How I code with AI changed a lot

47 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=XJAMTO2YGO8
https://www.youtube.com/watch?v=xJaMTo2YgO8

SUMMARY

The speaker discusses significant changes in their AI-driven workflow since their last video, emphasizing a shift in tools, models, and strategies for building applications. They highlight their experiences with various AI models, particularly GPT-5.5, and the importance of effective context management and communication with AI to enhance productivity.

- *The speaker has transitioned from using cursor and plan mode with Opus models to a more streamlined workflow with GPT-5.5.*
- *They emphasize the importance of context management and suggest giving AI access to relevant codebases for better outputs.*
- *The speaker has developed a new full-stack framework called Lakebed, which aims to simplify app development with agents.*
- *They prefer using T3 Code for managing AI harnesses, citing its stability and remote capabilities, while also acknowledging the effectiveness of the Codeex app.*
- *The speaker advocates for simplicity in prompts and workflows, often using two-sentence requests to guide the AI effectively.*
- *They highlight the importance of listening to AI feedback and adapting plans based on its suggestions for better project outcomes.*
- *The speaker shares insights on managing pull requests (PRs) and avoiding PR bloat by keeping changes focused and relevant.*
- *They conclude by encouraging developers to simplify their workflows and prioritize effective communication with AI to maximize productivity.*

About five months ago, I made a video about how I build with AI. A lot of you guys really liked it, just seeing into a workflow of somebody who's trying a little too hard to push the limits of these tools and build awesome things. And since then, I've kind of entirely changed my workflow. Back in that video, I was heavily using cursor and plan mode with Opus models and going really in depth on how I would use plans as the core piece to generate the outputs.

That's not really how I'm building at all anymore. from the idees that I work in to the models that I use to the workflows that I've built for myself to get the most value out of these agentic tools, everything's kind of changed and I really want to go in depth on it because to be frank going back to this old video now just hurts me a little bit because I wouldn't make most of the recommendations I made there today. So, what am I doing now? That's a really good question and I can't wait to answer it after I tell you guys a bit about today's sponsor. I don't know about you guys, but I'm shipping more apps than ever. Little things for my team, big things I put in front of people, and more. But I keep hitting the same friction points when I do that. Building the stuff's easy, but

authenticating users and charging them money tends to be really hard. I could just use today's sponsor, Clerk, but that would get kind of expensive. If I have 30 different projects and I'm paying 20 bucks a month for all of them, that's way too much, right? Nope. Because I successfully bullied them into doing unlimited apps on the paid plan and on the free plan. I still can't believe I won on this one because I was pushing them for a while about this because I always was the guy that wanted a bunch of different apps and I hated that Clerk was becoming my biggest expense. Now it's basically free. I pay the 20 bucks per month. I already have like 40 projects on my Clerk for $20 a month and I couldn't be happier with it. Especially now that I'm using their billing product. Yes, they built a Stripe alternative in. Don't worry, it's still using Stripe. They just have the best Stripe implementation I've ever seen. And believe me, I've tried a lot of them and I've invested in a bunch, too. The cost comes out to exactly the same as if you use Stripe yourself, but it's a 100 times easier to set up. All the billing data is set up on users. They have access to it through the user component that you already should be using. And setting up a pricing table couldn't be easier. You just call the pricing table component.

Waste less time on billing and off and spend more time making good software at soyv.link/cler. There's a lot to break down here from the models I pick to the harnesses I use to what UIs I'm working within IDEs, apps, etc. The way I style my prompts, the way I use or don't use plans, how I manage things remotely, which is a really fun angle I want to dive into as well. And most importantly, I would argue, how I actually think about the poll requests that I am making when I build with AI. I want to give a bit of context on where my experience is coming from here because the way I've been building and the things I've been building has been very different recently. I took on a pretty bold, pretty stupid project called Lakebed.

It's a new full stack framework, a new runtime, a new backend, a new database, a new cloud, a new lot of things to try and make it easier to build quick apps with agents. The point is to be a shitty cloud for shitty apps. But this means I had to do a lot of different things. Thankfully, I was mostly working on it myself, though, which balances out some of the chaos I put myself through when making it. But it gave me a lot to reflect on in terms of how I want to build with agents now. And I've shifted a lot of my perspectives through the experiences I've had here. When working on LakeBed, I tried a lot of different tools. I used cursor a little bit. I used the codeex app a lot. I used T3 code. I tried using cloud code a good bit as well. I even played with open code and some of the cool new like open source models throughout. It was a very eye-opening experience to see how those different models could work in a codebase like this, but also more importantly, how they handled different prompt styles, different harnesses, and different agent MD and Claude MD files in particular. But I want to answer this first question around models. What models am I using? Now, obviously saying anything about this is going to make the video incredibly dated. In fact, by the time this video is published, it's possible that a new state-of-the-art is out. But at this point in time, I really struggled to use anything other than GPT 5.5. I will explain more on why I'm enjoying 55 so much now, especially compared to how little I enjoyed it in the past. I've pretty much entirely stopped using clawed models. I'll occasionally pull one in to like make a quick landing page for me. But for the most part, I'm just using 55. I do think Composer 25 is really cool, but I have effectively unlimited inference on 55 because I'm on the $200 a month plan and I still am under that weird 10x thing they did for people who attended the 55 event. So I despite my best attempts cannot get these numbers down. They just did a reset that they probably didn't even need to do, but the worst I could get is about 6% down on my weekly usage when building a full cloud from scratch.

