally as a response to compaction because compaction didn't work well.

and I think like a lot of people like when you get into the space and you initially use Cloud Code, this is also what you realize. Every time you hit a compaction, you were scared because you lost all of your progress and the model sucked afterwards. And now the model the model's got competent enough that the automatic compaction is so good that they contain most of the useful information from the prior one. And at the same time, the models are so intelligent that they the assumptions they are making and the extrapolations are usually way better that it's also when it's losing information, it makes the right assumptions or it discovers the right ones. And this is basically like Kirby. And the model like wasn't capable of compaction. It's very trainable. It was trained in over time, so it's basically acquiring that capability and the feature's gone now for us because we don't need it anymore.

It's you don't have to worry about this like handoff flow anymore because the compaction is so good. You can just stay in the same thread for days on end and compaction will be good enough that you can just continue without having to worry about it. >> I'm just wondering what this means for your job. if if you're building things that are are being sucked up into another form of infrastructure and in some ways invisible infrastructure cuz I think you know the the model labs I presume it's you know the you know reinforcement learning via verifiable rewards on agent traces or whatever it is that allows them to then make the models really capable at this stuff. Kind of I compare it to you know we all used to do like what we call chain of thought prompting or whatever and then suddenly you know reasoning models seem to have sucked all of that up and I presume it was some form of RL that allowed that to happen as well but firstly is this what you what you think is is happening how these capabilities happen and then what does it mean for you as a builder? It feels like you're building on shifting sands.

>> Yeah we are building on shifting sands. I think like this is true and I think like this is also like the fun of it. Like it's always changing and you always have to be on your toes. what they are training I think like for one the

the AI bit like what they are figuring out it isn't the architecture ideas. If it would be the architecture idea it's like the Godfather of the space would be Schmidhuber who says he invented everything which came in AI in the last 30 years.

and I think it's more about the recipes. Like how can you actually like squeeze all the performance out of the models and this is especially how the at the moment I think like the current method is a lot of the environments where the data for reinforcement learning is generated. and I think like this is a really complex space where like the big model labs have a clear advantage where they can actually generate really good rollouts and actually have really good ways to actually evaluate whether a rollout is good that they can train in a lot of behavior which was previously not really touched by the models. And some of that is for example coding taste. I think like Fable is really good at taste where it sometimes just produces solutions where I say, "Hey, yeah, this is probably what I would have done would I have spent like a day thinking about the problem and then actually coding it up." Whereas I think GPT 5.5 often really doesn't have that.

I have to do like multiple rounds of feedback until I arrive at a solution. the trade-off is Fable is disgustingly more expensive. and I think over time it's still to be said like what the mid models can't acquire and whether we will see like a a limit in the capabilities which will come into the model. and this is like TBD. I think like I a lot of people have said like, "Okay, progress is slowing." and then like the next model generation comes out and blows it completely out of the water.

And I'm I wouldn't make a prediction in that domain because I think like there are still ways to go in how you can improve the models. Is that through scaling or better environments or better feedback mechanisms or better benchmarks? And I think there is more to come and more capabilities to acquire. But this also means like what is left for the human. And I think like one of the things which is clearly it's like what is actually worth building. I think like all of these decisions are at the moment like really far out of the model. And I think like the future, should the model acquire that as well, is really boring. So, I'm not really preparing or building for it.

because I think like if the model is producing all the ideas and it's executing on them, like nothing is left for us to do. And I think like this is a little bit boring. So, I think like this taste and deciding what to build on and what's worth putting into the product is like in the human domain. The second one is a verification. You still have to be on top of things and own the things the model has built.

Like it's your work in the end. So, you have to maintain like the level of capabilities in your craft, whatever you're doing, whether it's design, coding, engineering, that you can actually evaluate the output of the models and understand it, even if you delegate the thinking like how to solve something and the execution to the models as they become more capable. But in the end, like you're like the guardian or of the code base that actually decides, "Hey, should this make it in and is it good enough to be in the code base?" And in a lot of cases, it isn't because the model can't fit the entire code base into its context window. So, there are a lot of things that you probably have in your head that the model just hasn't. And at the same time, there is a lot of information which is domain-specific, which you either picked up over time or and that has never entered the training data, which you still can contribute to the model. And then it's really about like, "Hey, how can you bring that

knowledge to the model and level it up in your code base as well?"

>> Totally. There's so much in there. And you mentioned Fable, which you know, was released for 2 days, then pulled for all the political reasons. and then in the past 24 hours, it's been had a general release once again. I am interested in when you say it has taste. I mean, there's a big conversation around taste and judgment as opposed to in intelligence and we often think of models and harnesses as not having taste, as possessing certain forms of intelligence and procedural intelligence, mimicking reasoning, whatever it may be, large-scale at scale pattern matching.

what do you mean when you say it has taste when writing code? >> I think like this is a really big question. it's I think like software taste depends a lot of the the code base you're in and I would say taste is if I put a good engineer down and give him a lot of time to actually reason through a problem and understand it and come up with the right abstractions, with the right concepts and figure out the right places to integrate it into the code base and then write the code. If that matches with the output the model gives me, close enough, I would define that as taste. So, basically, what would a good engineer who isn't time constrained put out as the goal and then basically how close can we get to that is for me the biggest measure or metric of taste that I could come up with.

