A proper guide to Fable 5

43 MIN · YOUTUBE · HTTPS://YOUTU.BE/8GRMLR__OGQ?IS=9AOBVJGRTAXBWS77
https://youtu.be/8GRmLR__OGQ?is=9aOBVjgrTAxbWS77

SUMMARY

The speaker shares their transformative experience using the Fable model for coding tasks, highlighting its superior capabilities compared to previous models. They describe how Fable significantly increased their productivity, allowing them to complete a month's worth of work in just a few days. The speaker emphasizes the importance of adapting workflows and prompts to fully leverage Fable's potential, while also addressing common misconceptions about the model.

- *Fable model drastically improved productivity, enabling completion of extensive coding tasks in a short time.*
- *The speaker experienced a significant drop in work efficiency when Fable was unavailable, struggling with previous models.*
- *Fable excels in end-to-end implementations, testing, and task management, unlike earlier models.*
- *Effective prompting and workflow adaptation are crucial for maximizing Fable's capabilities.*
- *The speaker created specific workflows and sub-agents to streamline project management and code reviews.*
- *Cost management is essential; using lower reasoning levels can reduce expenses significantly.*
- *The integration of other models, like Codex, enhances Fable's functionality, especially for complex tasks.*
- *The speaker encourages experimentation and customization of workflows to improve individual coding practices.*

Saying Fable 5 has oneshot me would be a criminal understatement. I really appreciated the model the first day I tried it, but over the next two I started to genuinely love it and was blown away with what it was capable of. And then I got taken from us and I I struggled. I did my best to make decent work come out of Opus 48 and GPD55, but it just it wasn't the same. And I found myself just kind of running in circles waiting. And then the model came back and I honestly can't believe I'm saying this. It's better than I remembered.

There are a lot of things that aren't perfect about it. And there's ways to work around some of them, but god damn, when I'm actually trying to use this model for my day-to-day work, it blows me away. In the first day of having it back, I got more work done than I had in the month prior. It just it was incredible to watch as the model cranked through all of these things that I've been planning and thinking about and starting but not finishing. And it just shipped. It fixed a bunch of code. It merged a bunch of stuff. And it got me pumped for not just the work I was doing, but for the future of how we'll be doing work. It is incredible what you can do with this model, but there's one big catch. If you treat this model the way you did previous ones, you're not going to see the benefits. To put it simply, this model isn't a better opus.

If you take prompts that worked for Opus and you give them to Fable, it's not going to be much better. The difference in this model isn't just how much smarter it is or how much better the code is, it's how much further it

can go. Not just on like harder code tasks, but on end-to-end implementations, testing, verifying, and all the other pieces you need to write great software, especially once it has to break tasks up into smaller pieces and hand those off to sub agents. This model does incredible stuff, and I've been doing my best to get as much as possible out of it during this limited window where we have it in the subs, although it is coming back soon. And with that, I do want to make sure you guys know I already filmed a video today about a lot of this, specifically about all the misconceptions people have with the model. They're frustrating me because this model is great. So, if you're confused because you read Twitter and saw all these people saying the model's nerfed, it's terrible, it's way too expensive, all that type of stuff, go watch that video first. It'll help clear the air. It's not a necessary first view, but it will help a decent bit.

This video is different, though. This is the video where I show you all the things I've been doing with Fable, all the awesome stuff I've been able to ship, but most importantly, the ways that I've been doing it, the prompts I've been sending, the systems I have built, the changes I've made to my Claude MDs, to my skills, and all these other things, and how I've most importantly changed the way my brain works when working with models in order to maximize what I can get out of this new era. And this really is a new era.

Fable isn't just an improvement. It is a fundamental change in what these models are capable of enough that it makes it sound like I'm reading a script when I'm not. This is all off top. Like this model's screwing with me if you can't tell. I've effectively been oneshot. This is my psychosis moment and I can't wait to share it all with you. But someone's going to have to pay for my therapy and it'll probably come from the budget thanks to today's sponsor. My agents get a lot right when I'm building. It's honestly really impressive. But there's a few things they always get wrong no matter how good the tooling gets. The two that hurt me the most are off and billing. They're just like the kryptonite of AI agents.