So yeah, I think the limits are very generous on that plan. They are 10x more generous for me cuz I'm on that weird things of the event. I don't know how drastically that will change when the 10x is gone, but like the worst I was able to do damage wise would have been the equivalent of 60% with the normal limits. So I don't think it's a big deal. That said, the cheaper plans are much easier to hit limits on, especially if you use some of the cool new features in the harness. Harnesses are a much more interesting conversation in various ways. If you're not familiar with what a harness is, you should definitely watch my video on how cloud code works. It is an in-depth overview of what it takes to allow an AI to use your computer to edit code and actually do things for the work you're trying to do. To be very very simple with it, the harness is the set of tools and the actual application runtime, whatever that allows an agent to do things on your computer to edit code and whatnot. Most harnesses come as some form of CLI, whether that is the harness in cloud code or the harness in something like codeex, but some aren't necessarily focused on that. They are more like an SDK. The cursor harness, for example, is more in that direction.

They do have a CLI for it, but yeah, there are a lot of really cool harnesses nowadays, especially in the experimental space. Things like PI are really, really cool, and I do need to do a better deep dive on it, and I haven't yet. That said, as part of me using 55 more, I have found myself pretty much always defaulting to the codeex harness. There are a lot of things that Codex has been doing really well lately. One of the biggest ones is that they've kept the CLI boring. They're not loading it up with all sorts of crazy stuff. The CLI has just kind of stayed a simple, minimal, boring CLI, but they have been building really cool things into the Codeex app. And a lot of those do carry over to the CLI as well, which means they carry over to other things using the harness. Like, you know, my favorite way to actually build these things, T3 code. A lot of you seem to be confused about what T3 code is. T3 code is not a harness. It is not able to be paid for, and it is not, most importantly, T3 chat. T3 Chat is a different thing. T3 Chat is an app for chatting with AI agents, similar to chat GPT. T3 code is an app for managing your other AI harnesses, similar to something like conductor or kind of cursor, but really it's more similar to the codeex app. I'm very proud of what we have built with T3 code. And by we, I mostly mean Julius.

He's done the vast vast majority of the work. I have put a lot of time into the codeex app, though. I was using it for most of the building of Lakebed because I wanted to just do another deep dive on the competition, see what its strengths are, see what its weaknesses are, and get a rough idea of where things are at. As I mentioned before, the Codex app is really good. And if you're just using the GPT models and you're happy with Codeex, totally fine to stick with it.

If you've never tried one of these styles of agentic idees where you have multiple projects open at the same time, you have threads that are easy to swap between onset projects. This style of building is really, really nice. When anti-gravity added a view like this in its original release, I started to see the potential value in this way of building. Since then, I've quadrupled down. This is how I build. I only really open up editors to edit environment variables now. So, if I do actually use cursor, which I'll be real, I don't use it on my computer very much. I usually use it in their new cursor glass UI, which has for my experience been very broken and laggy. Apparently, it's getting better. I still have not had a good experience with it. The codeex app is very, very good and is probably the best bet for most people right now.

Obviously, I'm biased and I find T3 code to be much more pleasant, much more stable, much more reliable, especially with remote stuff. But realistically, codeex is what most people are going to experience. It's a great experience. I will also say if your only experience with an app like this is the Cloud Code desktop app, then you've not experienced an app like this because the Cloud Code desktop app is like a third class citizen at Anthropic. No one there really uses it. No one there's really trying to make it great. It's been stapled into the cloud desktop app in a way that isn't very pleasant. The Codex app is how most people at OpenAI interface with Codeex. In fact, I've heard many more people who aren't even technical using it as an alternative to CHBT itself. The Codex desktop app is the best way to use Codex. If you've only used it through the CLI, you've not really used Codex. So, I'm using T3 code. I would absolutely support people for choosing Codex instead. There's lots of other cool options to consider like conductor. My main gripes with conductor are that I had a really rough time trying to give them feedback. I was surprised how quickly my team got ghosted because we really wanted Conductor to be good. It's also closed source, which means if you don't like anything about it, you're kind of stuck.

Whereas T3 Code is entirely open source. So you can do whatever you want to and in fact a huge portion of our users are running forks of T3 code. It is also worth noting that if you're primarily using a claude code subscription, we are about to get heavily limited because they marketed a [__] change where they started giving you credits when you're a subscriber. The credits are what we would use instead of your actual subscription limits, which is absolute [__] If you use Cloud Code via the Claude Code CLI or the Cloud Code desktop app, you can get up to $5,000 of usage for 200 bucks a month. But if you use it in something like Conductor or T3 Code or if you SSH into a computer and call Claude-P instead of just the Claude CLI directly, then you only get $200 of usage. And if you go over that, it costs you money directly. It is a [__] change.