But, defining it, it's so specific. It depends on the language you're in, the code base, but also your preferences. I think like some people like really small functions which are descriptively named. Other people like the really deep modules and which hide behind a really good interface and I think like this comes down in the end what's your preference. So, this is also an attribute. How close can the model actually match what the user wants?

And this is often something that's that's pretty hard to do because it's it's really broad. There are like a lot of different scenarios, a are different users, a lot of different preferences of how code should be written. So, the model has to adapt. And I think this is another thing that the models are probably not going to be the best at because they they won't see everything. They can't train on every single way to produce code and output code.

so, if you have a very particular style, you're going to probably have a harder time producing code with it. But, also there might be more differentiation for your job. I would be really curious something like TigerBeetle, which are building like financial database. They have this like tiger style, which is really interesting for building building really reliable applications. And I would be really curious like how good are the models of actually producing that.

and if they aren't, I think it is for one a handicap because you can't take advantage of like the capabilities of the model to the fullest extent. But, as an engineer, it's also an advantage because if the model isn't so good, I have more of a staying power to actually stick around and be around. >> Awesome. What What is tiger style? >> Mhm. Tiger style is basically a a way of writing code they put out. And they have a few really interesting documents on that in in their code base and also in their docs. Like one of them is basically they put assertions in a lot of places for the entry and exit conditions. And this is like one of the style patterns they're using, yeah.

>> I just made it made me think of Street Fighter 2. I don't know. I came up in the '90s, so Sagat, the boxer, and

it the movie's coming out. He has a a boxing a punch which he goes, "Tiger! Tiger!" And I thought maybe they they were riffing on that, which would be cool. I also just quickly want to We're talking about like modern models, and by modern I mean like the past week or two or whatever.

but you know >> >> we hear that open weight models well open weight models aren't at you know par with a lot of frontier models yet. They lag you know 6 months or a year depending on who who you speak to. I do think you know with that models like claim 3.6 which you know for smaller code bases when you don't have to do like large things it can be pretty good. Like I I find it I'm pretty successful with with pie but then this in the past week or two we've seen GLM 5.2 right? which is a really interesting model besides not being able to show it images it it seems you know like kind of up there with a lot of the frontier lab models. The reason I'm asking you about the these models let me get this right well is because open weight models classically we don't think if if they if you can build good coding models it's because you've got access to traces. So you're not just a model lab you're an agent lab as well right? So I'm wondering kind of your thoughts on I suppose Z also has their their agent stuff but to build you know really successful coding models do you need to have access to all these traces and that is that why we're seeing the these ones become pretty good?

>> I think access helps. I don't think it's a cure all and I think like cursor open AI and cloud code have an advantage with a lot of the data but I think like the advantage is decreasing because the new traces they are getting and the most interesting ones they can't train on because usually that's in enterprise contracts and they have like strict data protection. Usually you have some form of CDR and you have an agreement that you can't train on that data. so I think like the the data advantage isn't really in the traces from from the customer for the training. I think it's more for grounding what the data mixes. So I think like this is something like Anthropic for example had this paper called Cleo which they put out which is basically like privacy preserving analysis of user traces and they basically extract something like a task distribution. Like what different kinds of task are the user actually using the model for? And this can be really interesting for training because you actually know hey, what's the realistic task distribution? what are users are doing with the models? How is it shifting? And you can apply that also do an issue analysis through that. Like what are the different issues that are present in the traces? And then basically come up with a recipe to target those. And through this they have a lot of advantage because they have a lot of ongoing usage.

and then they can basically target the data they generate to train the model on these specific data mixes like in terms of task but also try to create new environments that target specific issues that the model have and iterate based on that. And I think like this is a massive advantage that the big lamps have and that just comes through having a lot of users. but the advantage of actually using the traces of users to train on a model, I don't think that's as present anymore.

>> Well, I mean it's why Cursor is seems to be worth 60 billion US dollars as well. What an What an bloody exit as we say in Australia. I don't know if anyone's ever said that in in Australia. But I am I So, firstly, everyone watching, we've had some cool comments in in Discord, but feel free to ask any questions. Nico's building one of the most interesting coding agents and coding harnesses out there at at Amp.

and if you haven't checked it out, definitely check it out at ampcode. But Nico, I want to know what from my experience, I don't build coding agents, but I use them way too much. I use them 8 days a week, and things for me that have changed are like, now I don't really need to manage my own context as much as I I used to. I'll I'll have endless building sessions a bit more. you know, that like compaction is is changing. Queuing is interesting.