They could never get it right, especially the billing side and trying to get that linked to the user in a way that makes sense, the amount of weird random transient billing failures we get across services is so annoying. Well, it was until I started using today's sponsor more, Clerk, because not only do they have the best off platform that is the easiest to integrate across everything, be it web, mobile, or anything else. We're actually working with them right now in order to get the T3 code electron desktop app off for our new remote stuff that I'm really excited about. What's way more exciting to me is the subscription platform that's now built in. Having tried pretty much everything like this, I can confidently say Clerk is the easiest and has the best abstraction. Instead of all the obnoxious effort trying to set up a staging and production environment for Stripe with their own separate keys, their own separate product IDs, their own separate everything that you now have to mirror in your real database just to make sure it works. You just get to configure it in the clerk dashboard.

Once a user pays, that payment information is attached to the user itself where it belongs. I don't care what subscriptions exist. I care if this user is subscribed. And Clerk gets that right. They have custom components and helpers where you can check what state a user's in and what plan they have to make it trivial to render the right UI or go down the right path. And it ends up costing the exact same as going to Stripe directly, which is unbelievable.

Any one of these pieces would be worth moving to clerk for, especially having the user profile component with all the billing info built in. The amount of time Mark loses dealing with billing disputes and email is obnoxious. And this would have saved us so much of that trouble. Fun fact, if you Google search Stripe recommendations, my repo is the first one to come up. This is a multi-page guide on how to try and get Stripe to work properly in your apps.

But at this point, when people ask me how to get it right, I usually just point them at Clerk. Set up Both and payments right at soy. Clerk. The amount of work I've done with this model in the last few days is still just kind of screwing with me, especially because I was at a conference for half of it. Shout out to the T3 Code mobile app, by the way. I was doing a ton of work on my phone while also at this event and it shows I got a ton done like an unbelievable amount done. This is the closed PRs in Lakebed where reminder I'm still the only contributor. I have not forced this project onto my team yet.

They don't deserve the slop. It's cleaning up fast. Thank you Fable. But all of these PRs were done in literally two to three days. the sheer volume of work here, especially if you're looking at the ones that actually merged. Like this pile here, it's like I think 11 or 12 PRs that merged all not just in one day, but from one thread. And I keep going and I'm on page two. We're still in things from 2 days ago.

Then there's a 4 day ago one, 5 days ago, then a huge jump to 2 weeks because I was just not really feeling it. When I lost access to this model, I tried to run around in circles and get the best stuff I could out of Opus, but it wasn't at the bar I needed to ship. And what ended up happening is in this particular project, Lakebed, I had a bunch of these PR stack up of work that was like 50 to 80% complete, but I didn't like the SDK changes it made. I didn't really have confidence in the implementation, and cleaning it up would just take too long.

And the result was that I kept having ideas. I kept making PRs and they kept sitting there doing nothing. I probably had 20 or 30 PRs just sitting in this repo, not moving, and the project has been sitting there as a result, and I don't like that. I really want to ship Lakebed. I've talked about what Lake Bed is in some other videos. I don't want to bog you down with the details until I actually have it launched. I haven't earned the right to market this yet. I want it done. So, we'll do that later, but I want to talk about the workflows that got me this far in the project, as well as some of the cool things I'm doing on other projects that I've been working on. I'm going to start in a weird place here, but I promise it will help. I'm going to start with minmaxing tips and like cost reduction things in order to make sure that when you adopt this workflow, you don't immediately bankrupt yourself because if you were to go to the second half of this video and skip straight to that part, it will be expensive. That's what I did in the first 3 days we had Fable and it was expensive. I didn't pay cash. I bounced between two subs, but I looked at how much it would have cost and it was in the thousands of dollars. And all the work I just shared here, all of the PRs that were closed, fixed, updated, recreated, merged, I want you to just think in your head and guess how much you would expect that to cost with Fable running in a loop for 5 and 1/2 hours getting all that work done. You're probably thinking in the thousands, cuz that's what I would have expected. But it was only around 150 bucks. Not just Fable, but all of the other models I had Fable calling throughout. Wait, but Fable's so smart. Why are you using other models, Theo? And here's where the tips start. I mentioned in my previous video that I was teaching Fable how to use codecs and I'm going to show you how.