I can't believe they actually did it. I have a whole dedicated video on that. But again, we can't do anything. It sucks that we can't do anything. We will probably have a crappy terminal UI that will open when you use cloud code in T3 code because there isn't really another option for us. Blame anthropic. We can't do anything. We do support everything else you might want to use though. We support cursor and open code. You can enable them in settings. They just use the existing CLIs through ACP. It's nice. It's really cool having a tool like this that supports everything in one consistent cohesive UI, especially once you get into the remote stuff. I know I'm skipping some steps here, but I really want to emphasize this when we're talking about the apps because I think it's one of the most important things. I know what a lot of y'all are thinking.

Why don't you just SSH into a computer instead? Cuz like the problem we're trying to solve with remote control is what happens when I shut my laptop and an agent is running. I want to open and see the result. I don't want it to get stopped. I don't want to have the meme where I have the laptop half open as I walk around. I just want to be able to run the thing, close my laptop or go offline or code from an Uber and then have it updated reliably. I really thought Codeex would have this figured out. And to their credit, the mobile integration is largely there. The fact that there's this little button you can click to set up your phone through the chat GBT app to control codecs on your computer remotely is actually really, really cool. As I hinted at before, I like to code on machines that aren't this laptop, but I still like to control it from this laptop. And on this local network, I have my Mac Mini, and I try to control it from the Codeex app here with Codeex running on there. This is an actual Mac Mini running on my network that I can connect to remotely over like the Mac screen share stuff. I have

Codeex on here open and ready to go. I also have T3 Code open on here ready to go. I spent the last week trying to do everything through codecs. God damn did I hit a lot of issues. The mobile integration is mostly fine, but I'm just going to quickly go over some of the insultingly bad problems I had trying to control it remotely through the desktop app. The first one, and this was like an absolute stopper for me. Occasionally, the model picker would disappear. And when that happened, even though I was connected, I was unable to do anything on the remote machine. So, I had to connect back over to it via the screen share, close and reopen the Codeex app, come back here, close and reopen the Codeex app, and maybe it would work.

Like, maybe. Usually not, but sometimes it would. When it does work, it still has all sorts of fun issues. when you actually just like send a prompt usually responds faster. But do you see how slow that was to pop up like the history? It's really bad. And where it gets even rougher is if you have to open up the terminal for any reason. I'm going to type 1 through nine just by sliding my hand down the keyboard starting now. Oh, it actually did it kind of fast that time. It's still like super sticky keys.

Like I am typing as I speak. When I was trying this yesterday, there was up to a 30 second to two minute delay. I cannot stand. God, it's sticky keys so bad. Even when you type correctly, it's it's unusably bad. And then when you try to paste an image, it's like 5050 if it works at all. I have not been happy with the remote experience outside of mobile on Codeex. I tried it because I wanted to try the mobile stuff. Was pretty impressed with that. Assumed the desktop app would work well and it just didn't. Then my team reminded me that Julius went really hard on the remote stuff for T3 code. So I decided to try that instead. And I'm actually trying this a few different ways. I'm trying the baked into the app remote version, but I'm also trying the remote hosted version that was very easy to set up. I hop over to Helium. You'll see here I have my T3 code instance over my local network fully functioning. I can hop here. I can go to an existing thing. I can open the terminal. It types full speed. No issues because it's effectively actually sshing over. It has been significantly easier to build.

image paste work perfectly and I actually love having it in the browser because now this is a different thing from my local instance. I don't have to like remember which ones on which machine and deal with all of that. It's been very very nice. One of the coolest examples I've seen of somebody pushing the limits of the T3 Code remote stuff is Jack here. He doesn't have a computer. He does most of his work from an Android tablet. So what he did in order to get T3 code working on his tablet is he actually spun it up as a T3 code server in Replet. This gives him the ability to use the T3 code web app hosted on Replet to build projects remotely on Replet, which I think is really really cool. Point being T3 code remote hosting is great, especially if you combine it with something like Tailscale, which allows you to connect to devices on other networks very trivially without having to expose it to the whole web. It's been very nice to work with. I have been blown away with how stable it is. I've used this for the majority of the changes I've been making to LakeBed for the last day and a half.

And I'm also using it to spin up new projects remotely, too, which has been very surprisingly solid. The one catch here is mobile. We do have mobile web, and it works better than I would have expected. Julius is cooking a React Native app now. We should hopefully have that pretty soon. Fingers crossed. And if you do want to test this out yourself, you can just go to settings in T3 Code, hop over to connections, and relatively easily set up tail scale or network access remotely or on your existing network very easily. You can even do custom SSH

connections even though in codecs they're a bit rough. We actually have a UI to store a password when you set it up yourself in T3 code.