Steering is becoming really far more ergonomic than it was even a month ago in a lot of different harnesses and products. And if people haven't experienced steering, some I mean, different harnesses or different coding agents referred orders, different things. Most are calling it steering now, but it's it's where your agent is doing something and you want to inject more information into it. Previously, it would like stop and stutter, and then you'd restart and all of that. And now you can, well, steer it, right? we've got more remote controls, so I can do stuff remotely. these are things from my experience. I'm wondering for you, what does the bleeding edge of, you know, coding and general purpose agents look like today?

>> It's I think like one thing that you're seeing in basically everyone and with us as well is like more execution and sandboxes. that the the agent isn't running on your device, but it's running on a sandbox. And I think like this can help with multiple things, but at the same time a few things are still missing to make that really flawless for the user. So, I think like it's helping, for example, with like all the like work trio multiple checkouts on your own device that a lot of people are doing.

And I think this also came with a tone shift. Like, I think half a year ago or 3 months ago, which is ages in in the agent world, is like a lot of people were like measuring themselves like how many parallel agent sessions they can run. And this usually automatically comes I need multiple work trees or multiple checkouts of the repo so I can actually run them. and I think like now it's like it's shifting back again that users are more like running a few at the same time.

I'm actively driving one. So, I have one session which I'm really involved in. And then I have maybe like two or three other ones which are more like just background tasks. and this makes sandboxes for me really attractive because I can just launch off a bunch of things in the background. Like, when I see a bug report on Axo or on Discord, I can just take it, throw it in a sandbox, continue with my work, and once I'm finished I can actually look at the code what it produced, either pull it push it on a branch, and check it out locally, or it's just merge it automatically. And I think this makes it a way more convenient because I don't think have to think about like, "Hey, I have to have multiple checkouts. All of them have to be up to date. I have to constantly pull. All of them need the same environment checkout. And all of them also need to be able to run the dev server and other things which have to run locally." Which usually is pretty hard. because how many ports do you need for five local checkouts?

and for that like sandboxes are a massive advantage. Disadvantage is it's way harder to actually do something like code review or quickly look through the code, read through it, and check it out. and I think like for that there's probably going to be a lot of improvement as we figure things out. And stuff like that could be a new kind of handoff which makes it really easy to basically take the threat which we you were running in a sandbox and take all the changes the agent made and just pull them locally automatically.

And I think like there is a lot of interesting development happen happening. Another thing is like more automations and workflows. And what a lot of people are doing now is basically having one threat one agent managing other agent. And he's basically like this I hate the term but loop engineering but like one agent is basically doing looping over for example a Discord channel and it's just basically delegating the work to other agents which is very easy with sandboxes because he can there's actually no bounds of how many agents he can start up and just push the work to.

Also automatic triggers is I think something that we will see way more of. Like for example just have a connection to Google Cloud and when you see an error or an alert pop up way too frequently it's being automatically pushed to an agent whose actually whose responsibility is to actually check it off and then the agent either delivers a code delivers a PR or just delivers a message with his assessment whether this should actually be looked at by a human.

>> I hate the term loop engineering. You heard it here first people. Leave it drop it in the comments. Do you hate the term loop engineering or do you love it as much as Boris Cherny and Pete Steinberger who also have access to you know have use millions of dollars of tokens monthly or whatever right? So maybe it's it's more acceptable there. But finally when you mentioned a few months ago back in you know ancient times when we were talking about parallelizing and that type of stuff I was like, now it's all it's all loops, right? Which I I actually wanted to ask you about about that.

Do Do you Do you mostly loop engineer these days, Nico? >> No. It's The like I do a lot of work in like eval and on the model side. So, I think like the loop engineering is hard there to do in that because it's like I'm testing the model a lot and then I'm reading through a lot of JSON. And it's I'm not sure. A lot of it it comes down to how your product is set up and what your product is actually. I think when you have something like really well-established bug reports, issue tracking which is coming in, it's really easy.

At the moment, I do a lot of it just manually. Like when I see a bug report come in, which I think should be looked at, I throw it in a thread and have it investigate. I think I could automate more. At the moment, I don't see the need to. And a lot of things is like the I think with agents I I like to protect my time and to actually have some downtime where I can actually think through different problems and actually just implement new features, just think for a bit and put like hand on paper. And I think like with loop engineering, it's going in that direction again, hey, I'm spinning up millions of threads which I have to manage.

Which is like this TikTok phenomenon for coding. I am basically switching between hundreds of threads and my attention isn't on a single one. So, I think like loop engineering is most suited for things where I know, hey, I have a very good input which defines the task very well and where the scope is small enough that I actually know that the model can produce very good things.