But first, I just want to make sure I emphasize this point. I talked about it before. Do not use Fable on higher than high reasoning efforts. The X high and max options are dangerous and ultra code causes it to use workflows when it shouldn't. Those three options, pretty much everything to the right of high.

These are dangerous. They look so enticing though. the fancy gradient when you hop over it like it's a slot machine begging you to put another coin in. I'm going to beg you to not fall for it. High is way smarter. I know that sounds like I just made a mistake. I didn't. X high and max end up secondguessing themselves too much because they run in loops. They'll just go longer and longer with their reasoning and the result is often worse code that is way overdone that has way too many changes for the simple thing you're asking for at a cost that is absurdly higher than it should have been. Low, medium, and high might not be as efficient as like GBD 5.5, but they are capable of things that 55 is just not even close to capable of. And they can still run for really long times. The reasoning effort does not determine how long it can work for, which a lot of people seem to think that max can solve harder problems and work longer than high can. But the reasoning effort only applies per tool call and per change. So if you have work that takes 500 steps, Max isn't going to do more steps necessarily. it's going to think more per step and most of the steps don't require that much thought.

The problem is that X high and max since they are thinking too much per step they're going to overthink the thing and they do and they do aggressively as does set 5 as arguably Opus 48 does as well although not quite as bad. I am admittedly a little scared that this five family of models from Anthropic might have a big problem with overreasoning with these two options. So honestly just keep it on low through high. I personally leave it on high and don't think about it. I'm only thinking about it now because I saw people complaining about their bills and I talked to them and every single person complaining about blasting through their usage too fast was on X high max or Ultra Code. By the way, you might not have known this. Ultra Code uses high under the hood. It just spins up a ton more of them. So, even Anthropic agrees high is the best bang for your buck.

Trust me, just just go with that. It's a default for a reason. It's their default in other things for a reason. It's the right level. Now that that's out of the way and I've probably cut your bill by half or more just by making that one change, let's talk about the rest. Specifically, all the fun things with how I taught Claude Code to use codecs. I talked about this a good bit in the other video, so I'll do a brief recap before showing the implementation. The TLDDR here is that the usage you get for the codec sub is insanely generous. So things that it makes sense to use for you probably should. One of those things is computer use, which OpenAI just kind of slaughters Anthropic at right now.

The Codeex desktop app has a ton of cool hacks that let it use your Mac in full. Not just like opening up a tab in Chrome and poorly navigating it. More like setting up Xcode for you or navigating complex applications and pulling data between different things and sending you a video when it's done. The computer use in codeex is insane. So teaching cloud code how to call that is awesome. But the biggest thing is the tasks that require a lot of token usage. things like digging through logs, things like reading giant PDFs and implementation specs, things like doing computer use and looking at hundreds of screenshots over an hour as it tries to navigate your machine. Teaching Fable how to designate those tasks and route to the right models didn't take too much time.

I spent maybe an hour on this and it has massively cut my costs. All those PRs I was showing, as I said, my use across all the models is like $150ish dollars, but that all fit easily in my two subs, my Cloud Code sub and my codec sub. Neither broke 40% usage. In fact, my codec sub was at like 15% or so for the whole week. And my cloud code sub admittedly reset on like I think it was Friday evening and we got the model back on was it Thursday or Wednesday?