As I mentioned before, Julius went really hard on this stuff because he really likes these remote flows. And the result is that it is one of the best experiences I've ever had doing remote coding with AI by far. No longer am I stuck using tools like Termius to SSH into some shitty Linux box and try to make a terminal UI work on my phone. I still can't believe people are doing that. I will crash out briefly on the SSH terminal people for a second because I need you guys to understand how much you're suffering when you do that. First off, you have to deal with T-Mux or Zelj or GNU screen or something. So if you disconnect, you don't kill the work that you're doing. Second off, you now have to have a bunch of weird key bindings or some other abstraction to switch between different threads you're working on. God forbid you want to go do something in a work tree. Good luck. Have fun with that. You're going to be writing a lot of handwritten git commands to spin up that work tree. And then what happens when you want to paste an image? I would estimate that a third to half of my prompts have an image in them. Whether I'm just quickly grabbing an error screenshot and pasting it or if I have some UI that I want to throw over to the agent to tell it like, "Hey, can you make this better? Hey, can you make it look like this other thing?" I use images a lot in my prompts. The fact that they kind of almost work in traditional CLIs and then don't work at all over SSH is insulting. I've been fighting this fight for a while. I as a person who spent the vast majority of their computer time in terminals throughout their life, I really really don't like doing agent coding in terminals anymore. I'll still do it occasionally for like quick demos or like editing things on my computer. Like if I'm trying to config some directory or like fix my dot file, stuff like that, I absolutely will still use codec cli. But when I'm trying to do real work in a real codebase, I want the app every single time. I still can't believe the pasting images over SSH thing is as absurd as it is. And while it's cool people are like building raycast workflows in order to upload an image easily to send it as a URL over for the agent, like it's cool, you shouldn't have to do that. The amount of workarounds I've seen people make just to be able to use cloud code over SSH is hilarious and painful and no, I I I will not support that mental illness. Please try a good desktop app for agentic coding before assuming the terminals are the only solution because you tried the codeex desktop app six months ago and it was [__] It's still [__] I understand if that's the only experience you've had that you think CLIs are better. They're not. A good desktop app for coding will [__] all over a CLI any day. As I mentioned before, working remotely is really, really nice. Especially if you're like me and you find yourself randomly having to like leave your office to go to some event or go to some coffee shop meeting or you just like want to check in on your work as you're pacing around the office. I really like being able to do work remotely and I find myself spinning up more and more of my work remotely. when I know I'm going to be sitting at my desk for a while, like when I'm streaming or when I'm working with my team, I will still often run things in T3 Code locally, but almost all the time otherwise, I am connecting to that remote Mac and doing things through that. But now I need to talk about how I actually do the work in these tools. Cuz thus far, we've just been focusing on the tooling side. And in order to understand why I like 55 so much and why I like remote coding so much, I think my actual ways of working with the models are worth understanding more. Context management really is the name of the game for getting these things right. An underrated trick that I found really useful is giving the agent the ability to explore other code bases that might be relevant. In this case, I was trying to set up O for the userfacing apps people would deploy and generate with Lakebed. So I needed it to have a good O solution. I wanted to see if my implementation for my O service I built before called Shu would be a good fit. So rather than like describing it or throwing it at the docs, I just cloned down the repo and told the model take

advantage of the O implementation I have in Shu with a link to the path on this computer with that implementation so it could use that as a reference point. This type of context manipulation results in much more reliable outputs from the models and has been one of the biggest improvements I have personally experienced. Even just telling the model like go clone this repo and throw it in some scratch directory in order to figure things out. There's one other super reliable thing I want you to know about though. Our sponsor agents can write surprisingly good software as long as the problems they're solving are simple. There are certain things that just aren't though. You know, like DNS, the thing that sucks for everybody. Be really nice if DNS was simpler. Oh, Dian Simple's on the screen, isn't it? I I love these guys. I've been so blown away with every interaction I've had with them, all of the cool things they do.

The fact that they made a good SDK for managing your DNS is incredible. If you're trying to build services where users can like register domains or subdomains or manage masks and set up forwarding and do all these types of things, good luck doing that programmatically because the APIs that exist for it suck unless you're using D and Simple. Like, how cool is it that I can call client.registar register our check domain to see if a domain's available and then buy it all from just writing TypeScript. This is so useful that I wish I could do it through a CLI.

Oh. Oh, is the CLI on the screen now? Yeah, this one was so cool. I did a call with the guy who made it because I was blown away and I wanted to give him some feedback to make it better for agents because having an agent able to run a CLI to debug DNS issues is like a thing that I would have killed four years ago. And now that I have it, it's like, oh yeah, obviously the CLI is no joke. This is a full-time project for one of the engineers there and he went hard on it.

Everything you would do through the SDK is available and it's all also exposed with the help commands so that your agent can see how to use it. You're too busy to be debugging DNS. Let your agents do it for you at soyv.link/dnsimple. Here is the original thread where I started span aka lakebed, my new cloud framework everything project that I'm very proud of. You might notice this is quite a blob of text. The reason for this is that I used voiceto text. I found that using whisper flow or other voicetoext tools makes me write much better prompts and this one was meant to be a very much thought dump like I wanted to plan out with the model especially for these types of large changes. I do like planning and I want to be clear about something. I do not mean plan mode. I like to work with the model and not be scared of letting it write code or make changes or test things. And plan mode is a little too restrictive. And there are problems here. Like I've had times where I wanted to talk to the model about a thing and it just went and did the thing. 55 is really guilty of this. But again, I prefer working this way where I'm not necessarily in the plan mindset and then in the edit mindset. It's more of a natural back and forth. So here I started with a very important thing. my end goal and I find that this is the thing most developers miss. You guys love focusing on the details and I understand why that's been what mattered our whole career. Especially when you're advising more junior engineers, it is important to be detailed about how the thing should work, not just what the thing is. I found myself moving over to this more higher level like here is what I want it to do, not how I want you to implement it. And I started with roughly that this directory is for a new project. I want to start called span.