And the I'm not sure whether I have found that yet in a lot of the the bug reports or the issues we found. But it's like our product changed a lot recently, but we went into like a really distributed system, and I think like this is something the models are also very bad at. So if I feed in bug reports, I that touch like the distributed system

part, I would never be confident enough in the model to actually just push it or merge it automatically.

>> Yep. We've got a question about loop engineering. How much of it is hype? How much is real change in the agentic AI world? I think you've you've answered a a lot of that, particularly in Once again, you heard it here first, loop engineering is the Tik Tok of the agentic AI world. I will I I will add I'll say as you pointed out, there are places where it is useful. When you have something of some obvious hill climbing form of stochastic stochastic gradient descent, something you're optimizing for, such as auto research. So that's why Kaparthy's example of auto research, which if somebody wants to link to that in Discord, please do, otherwise I can do it later, is really nice. You're you know, you're training a deep learning model, and you're getting an agent to just edit the code as it hill climbs, you know, the loss function. And of course, you still want your hot assets and train test split and all all of this this type of stuff to make sure you're not not over fitting. But in that case, you know, looping can actually that form of auto research loop can be useful.

Doug Turnbull on our shows you skills, and I can link to this later as well, gave a really nice example of how this applies to how you can use this when building agentic search systems as well. So if you've got something specific, a number that you're optimizing for, it can be useful there. As opposed to if you're doing like agentic exploratory data analysis, for example, where part of the optimization function there is human understanding, right? And in that case, or when you're doing evals and looking at data to to your point, right?

like some of the stuff there you can't outsource to a loop as well. And there's a whole world in between. I will also say a lot of, you know, really sharp, really smart builders, and Armon Rodman is is one of them, and everyone should check out, I think, episode 7 of the state of agentic coding podcast with Armon and Ben Binninger, says explicitly he doesn't really do loop loop engineering because he doesn't really trust it yet, and most of his use cases it isn't really useful for. so, I think to encapsulate that, some classic things it can be good for, but when you're building cutting-edge stuff like you are, Nico, it doesn't seem that appropriate or relevant.

>> Yeah, and it's also I think like our our setup is different. It's like if you have I think like the predictable I think like I have to separate two things. I think like one is a Ralph loop. like this form of loop where you basically just have to the model like loop on a task until it's finished. I think like this is one kind of loop. And I think like for that, the models tend to be really good at if you have a clear objective it's running against. And I think like auto research is the best example. And then the other other loop I'm seeing is basically the model driving your entire code base. So, it basically has a place where it picks up bugs, feature requests, issues, nits, and it automatically delegates them to other agents. And I think like the loop engineering bit, I think like the Ralph loop has a real It has a place for a lot of things, but what it usually ends up with is like a whole a lot of code that you then actually have to review. And what I haven't found it's it's easier for me for my personal workflow to actually do it chunk-wise.

and do it in mergeable chunks which I can put in the code base. So, I can keep an overview and an understanding of the code that I'm actually putting in and build it incrementally. for the loop engineering bit, if I'm not sure whether a lot of companies have for one a good enough setup where actually the tasks are well-defined enough in wherever you're sourcing them from that they can just be given to the model as is

without any additional feedback from any engineer which tells it, "Hey, where to look and what to do." And also whether their code base is and how they are driving the code base is set up in a way that I'm confident in a model to just merge it automatically. And this is stuff like, "Hey, I have really fast rollbacks."

or I actually know that over time we have a a certain quality level in the code base already. So, the agent is producing higher quality code. And at the same time I'm refactoring constantly to check on the quality to actually make sure when the agent is producing bad code but still merging it, I can keep the quality level high. And there are some examples which is I'm I really wondering about open claw. Like what Peter is doing.

Like he's doing some crazy stuff. I think like he's put pushing the edge, but I I in my case couldn't do it where I just automatically take an issue, let it work, and let it merge it. but I think like he is really pushing the frontier of like where the agent could be like when the intelligence gets better and better and better. But in like a lot of enterprise settings, I think you you couldn't do something like that.

>> Totally. Okay, I have a few more things to say on this. Firstly, in the Discord I've put Jeff Huntley's original everything is a rough loop blog. I put the Auto Research stuff in and I put in Doug's your agent skills Auto Research for agentic search agents. I also just want to get off Twitter, everyone. Like seriously, go and build stuff and don't listen to all this nonsense. Also, remember the lab like never ask a barber if you need a haircut. Like of course the labs want you to think about building. Of course Boris Cherry, I don't know whether he self-consciously does. Of course Anthropic wants you to loop things, to spend as many tokens to build things and then have to spend as many tokens to solve them as well, right? So, just you know, reason through these things.