Yeah, I think it was Wednesday. We got it back and my reset was Thursday night and I got to 40% on my weekly while pushing it my hardest. Their usage is totally reasonable as long as you take advantage of these tips. Most of my work was in my global claude.md. I spent a good bit of time in here. I already made some changes here about specific behaviors I want like which tech should it use when initializing a new app.

Honestly, I'm going to change a lot of this to like bed soon. the style of code I like, general preferences, just like the normal stuff you probably have or at least played with putting in your claw MD. I could do a very long video about how to influence your agents to have the same psychosis as you. It's a thing I think a lot about. I haven't went quite as hard in my global claimd on that yet, but I did go a lot harder in this file last week because again, I wanted to try and teach it how to use these types of things. It all starts here. If computer use is helpful for completing or verifying work, shell out to GPT55 with codecs for it. Specifically, the words shell out here are important because what I'm telling the model is that it can call GBD55 via the shell via bash because it has the ability to use bash.

It's using cloud code. Of course, it does. And then I give more hints. I have another set of skills that we'll get to in a bit, but I think reading through here is the best starting point. I have a section in my cloudmd for picking the right models for workflows and sub aents. If you're not familiar with the difference here, sub aents are the idea that the agent can call a tool to spawn another agent to go do a thing, which is useful if you have like five files that you want to have analyzed and you want five sub agents to analyze each one.

Workflows are different. Workflows are programmatically defining all of the cool things that you need your sub agents to do. So if maybe you have different stages where the stage one is to go through each file and then stage two is if flagged do another thing with two additional reviewers workflows allow that where they allow you to programmatically take the results of a stage and use that to dynamically cue things in a different stage. It's all just a big JavaScript file the model will write and then use that to trigger and break down all of the work for these types of big longunning tasks. For example, it was super helpful when triaging all of the poll requests that I had left open and stale on Lakebed. It made a workflow to go through all of them and categorize them. And once they were categorized, they could be thrown to other stages with other types of sub aents to do different types of things.

People have tried to build custom flows for this in their own tools like defining this is what a review sub agent is, this is what an adversarial review sub agent is, this is what an exploratory sub agent is. That was all stupid and never made sense. Now it makes even less sense because Fable can invent those different archetypes depending on the needs for your specific task. Every time I'm trying to get an agent to review things, the needs are slightly different and the model is now smart enough and understands sub aents well enough to define that itself. And

this is where the power of the model starts to really shine. But at the same time, its awareness of the specific benefits and negatives of given models is not quite there. When they trained this model, 55 didn't exist and all these cool computer use capabilities weren't there. Opus 48 also didn't exist, so it's not sure what it is and isn't capable of. So, I had a good gut feel for where to put things. I also want to call something out because you might have read it on the screen. I say OpenAI is near free for me due to a deal. I promise you I have no special deal with OpenAI at this point in time.

This is just how Claude interpreted what I told it, which was that there was a current deal on the amount of usage you got in Codeex where it was double at the time. It's no longer double, but it's still insanely generous. As I said before, I've been using this heavily and I've only used 15% of my weekly limit. That's nothing special. I'm using this model for hours a day every day. It's just genuinely hard to exhaust your limits on codecs. I think it's like 14 grand or so a month of inference you get right now. It's insane. And that's also outside of the resets of which I've stacked a ton. So, while Fable interprets that as a deal, that's the same deal you get too. Don't read into that. As such, I ranked the cost in 55 to be relatively good. The scores here are the score 1 to 10 on how beneficial is it for this thing. 55's cost to me is relatively low because my usage there is basically infinite it feels, especially when you combine that with the resets available. So, I gave this a cost in quotes of nine. This is me telling Fable, "Yo, by the way, you can basically use 55 for anything. I don't care. This model's just really efficient." And it does that. I also rank intelligence and taste. These are the categories I chose because I care about these things and I have a lot of problems with GPT models writing code that isn't necessarily the code that I would have wanted in my codebase. They could solve any problem at any level of complexity and they can match patterns they are shown very well. But 55 writes TypeScript like a Python dev and it writes Rust like a super paranoid C++ dev. It's just not the code I necessarily want, especially for like public facing SDKs and APIs. So even when I was using 55 more heavily, I used to have it call opus in order to get feedback on APIs and SDKs in order to clean up the code. Now I've just inverted and I let Fable steer everything and life has been much better since. So I call out that GBD55's cost and intelligence are both very high, but its taste not necessarily quite as much.