The goal is admittedly complex. I want to rethink how clouds work from first principles. I want to build a new full stack TypeScript framework that is all the pieces you need to build full applications including simple, minimal, reliable offlayer database synchronization, file storage, and more. The goal is to make everything you need available via the code instead of having to go through other layers of platforms. If a user has to open a dashboard, we have failed because I want this to work for agents in all different ways that they would need to initialize projects.

The formatting got all screwed up cuz I was voice detecting. I wasn't even paying attention. I thought this was going to be a throwaway, but the fact that I did this as a throwaway and it went so well shows how powerful these tools have gotten. To open, I said I wanted first for it to roast the plan and give me all my thoughts and feedback before we proceed. Do whatever research you need, yada yada. Says, I love the ambition, but the roast is simple. This is not one project. It's a runtime database, sync engine, object store, deployment control plane, local emulator, security model, migration system, observability stack, and agent interface wearing one trench coat. The idea is viable only if span starts with a brutally small thesis. It tried to insist that things like convex instant and jazz meant it wasn't necessary.

Also, Cloudflare being as good as Cloudflare is yada yada. I had to slowly convince it. The important thing here, and I know this is hard for people, you have to read what it says. I know people tend to gloss over the text the model puts out and instead read the code it puts out or maybe they'll even read the plan it puts out but rarely. I find that devs have this instinct where they care more about the code output and not enough about what it said and that's entirely backwards. You have to read what it says. And if you don't like how it's saying things or it says too much, steer it the way you want it to talk instead. Tell it to be more brief and concise. Tell it that it wrote way too much [__] Tell it to format things in the ways you want to read. But you need to get the model talking to you in a way you'll actually read what it says. Mark is a victim of this. And I was planning on yelling at him about this when he was here and I forgot. I will do it later.

You got to read the text, especially when you're doing big sweeping changes. And accordingly, you have to respond to the model based on what it said. So, I went through here and typed by hand the different sections and my thoughts on what it had to say, saying, "Here are the things that I agree with. Here's the parts I don't agree with." Trying to get the context of this thread to be more in the direction I want this project in.

You got to kind of treat it like you're convincing somebody of your way of wanting to do things. I put the work in to convince the model to do the thing I want. I see a lot of questions in chat already, like, "What skill command or framework are you using to get the grilling? What skills do you have set up? Have I tried the superpowers plugin? All this shit." No, you're all coping. You don't need all of that [__] I have almost zero skills installed. Just talk to the [__] model. They're smart enough now. I don't even have the super small and useful grill me skill for a matt set up because I just have it as a binding in Whisper Flow. Grill me skill.

I just hold down my whisper flow key. I say grill me skill and it just pastes the exact markdown into the input. So I can just do that and it will do the same thing. Super easy. Even then I can just tell it to and it usually does it. You guys care too much. I don't bother with skills usually. After I gave my feedback, the model agreed more.

This context makes the idea much sharper. I buy it more now. The category isn't new cloud. It's an agentnative app substrate for tiny fullstack apps. Yes, it got it now. And this is why you talk to the model and you read what it says. You need to make sure you and the agent and the context are on the same page. And this is the biggest thing I want you to take from this video. The most important thing when you're building with AI is that the AI understands what you want and how you build. You can do that by writing a 5,000line agent MD that's global on your computer with all of the things you like and do and such. You could also become a famous influencer and tell the model, "I'm Theo, build the way I like to build." which works sometimes, but the easiest thing to do is to just read the outputs and steer the model in the direction you want to go in. And if you notice it making the same mistakes over and over again, go into the agent MD and try to give the model your psychosis. That's what I've done. And I think that's one of the strongest things I did that made it possible for me to make this project so quickly. You'll notice that in this Lakebed Agents MD, there are no file paths. There are no technical decisions or enforcements. There's a couple small general rules at the bottom that I might even delete because I haven't found them to be super useful. The point of the agents MD, at least how I use it now, is to make the model more steered towards what I'm trying to do. I wrote this one almost like a letter from me to the agent to tell it how we're thinking, what we're building, and why we're doing this so that it's less likely to have bad assumptions or ask weird questions or work outside of the technical constraints that I want it to work within. This document has helped so much. I noticed almost immediately after writing this by hand, by the way, my agents did not write this file. I wrote this file. After I wrote that, I found that I didn't really have to do much to get the agent to build how I wanted it to. Once it had the context of how I was thinking about this, it started behaving way, way better. The craziest thing that I did throughout this project, and I know this is going to be hard for a lot of you, when the agent pushed back, I listened. When it said certain things were not necessarily the right idea or were too hard to justify, I listened and I delayed those. And one more hack that I found really nice is one that comes from our friends over at Enthropic.