there's and to your point earlier, you like triggers increasingly important. And this is something I put a link to the workshop I did with our mutual friend Ivan Leo, who was at Manas then and now is at DeepMind on building in Python a pie-like and open-claw-like agent. And guess what? One of the We didn't talk about loops. We talked about hooks, right? Get When building When building agents and/or using agents, thinking through what's a reasoning loop, what's a tool call, considering context. If you're building agents that you like most people building agents, customer service agents, whatever, like you don't even need to think about context cuz you want to get to resolution in, you know, a three-turn conversation or whatever it may be.

think about cron jobs. As I always joke, Dr. Dre called his album The Chronic for a reason. and think about hooks and and triggers. These will get you a lot further than all this loop engineering business for most things, right, Nico? >> Yeah, and it's like when something can be done deterministically, I would always do it deterministically and then put a fuzzy AI in the middle of it. but at the same time, I think like you can underestimate like what agents can do as well. I think like a lot of stuff is just still stuck in I'm chaining eight models together in a workflow. which probably can be done one shot by an agent if I give it two or three tools and structure the context in a good enough way. So, I think like there is a mix of like you shouldn't be too stuck either way that you're always thinking about workflows and triggers or you're always thinking about agents.

It's always switching and it's always dynamic. And usually when the model acquires anything like this is part of

the Kobe thing. Like when the model acquires more capabilities, you can remove more of the scaffolding and the harness you put around it, which includes workflows, triggers, hooks, etc. >> Mhm. So, now I we've been talking about what it, you know, what all the experience of what you think about building an agent harness and building an coding agent which, you know, is been heavily adopted by the enterprise as well. It's not just not just punters.

and I'm wondering your thoughts on what the experience what builders should be thinking of. Not people who are building coding agents, but people using them cuz it's pretty stressful, man, as an end user like having constant changes and updates and some which super power you, but some which like you even you try to sleep and then you get up and it's like all this new new stuff. So, how should people using them be be thinking about using stable products or these types of things?

>> Using stable products, I'm not sure whether we have stable products anymore. >> Yeah. >> The >> Or not, right? >> The how how should they think about it? I think like you should I think like agents have found the exact right place to dominate first, which is engineering. Because I think like engineers are all lifelong learners. You're constantly adapting, you're learning new things, you're learning new languages, you're learning new frameworks, you're learning new concepts, you're diving deeper in the material. And I think like the same thing applies to agents as well. Like you should learn how stuff works, you have a good understanding of engineering, software engineering, coding, the things you're working with, but you should also build up a good understanding of how agents work and what tasks you can give it, and also how to best present it to the agent. And I think like both are skills that you need to have in the future, and I don't think you can survive without either one. I think like they you still need the technical jobs, but you need also to be able to like just give tasks to the agents and phrase it in a way that they actually can do a good chunk of the coding.

and I think like this is just like experimenting a lot, trying out a lot, and building up your understanding of how it works over time and constantly refining it as new models come out. And it I think it really pays to actually just try the new models, compare it, what does work, why does it work, why doesn't it work, and also why might it be the way I prompt or use the agent that's the bottleneck as opposed to the model itself.

And this is what for me the the postmortem is about. And I think like I used that a lot also for my own instructions. Like often the model fails because I phrase something poorly or it tells me an assumption it's it made which is wrong where I have to correct it. And I think like this concept reflecting, improving, and then basically applying it again is something that applies to both engineering and to using agents. I think like you just have to stay on top of it and actually do it.

And at the same time try not to stress too much about it and succumb to the FOMO that you have on Twitter. I think it's fine to like you don't have to jump on the latest trend all the time. You don't have to be like loop engineering, Ralph engineering. I think like you have to develop your own philosophy of what makes you productive with agents and that's different based on what you're working on and also what your style of working is.

And I think like as long as you can develop a philosophy and actually improve it over time as the models improve and also keep an open mind to actually new techniques like loop engineering as well. you're in a really good position. And I think like loop engineering I think like at the moment it's not like as applicable to what I'm doing in my day-to-day but it might be in half a year. So I have to constantly keep an open mind as well because like what I disregard in like at the moment with the current models capabilities or like half a year ago could come back at some point.

>> Totally. And what actually that's a really interesting point cuz where my mind went was when we spoke a while ago I was like do you write code anymore? Do you read code? And you're like well it's funny I kind of bounce back and forth depending on what the models and the harnesses are like and you're like I'm putting words in your mouth but something like few months ago I I barely not writing code at all. Agents were writing all the code, and now I'm actually back to writing. So, maybe you can tell us about this this oscillatory experience you have kind of building on the bleeding edge.

>> Yeah, it's it's constantly switching, and it's also depends on like what I'm working on. When it's some stuff where I know like the code is very well, and I know all the issues and the problems very well. Usually, I can just like tell exactly the model exactly like what to build, where to put it, and how to implement it. And then I don't have to read the code or edit as much. And this switches back and forth as we are just a product and improve it over time. So, this is one part of it. The other part is just sometimes you have like weeks where it's actually it's working very well. And it's either the model or you have adjusted to a certain model in a really good way where you can actually hit a sweet spot where it's just flowing constantly. And then you have weeks where I'm like, "Hey, this it's like agents all suck. They can't produce a single line of code correctly. And then I'm hands-on more. And or when I notice, "Hey, I have produced a lot of work with the agent, and I am not confident about the edits anymore because I'm not sure whether it's a good edit to make or it's in the right place.