So if you're customizing your cloud MD and agents MD, the biggest thing you can do in there is effectively a glossery. The terms you like to use to describe things that might be understood by the model might not be. Just write them down and what you mean by it. So for me, intelligence is how hard of a problem the model can handle unsupervised. And taste is things like UIUX, code quality, API design, and copy. This both helps me explain to the model what I have in mind when I'm describing tasks and work and the issues I have with things, but it also helps the model understand what I mean further in the document with intelligence and taste and how to apply this information when it decides what models to call for different things. So, as I said, 55 intelligent, no taste. And with Sonet 5, not much cheaper, much less intelligent, slightly more taste.

Opus 48 slightly more expensive because again Sonet 5 is so token hungry that Opus 48 is often cheaper meaningfully more intelligent but still not quite as smart as 55 way higher taste and then Fable where the cost sucks intelligence is best in class and taste is also bestin-class this is genuinely how I feel about these models by the way so I just told the model how I feel and I told it how to apply these things these are defaults not limits you have standing permission to override them if a cheaper model's outputs don't meet the bar rerun or redo the work with a smarter model without asking. Judge the output, not the price tag. Escalating costs less than

shipping mediocre work.

You couldn't tell. I did not write anything from here down. The next point is around cost. It says cost is a tiebreaker only when axes conflict for anything that ships. Intelligence is greater than taste is greater than cost. Honestly, again, the model wrote this. I don't like it. Don't let cost prevent you from using the right model for the job. Instead, take advantage of cheaper options to get more information and try things before moving the work to a more expensive option there. That's much better aligned with how I feel. Real changes happening on the fly, guys.

Isn't that cool? Thanks for giving me incentive to actually read the slop. Next, I talk about bulk mechanical work like clear spec implicitations, data analysis, migrations. 55 is effectively free. Anything userfacing like UI copy, API design needs taste greater than seven. Reviews of plans and implementations, Fable 5 or Opus 48. Optionally 55 is an extra independent perspective. A big call out here. Never use haiku. Just don't at this point. It's not useful for anything real, especially with 55 being effectively free. I then call out the mechanic of 55 only being reachable through the codec cli. I say use the codex implementation codex review and codex computer use skills for work that they don't cover.

Investigation data analysis. Run codeex exec- readon directly with a self-contained prompt. Instructions again on how to use codecs. I get some insertions here on how to use 55 inside of workflows with sub aents because you can't just call 55 as an option when defining a workflow. You have to call the claude model when using cloud codes workflows. So I gave it some instructions here that it can use sonnet on low in order to spawn 55 get its results and then report those back. It ends up being really cheap. It lets that spin up and use 55i for real work and then come back with results. I was getting annoyed that I couldn't see which sub aents in workflows were using 55. So I requested that it put a prefix in front so I can more easily see it and that ended up helping a ton. The tasks can time out so I called this out. A lot of these are things that got appended as I ran into problems and I hope you take that lesson here. Most of this wasn't me having this epiphany on how to get it all right. I spent about half an hour getting it mostly working and then as I ran into problems I would go back to the original thread and tell Claude, "Hey, I had this problem. How can we prevent this going forward? And it would suggest changes. I would tell it to cut them in half and then put them in here. Went pretty well. And with that, let's look at my skills, which you might have noticed here. There are not a whole lot of skills. I'm normally not a big skills user. In fact, I plan to delete these once better models come out, or better yet, Anthropic gets good at computer use, but for now, having these here has been very helpful. I have Codex review with a description that says you can ask the Codex CLI 55 for an independent code review of uncommitted changes, a branch diff, a commit or a specific implementation yada yada. This is how you use it for review work. I then call out here that Codex is an independent reviewer when the user wants a second pass review or when the change is broad enough that another agent perspective is useful. yada yada yada. I define a workflow. You identify the review target. You create a temporary artifact directory for the Codex report. You run Codex review with a focused review prompt. And then you read Codex's report and verify the important claims against the code before presenting them. Again, this is meant to be called by a sub agent that is triggering codeex and it gets the feedback and then passes it back up to the parent model. The most important piece here is the commands.