Having the model write an HTML file for the plan. It's so much nicer to read. I found this much easier to like read and go through the whole plan and give feedback and answer all of the remaining questions. It was so nice. That said, the first HTML page it made was horrible. It looked awful. So, I had to like yell at the model a whole bunch about that. Also, apparently in the Codeex app, the questions tool just didn't work in the remote mode because at this point, I was controlling this from my phone because I was busy and it couldn't ask me questions. It might be that mode. It might be the not using play mode. I don't know what it was, but it was insisting it couldn't ask me questions. So, I had to like go through and answer all of them directly. And then I was like, the UI for the pages is awful. I got it to clean it up. After a while, it like still was full of useless crap. I gave it as much as feedback.

This is also another trick that I found really useful. Get a screenshot tool that lets you actually do [__] Like being able to point an arrow at a thing and say, "This sucks." And makes it so much easier to steer the model to make the right changes with things. I press control C, the thing goes away. It's on my clipboard. I use Shotter. It's great. There's lots of other options that are good, too. Get a good screenshot tool.

It makes life so much easier. Once I got the HTML in a good state, I found that every plan I did from that point was really nice looking because it would just look at the existing one and be like, "Oh [__] I'm gonna copy this formatting." And this is like one of the coolest things is once you get the agent to behave how you want if there

is enough proof of that behavior in your codebase whether that is HTML plans whether that is your agents MD steering it certain ways whether it's the code itself once you get the model to behave how you want it to everything else almost stops mattering and that's one of the coolest things to discover as I've been using 55 like in its default state copying the prompts I used to use with the big bloated useless agents MD 55 sucks. When you take the time to condense and steer it the way you want it to work, it becomes the coolest way to work I've ever built with. And here's where we're going to get into the other things about how I built. You might have noticed I have a lot of threads here.

These are just the ones I did in the codeex app. I have another few dozen in T3 code all for the same project that I did in 5 days. I probably started over a 100 threads on this one project in 5 days. And I know what you're thinking already. Oh, so you have all of those running in separate work trees and you're hopping between 15 of them. Cool. Everyone says they do that now, but there's no way you're productive. Nope.

Almost every single one of these threads was run by itself on main alone. I found that I am just genuinely way less interested in these more parallel workflows lately because it's just too much context to keep track of. And when you spin up a smart enough model, especially if you're taking advantage of fast mode, which I've surprisingly been using a lot, it's not worth the price increase. But if you're not getting close to hitting your limits on the Codeex plan, it's very nice to use. The fact that fast mode is included on your plan, it just increases how fast you go through your limits on Codeex when it costs actual money on cloud code is hilarious to me. But yeah, I've been using fast mode on my codec subscription. I've never come even close to hitting my limits. It's been very nice. So much so that I don't even find myself leaving extra high as much. If I'm just doing UI stuff or I want it to respond faster, I'll hop over to low.

But I've gotten good enough at keeping extra high on tasks that I found it fine. You'll notice a lot of these threads are literally just one prompt. This one was very simple. What would it look like to let users bring environment variables for serverside code? Ideally, they'd be able to update aenv.lakeb.server file and run npxlbed deploy to push those environment variables into the cloud for their deployment. A minute and 54 seconds later, it wrote a whole model for how this should work that I thought was great and it really seemed to understand what I wanted here. The important design choice section shows the name of the file. It said this should be the source of truth with replace semantics. If we add something in deploy, it gets created and updated.

If we edit something locally, the deploy will rotate it. And if we delete something from it, then lake deploy will delete it. Exactly what I wanted. I didn't have to get more specific. I didn't have to write a longass prompt. I wrote two sentences and then it speced out exactly what I wanted. To which I responded, "Love it. Build it." 10 minutes later, the whole thing is working exactly how I wanted. No additional changes needed to be made. I just pushed it to GitHub and I was done.

It was great. And I started my next thread, which was I want to be able to manually bump rate limits for a given user. This is cuz one of my friends who was trying it out was loving it and wanted a higher rate limit. So, I just told it add this feature. And I gave it a little more detail here. I want a new users table in the admin dashboards. I want to be able to control this myself with the ability for me, specifically me, to set custom overrides for

someone's limits. Another important thing, and I had this in my agent MD, I didn't talk about it before. A glossery of terms and language to help the model understand what you're saying, can be very, very helpful. I found that with this project, it was difficult because there is me, the person working on this thing. There's also me as a user of the thing. There's the agent building lakebed and then there's the agent using lakebed to build apps for you the user of lakebed. So I gave it these specific terms where you is the agent that is actually going to make changes. Me we and us are the humans that are building lake bed itself. Developers refers to our users people who are going to build things on top of lakebed and then agents which is the thing the developers are using to build with this. Again, the point here is to make it easier for the model to know what I'm referring to as I discuss these things. Is another one of the themes you'll notice. Generally speaking, you should try to keep things simple. And if it doesn't work, that doesn't mean make it more complex. It means fix the things that prevent it from being simple. If talking with the model in plain language was preventing it from getting what you're saying, you shouldn't get way more oversp specific in all of your prompts. You should make slight changes to your agents MD and to your claude MD so that you can keep your prompt simple. And you'll notice almost all of my prompts here are two sentences or less. Oh no, this one was three sentences in a list of things, but it did exactly what I wanted the first try.

And you'll notice this looks almost identical to how the homepage looks now. Nice and simple. And as I said before, none of these threads were run in parallel. I would do a task, I would complete the task, and then I would make a new thread and start the next task. And I would just do that over and over again. As Morg just said in chat, make the difficult change easy, then make the change easily. Yep. Why new threads? I always make new threads because I don't want old context getting in the way. I treat every thread as a pile of information that is steering the model.