And then I usually have to do more legwork and actually read more code again and actually do some stuff by hand so I actually can build up my understanding again and make sure that the model is actually doing good patches, good edits. >> Totally. Okay. If someone wants to build agents, and they said to you, "What programming language should I learn?" Would you say TypeScript or Python or something else? >> I would never say Python. And I would I don't think at the moment there is much advantage out of TypeScript.

Maybe Rust if you need a lot of memory safety, but I think like the both TypeScript and Rust have massive advantages with agents. For TypeScript it being you can use it across the entire product, which helps a lot. so you can use it in the front end and in the back end. And I think like for most software, I think like this is a really good thing to have. And you aren't as performance sensitive that you actually need something like Go, Python, or something.

it also improves the tools you can give to the agent because it doesn't have to juggle multiple feedback mechanisms for the code base. So because it can just use one tool for type checking, which works across front end and back end or your different microservices. for us it's similar. I think like Rust by you can use it for a lot of different things. And you have the type checker, which is really pedantic, which helps with agents a lot because

it has a way more guarantees. So you remove one problem. Python, like why I don't like it as much. First of all, it's like typing.

I think like typing is one feedback mechanisms for the models, which actually help them improve a lot. and I think the in in Python there are ways for the model to actually up, which isn't present in other languages. And especially Rust. And I think like TypeScript is kind of in the middle because you still don't have the same guarantees as in Rust, but you have more guarantees as in Python or better feedback mechanisms.

but in the end, I think like the it always like this is my personal pick or my personal picks. I used to do a lot of Python as well, and back then I would have said pick Python, because it's the technology or the language I am most familiar with. And I think like this still dominates. Pick the language framework you are most familiar with, because one of the feedback mechanisms for what the agent is doing is you, and you're more likely to give good feedback and good advice when it's actually in a language or a framework that you understand deeply than if you pick the latest technology or the latest, trending meta, which everyone else is advising.

>> So, firstly I I think I agree. I actually don't know enough about TypeScript and Rust to totally agree, but people far smarter than me, including yourself, all all kind of converge around what what what you're saying. I do want to share an experience I had a couple of weeks ago. I was building an agent harness from from scratch, and I thought in my global agent instructions it said use Python, and in fact it did, but the agent decided to ignore them or didn't read them for whatever reason, and started building this thing in TypeScript. And I was like, "Oh, okay.

I'll see what's up." so I continued and after a while I started getting kind of lost. And what I realized in that moment, I've had similar experience or experiences with similar similar revelations, to step back a bit, as you know Nico and maybe some listeners know, my background's in like science and data and ML stuff, and so I've learned all the software stuff on the fly as I've needed to needed to. I've never like really sat down and learned the fundamentals. And funnily now I feel like more than ever I should go back and learn the fundamentals. And it feels like this cognitive dissonance because it's like oh, anybody can build software now, which is kind of true, but clearly not true.

AI agents are enablers, but but they move so quickly they're exposing all my weaknesses I I I think and the fact that I don't really understand TypeScript yet means I'm not comfortable in debugging or reviewing. and I'm just interested in your thoughts on on that. Like how how much do people need to really understand the fundamentals versus just kind of, you know, start building and be able to do things on the fly? >> I think that you there are two different things to it. I think like you have to understand the fundamentals, but the best way to understand the fundamentals is actually building projects and seeing where it breaks and where your understanding fails.

And in in TypeScript, there are a bunch of things which you have to understand which help a lot to actually figure out, okay, what's going wrong? And one of them is like the event loop and micro queues. which just help you figure out, okay, how could this stuff break or how is it breaking? but I think like this is also something you

just pick up over time. And I think like this is also one of the main advantages of agents. They help you to start. In like when I was in in university, I think it was really hard to actually build projects end-to-end because you had to know so much already at the start of your like learning journey. You had to know like something in the back end, a bunch of obscure technologies, you had to know cloud infrastructure, how to deploy it, you had to know like HTML, CSS, JavaScript for the front end, and then how to stick all of these different pieces together.

And I think like because you can delegate way more to agents, you can learn faster. And then you can also selectively pick certain areas which you actually want to learn more about. So, if you want to learn more about TypeScript, or it's like you think it's going to be important for whatever you're doing, you can't just build a bunch of projects, ignore everything in the front end, and just write code it, and just really drill down on on the back end bit which you code up in TypeScript, and try to understand it deeply, and just go back and forth with the model and ask it questions about the code base. Like, why did it make certain decisions? Like, what are these functions doing under the hood? Like, how is TypeScript working?