Not because the model can't get them right, but because the one or two times they get it wrong, it's really annoying. So you ask the model when that happens, what did you get wrong? What is the right structure for this? And then once you have that, you can go add it here. And for me, it has worked pretty damn well. There are then some instructions on how to prompt Codeex because I've noticed that Claude likes to prompt Codeex as though it is Claude, which it is not. The prompt should be much simpler. And this is an example of a very simple prompt that makes sense here. I would even argue things like do not edit files does not really necessarily need to be here because again, it's a codeex model. It's not going to do things you don't tell it to, unlike Claude models, which love to do things you don't tell them to. Another problem I had is that sometimes Codex wouldn't find anything and that would confuse the parent model and it would rerun. So I added a bunch of these call outs like if Codex finds nothing, say that clearly and mention what the review target is that it inspected. That ended up reducing the issues I had a ton.

Again, you need to set this up for yourself. Don't just blindly copy paste my stuff. There's a reason I'm not posting the links anywhere. I want you guys to learn from this. Maybe screenshot it and pass it to the model and say, "Hey, I want to set up something similar. Can you guide me through it?" Do it. Learn it. change it. Experiment. If you're scared of going in and editing these files, you need to get over that. And if I let you guys just copy paste my stuff, you'll be afraid of breaking things when you make small changes. I need to know you won't be scared of that because this is where the fun comes in. I have a codeex implementation sub agent that's very similar for doing bounded work, usually on a work tree, in order to set up the model to go make changes and bring back useful results. It's a little more guarded than I would like, but it works pretty well overall. I haven't had enough issues to care. If I do, I will change it and I will let you guys know in future videos. But my favorite by far is the computer use one because this is what allows Claude to have Codeex's computer use powers without having to build it into Claude or more importantly waste a ton of your money in usage.

Remember, the model only sees the description of the skill until it uses it. And once it uses it, it pulls all of the rest of the text in. So the description should have all of the info that Claude needs to know whether or not it should call this and get the rest of the context. This one's a little wordier than I would like, but it's been good about calling it when it should and not calling it when it shouldn't. It says, "Ask codec CLI with GBD55 to run local app verification that needs computer use, browser automation, simulators, screenshots, app launching, or independent runtime inspection." This is how 55 is invoked for computer use work.

Use when the user asks Claude to have Codeex or 55 test a flow, verify user behavior, inspect a running app, capture screenshots, or report confirmation and feedback about implemented behavior. I might even change this now from when the user asks Claude to have codeex or GVD55 test a flow to just when the user asks Claude to test a flow to default here instead. In fact, we're going to do that. Use when the user asks Claude to test a flow, verify UI behavior, inspect running app, capture screenshots, or report confirmation and feedback of unimpeded behavior that benefits from computer use functionality. There we go.

Again, adapt these as you go. Think about your experience and don't be afraid to make changes. It's so cheap to do and it can meaningfully improve your workflow. And once you get in the habit of changing, editing, and getting these things how you want, you'll end up much happier overall. And now is when we start diving into

how I actually got all of this work done. Claude code is admittedly broken in a lot of ways, so it won't be the easiest thing to showcase, but I will do my best and I will break down how I was thinking about things and how I took advantage of them as well. I started with a really simple prompt. investigate and review the open PRs for Lakebed right now. This was a simple request. I wanted to ask the model to deal with all of the things that I had left open. Remember I said I had a bunch of open? We now have the number. It was 16 open PRs of somewhat finished, somewhat trumped, somewhat dead work. And the variety of the work there was crazy. I call this out accordingly in the prompt. I think it's actually a good example of a prompt for this era of working. Investigate and review the open PRs for Lakebed right now. I wanted a list that indicates the following. Which are ready to merge.