And if I'm doing something different, like this thread, I was working on environment variables and then this thread I'm working on user limits. These are different concerns. and having the same thread with different concerns within it just biases the model towards things that aren't necessarily correct. Remember the way these models work is a bunch of parameterization. All of these sets of characters that are in your history that are in the model, your history changes what points to what in the model. So the more stuff in your history, the more customized the model is to an extent. Every additional word in your chat history is changing how the model behaves. I don't want to deal with that. And to those saying, "Wait, does that mean the model has to explore the codebase every time?" Yeah, it does. It doesn't [__] matter. It does a great job. It still completes all of these things in seconds. And I find that in real code bases, there is so much stuff going on that the history of your previous change is just going to confuse the next one. Also, chat pointing out they notice that I'm not mentioning file specifically or applying skills for it to work on. Correct. I'm not doing anything more specific. I trust the model to find the right file. I am more likely to recommend the wrong file than the model is half the time. If it turns out the file that I set isn't necessarily the right one for the change, the model's going to get confused and try to make it work in that file. If I have a good dev on the team, I'm not telling them what file to edit.

I'm telling them what to do and they'll figure out what to change. And I would find half the time when I look at the diff for the changes it made that I am surprised what files it ended up changing. Again, don't add details unless it needs the details. Try to be more sparse with your requests and prompts. Figure out what it doesn't

understand and figure out how to fix that without making your [__] too complex. Yeah, I'm just looking through all my threads here. Almost all of them are actually just two sentences. This was a voice to text, which is why it's a bit longer and I need a little more context. What I wanted there. Yeah, for the most part, all of these are very simple. Here's an example of me sharing a log screenshot instead of actually copy pasting logs cuz it's so much easier to do. One more pro tip. When you have ideas that are a bit complex or the model struggles to understand it, don't overexlain. Just give examples. The more simple an example you give that contains the problem you're trying to solve, the better job the model will do at solving it. So this example I was trying to discuss custom domain flows for LakeBed, which coming soon. I have a lot of layers to fix here to get it right. I gave the example of I have a project on Lake Bed and gave it an actual URL to an actual lake bed project I had deployed.

I have a domain on Verscell T3.gg. I've configured a CNAME for lake bed demo T3gg where the value is this. What's the best path to make this work? Handling SSL and whatnot. This is to tell it like specifically what my question is, like what my where my problem is located. And then the goal as well. The goal is to be as easy as possible for our users without massively ballooning costs. It's silly, but I think this is a great example of a prompt for something complex. I said what I wanted. I gave it a very clear, concise example so it would understand exactly what I wanted to do. And then I steered it towards the parts that I wanted the most help with and gave it specific goals in order to keep it within my constraints. Gave me a bunch of info. It did not give me enough info about costs though, so I asked it very specifically, give me a breakdown of all of the costs I would incur by going with your proposal. And then it did exactly that. I didn't like the proposal. So I got more specific with an exact flow I would want the user to go through. This example I say the user would run this command npx lake bed domains add demo 1.t3.gg. Then the CLI would output to add the domain set the following records and you go set these records and I specify I would go and assign those values wait for lake to pick up the change issue SSL and good to go. ask more questions for more clarification. But I really just use this thread to figure out the how to do this. What I would normally do at the end if I was ready to go with this is I would ask it to write down a simple plan based on what we discussed that I could have as the markdown or the HTML plan that I would read, confirm, and then hand off to a new thread to go actually build it. Once you actually get the model to start doing the work, one of the most important things is to give it the tools it needs to verify that the work was done. This could be CLI commands it can run. This could be a test suite it writes. This could be computer use where it actually goes to the page to see if it's correct or not.

This is one of the things I have found to be the best about using Codex. Their plugins and specifically the computer use stuff that they have built is really really good. Codex's computer use will let Codeex control apps on your computer. It'll let it control a full browser with an extension and it can even do it when the computer's locked now, which is really, really cool. I've been blown away with how useful this is. My issue is that I don't like it working on my computer cuz I want to use my computer when the agents are running.

So, again, this is where the remote stuff got really nice. I've also noticed that once you set these things up in the Codeex app that you can use them through the Codex CLI, which means you can use them through T3 code. So I set up all the computer use stuff in the codeex app and then I went back to using T3 code remotely and now my T3 code can verify changes by deploying an app on Lakebed and then going to Lakebed to see if it actually deployed or not and make sure it behaves as expected. It's so powerful and I found that it makes the likelihood

that by the time the agent pings me that it's done that it actually did it correctly is way higher. Throughout this whole project, I would say of the like 50 plus threads I've done, maybe four or five of them didn't do what I wanted first try. The rest all behaved exactly how I expected them to and it worked very well. And also to be very clear, I'm not using the new like goal skill feature thing inside of Codeex. It seems really cool. It didn't work in the app for me when I tried it and I don't use the CLI a whole lot and certainly not for these long running things. And most of the stuff I'm doing, even the stuff that seems really difficult, ends up being done under 10 minutes, especially on fast mode. Here, I overhauled the runtime for anonymous deployments and it took 7 minutes. Code Rabbit found issues, so I screenshotted the issues in the Code Rabbit PR, told it to fix them, and it did. And then there was conflicts on Maine, so I told it there's conflicts with latest Maine. Please address them and push up your changes when done. And then it did. And then I merged it. and then it was good. If your takeaway so far is that this is surprisingly simple, you have the right takeaway. It's very simple. The simpler your flow, the better. Do things the stupid easy way.