Ask it how this would break. Why is this exception handled in this way? And through that, you build up your intuition of a language as well. And if you supplement that with something that corrects hallucinations or wrong advice by the model, like just read a few books or good blog posts by people which are respected in the in the language community in TypeScript, like Matt Pocock, for example, or do his course, then you can basically do both of these ways. You can already be building and doing stuff, but at the same time build your understanding of the code base you're working in, and build the understanding of the technology under the hood at the same time.

>> Super cool. I love all of that advice. We mentioned, you know, some, you know, current programming languages and historical ones. I'm wondering your thoughts on if we need new programming languages for agents. >> It's I think like this is what a lot of people are working on. I think there are interesting developments which are happening in that domain. There are a few really interesting papers which are using Lean in combination with agents, which is really interesting. Like, Lean is a mathematical language under the wood, which basically allows you to verify programs that they are correct. And I think like this is an interesting like it is new, but it's an interesting thing in combination with agents because before it was just so labor-intensive to produce all the code in Lean, and now it's getting easier.

And at the same time there are now language trade-offs which you can make that make maybe a language more laborious to write and understand, but better for the agent or easier to give feedback. And I think like Rust is one of them. Rust is really annoying to write. Like you have to build up your mental model for a while about ownership until you actually understand it and can produce sensible code. And until then it's annoying to write, but with agents because they are generating your code your way faster. And the the trade-off of being annoying to write is usually a hey I have something in return, and this is in Rust's case a really good type system and feedback mechanism for the model that something is not correct. And do we need new programming languages then? I think like the trade-off of a new programming language is always trading data because the model isn't trained on it.

so I would say probably. But what is a good programming language for the agent as its capabilities are just increasing massively? I'm not sure. And I think like you can already get a lot of done in the existing languages, and usually you already have like a really well-established ecosystem around them which give the agents for one a lot of tools, for the other a lot of training data that existing languages already have like a lot of built-up advantages in itself.

>> So, we're going to have to wrap up soon. I've got a couple more questions and then I'd like to give a little teaser for the workshop we're running next week where we'll show people how to build a coding agent. but first I'm I'm wondering and you know, am I now I'm asking an active prediction, but we do work in AI. Of course, as Niels Bohr once said, "Making predictions is tough, especially when they're about the future."

But, I'm wondering what you see the next versions of coding agents looking like as models get better over you know, maybe 3, 6, 12 months. What do you whatever you feel comfortable? >> I think like this is one prediction that especially Thorson Thorson Ballwich is like one of our engineers has which I agree with. It's like the majority of agent work will happen in sandboxes. And I think like people misunderstand that a lot because they think like, "Hey, I'm doing all my work with agents in a sandbox." And I don't think that's the clear reason. I think like we will just they have way more agents running in the background automatically because they pick up tasks from for example, your issue tracking whenever it's created or from some customer feedback channel automatically or from your logging system. That like the majority of stuff will just run in the background and then produce something which is handed and either merged automatically if it seems fine or it's handed over to the human which then continues his work on his own device. So, I think like this will be a split we will be seeing more and more of that the majority of spent will happen in the cloud, but the majority of your work is probably still on device.

And I think like both of them can be true. I think this is one. I think like we will see way more way more access restrictions before it gets better again, before we have figured it out of how to actually do it. And I think the third one, I think the I think we will see an acceleration of open source models catching up. I think that I would say they're at the moment two generations behind and I think like they can get to like being one generation behind the current frontier of the big labs. And I think like then they're real in a real position to be a valid alternative.

because right now already I think like Fable shouldn't be used as your default model. It's way too expensive and most tasks that I'm doing in my day-to-day don't warrant that I spend $100 on them. And so I'm usually using like GPT-5.5 in our case, which is like in the deep agent mode. and when open open models can catch up to basically this standard, I think they will see a majority of the tokens that are being being accumulated, but probably not the majority of the spend companies are making. Because the frontier models will be so expensive that the majority of spend will probably still go to them.

>> So when you say one generation, we thinking 3 months, 6 months? >> I think it's like I think the cycles are like 3 months probably and I think like they can get to that. I think like the we will see like this year probably the next generation of GLM-5.3 I think could catch up to Opus 4A that GPT 5.5. And it's like they have one

release that's probably in the next 2 or 3 months. So, I think that they can really catch up and then whatever Quen and the other labs are doing as well.

>> Totally. Look, we have a bunch of interesting questions in Discord which we don't quite have time for and so if you want to jump in and and have a look at them later. Otherwise, we can answer them next week cuz they're all relevant to what we're doing next week and I'm going to put a link to our workshop next week in Discord, but dude, I'm wondering if you can tell us a bit about what we'll be doing next week in our how to build a coding agent workshop.

>> Yeah, I think like a lot of what a coding agent is is like a way to massage tokens and data and keep the relevant stuff in. So, I think like we'll be doing a lot of stuff around like context management and the different features around it. And I actually want to look at like how it evolved as well. And I think like why how we went from compaction to handoff to compaction because it really shows you well the feature life cycle with models and this like curvy effect we talked about and how you actually in my opinion have to think about developing features around agents that you constantly have to be willing to just try new stuff, try old stuff which didn't work, and throw the existing stuff which works away.

>> I'm sorry, I'm just deleting a comment. I Man, this is hilarious. I did that serves me right for trying to write in Discord while chatting. And also, we'll be doing this in TypeScript, right? >> Yeah. So, we will probably be using I will use a bunch of examples like especially Pi and build on top of Pi because I think like Mario has built so much stuff already that I don't have to recreate it.

And also show it like how I would do it for example if you want to extend the coding agent that you're actually at the moment using so just as an example in amp of how you can extend it with your own features with for example plugins. And I think like the same transfers to whatever your preferred flavor or harness at the moment open code Pi cloud code codex. >> Mhm. Awesome. And maybe for those who don't know you could just tell us a bit about Pi.

>> Pi is an open source project by Mario Techner and I think like if you're building something agentic I you have to do a lot of groundwork which is basically hey I'm doing the calls to different providers. I have the main agent loop which is updating the data and being resent to the model on every turn and it's basically a really good framework around that and really minimal and well written so it makes it really easy to to reason and talk about and it's also open source so you can actually look at the source code.

>> Yeah. And and it's I I mean it starts off having four tools right? Read write edit bash and then it can build its own schools on the fly and and hot reload them as as need be in order to extend itself which and the system prompt is readable and short. I mean you can you can grok it. You can understand what's happening there. >> Yeah. I think that is read still in there? I'm not sure at the moment.

>> >> It's because for most models you don't need it anymore. Like you don't have a read tool in amp anymore. >> Right. So you just rely on bash essentially to Yeah. I mean that's the thing. ideally you just have a bash tool,

right? But if you have a right tool, it it makes things easier, essentially, right? >> Yeah, but I think like another prediction, I think like more and more of the tools will disappear into bash.

>> I mean, it's all back everything you need to do is we're talking about like general computer usage agents, and everything you need it can be done with bash, just not necessarily well with bash or easily with a bash tool, right? >> Yeah. >> look, final question. I've I've linked to Amp in in Discord, of course, but for those who don't know much about Amp, we've talked about threads and that type of stuff, of course. When I first heard about about Amp and started using it, it was, you know, a lot of wonderful threading and sharing threads and and that type of stuff, but it is something that's evolved over time. maybe you could we could just step back a bit, and you could tell us kind of the overarching the overarching philosophy of Amp and why people should check it out.

>> I I think it's like we One thing about Amp is like that we try to focus on, we try to stay on the frontier. Like we try to always squeeze the most performance out of the the current models, and actually think through where they actually make the most sense. And for that, we rely on multiple models. So, we are constantly switching them out, changing our tools, changing the system prompts, to actually get the most out of all the different models.

And this is something that like the big labs, for example, can't do as much because they are usually only relying on their own models. so, I think like the model mixes really improve or like help to improve the the performance you're getting and the quality of the outputs, because I can give my main agents, for example, an sub-agent at hand, which uses a way higher reasoning and a way more costlier model to actually correct its work.

so, I can rely on a cheaper model as the main agent. so, this is one and I think like the second big advantage we have is like we're focusing a lot on team settings. so, we have a lot of thread sharing and reusable threads, reuse the context window of previous threads, and also just extract the knowledge. And I think like these are things that actually make it way more repeatable, and you don't lose as much of your work that you do with like when you restart the conversation every single turn, because this is like I think like one of the big pet peeves with agents is like this catastrophic forgetting, like the agent always have amnesia when you start a new task, because you're in a new context window. And I think like we have a lot of tooling around that, so the agent is actually improving over time and actually can refer or go back to previous work it already has done to actually get the required information or reuse the work that was previously done, so it doesn't go to waste completely.

>> Super cool. You heard it here, people. Check out Amp Code. I've linked to it. Check out their wonderful blog, where there are so many things to learn about what's happening in the space. you can follow Nico on Twitter. He's on LinkedIn as well. and come and join us to build a coding agent with us next week. Also, check out our workshops and Substack, all in Discord, all in the description on YouTube as well. Obligatory like and subscribe, and share with a friend if you're up for that. thanks everyone for joining. Nico, most of all, always love chatting with you, man.

Always learn so much. and thanks for coming and sharing your time and expertise. >> Yeah, it was good to be here, and see you guys next week. >> I'm so excited to build a coding agent with you live. And everyone, all of

you as well. All right. Thanks, everyone, and see you next week. And scene. Man, that was fun.