And if it doesn't work, figure out how to make it work. I want to talk a bit about my PR flow now, cuz I think this is important. When I start work, I usually start it with a rough idea of how complex the change will be in my head. It varies a lot depending on what I'm doing. Sometimes a task ends up being a lot more complex than I expected it to be. If I'm unsure of the complexity of the task, I'll usually start by asking the model for its thoughts and we'll have that back and forth. When I have a good gut feel of how big the change is, at that point, I'll often decide if it's worth doing on the branch I'm currently on and in the work tree I'm currently on, or if I should go make a new work tree to finish up this task. Sometimes once I'm halfway through the planning process, I realize this is going to be a lot more work than expected. So I just copy the initial prompt. I go spin up a new thread in a work tree. I paste that prompt with a couple additional steering comments to make sure it doesn't go down the bad path I was going down earlier. This has worked very well for me. That said, in this project that I've done hundreds of commits on, again, it's a solo project, so it is different in that sense. I have not actually found myself reaching to make PRs that often and I even closed two of the ones that I made because I didn't find myself needing it. when I made big enough changes that I really wanted to sit with and get second opinions on, things that are security related, things that change the hosting layer, thinking through what changes need extra eyes, not just like a human looking at it, but other agents or code review tools looking at it, like building the intuition for what changes would benefit from that is really important. And sometimes you'll put up the PR and something like code rabbit, macroscope, grapile, whatever you're using will catch a bunch of issues and then you tell the agent to fix them and it does and then it catches more. At that point, I'll just tell the agent to run in a loop until it resolves all of the issues. Usually using something like the CLI for code rev, which has been very helpful for this type of thing. I had two or so of these where there were just so many pieces of feedback that I didn't want to keep copy pasting back and forth or telling the agent go check the PR again over and over and instead just told it run the CLI until you don't get any feedback. And that worked great.

That ended up working specifically for a lot of the ownership stuff that I was trying to get right with this project. That said, in real projects with lots of developers, you're gonna start running into problems with PR bloat, especially, and this is one of the coolest, but also the most painful things in a tool like T3 Code. We made it a bit too easy to make a PR. You can spin up a new thread, make changes, and then click one button to commit them, branch them, PR them.

Super nice when you're using PRs as an artifact for review. Not so nice when you're accidentally spamming your projects with a bunch of PRs that'll never merge. I do love the one-click. I file a lot more PRs when I have that feature. Getting a good workflow in codeex for turning your changes into a PR was annoying. So yeah, this was very very helpful. But sometimes those PRs will just sit around and you don't know if they're worth merging or not anymore or even looking at. I've had that happen a few times even solo on this. And again, bring in the agent. I noticed one of my branches kind of got stale and had a lot of conflicts. So I brought this branch up in a new thread. I asked how up to date is the theo/aban overhaul branch. Compare against the latest main.

If these changes are still worth merging, fix the conflicts and push up the finished branch. It inspected it. The work tree is clean and currently on a local helper branch branch is significantly behind. It has 49 commits not in this while the admin overhaul has two commits on a main diff concentrated reading the changes. And at the end here it said after inspecting both sides the branch's useful works already been superseded on main. Maine already has the admin user detail route. This route user detail UI and smokeote coverage for the admin user shell API. Board of the merge and did not push because pushing a fixed branch would just add noise for changes that are already represented in newer forms on main. Recommendation close and delete. Awesome. Generally it is important to try and keep PRs from getting too stale because it gets really bad really fast. Take it from us with 414 open PRs on T3 Code right now. It gets bad. I think I covered everything I have to here. I know this video is a bit chaotic, but I kind of just wanted to show you the chaos. What does it actually look like with real projects I'm working on? That's why I showed you the actual threads that I was using to build real projects that I'm deploying now. I've never been happier with my flow, which means it's probably all about to change again. And I'll be sure to do another updated video if these things shift over time. But for now, just try to keep it simple. I find most developers are trying too hard to engineer their workflows. And I get that. I love engineering my stuff, too.

But generally speaking, keeping it simpler makes things much better. And the fact that I have made my set of tools here and my way of thinking about things here so simple has made me way, way more productive as a result. And while I do dearly love T3 Code and what it enables for me, especially around the remote stuff, the Codeex app is fine. Just like stop using sidebars in your IDEs, guys. Stop using CLI. Stop using these things that make it too hard to just start a new thread or start working on a new change. If you're looking at the code more than you're looking at the conversation about the code, you're already behind. It's time for us all to let go a little bit and try to build the way the AI is strongest, which is with a good conversation that builds the right context for the model to go do the right thing. I know this is a bit different and I really hope it was helpful for y'all. Let me know how you guys feel.

And until next time, peace nerds.