Which are mostly good but need a rebase or other small touch-ups. Which have been trumped either with better PRs or things that have already merged. And which have good ideas but are better off scrapped and rewritten. Use a workflow to break up the work in multiple reviewers to help solidify your decisions. And then it ran for a while. Not too long, but a meaningful amount. It spawned a workflow. You can see here the code that it wrote for said workflow. It ran for a while and then it came back with a list. All 48 agents finished. 16 investigators, one per PR.

Each verdict then stress tested by Fable plus Opus judge panel. 14 of 16 calls were unanimous. I resolved the two contested ones below. Here's the triage. First, we have one PR that was ready to merge, which was the separation of docs cuz I had both 55 and Opus 48 were kind of blurring the line between docs for my users and docs for the maintainers of Lakebed. So, I did a hard cut and that PR was pretty good and easy to merge.

So, that got merged fast. Then there was a couple others that were mostly good but needed some touch-ups. Then there was a pile of PRs that had been trumped. And then the good ideas that should be scrapped and rewritten as well as pairing them together for the things that were similar. I gave a suggested order of operations. Start by merging 69. Then fix the allow list for 68 and merge rebate 61 then 48 yada yada calls the ones to close out and all of this. I liked how it was thinking. I liked the feedback it gave. I actually did read all of this cuz I wanted to understand what it wanted to do. And once I was content with what it said, I realized, you know, I have a staging environment set up here. It won't affect prod if it gets things wrong. Let's yolo a little bit. This was a goal I set. First, I said to close all PRs that have been trumped or otherwise not worth keeping around. Then, help me spec and prioritize the work for the other PRs you think you should close. Write up new HTML plans for a best path forward implementation for those features and fixes, honoring the goal of simplicity that we strive for in Lakebed. Break up the planning work into sub agents.

Review the plans with sub aents as well. A workflow may be helpful. Once you have plans you're happy with for each of those pieces of work. Share the links with me so I can review them. Make sure the plans you write have links to the PRs that they are inspired by. If multiple PRs implement the same feature, link to all of them. Make sure to describe where the existing implementation is and how you feel we should handle the next one. I was wrong.

This one wasn't the goal. This is the go clean up, destroy all the dead PRs, and then tell me in detail what you

want to do to the remaining stuff. And after a couple more minutes, it had done just that. It gave me links using my HTML planning skill for all the different things it wants to add and how it thinks they should be implemented. That was actually a really pleasant read, which by the way, I did on my phone. Super convenient. So, I read through most of these. I looked at the PRs that had inspired the changes and made an executive decision. All of them looked pretty damn good. Yeah, five of these remaining pieces looked really damn good. I did have one PR that I wanted to possibly merge at this point. So, I asked it about that one. I took a look.

It said it was good, so I merged it. Wait, no, I had to rebase it. Yeah, I asked it a question. Which, by the way, ask your models questions when you're not sure about things. It's a great opportunity to align yourself and the model and what the plan is for something. So, I asked it, what are the streams of work we need to complete to address all the PRs that we've discussed? Is it as simple as rebate 61, merge 48, 68, reimplement the rest? If I wanted you to implement all of the plans you made, would it be best to do that as a single workflow or to make multiple work trees and threads by hand myself? I expected it here to push back and say, "Yeah, you obviously need to break this up into different threads." It decided to break this down into multiple streams. The first one being to land the four open PRs that were ready to go.

Then to implement the five plans that were left over, so eightish PRs total. Also note that each plan deliberately decomposes into bounded PRs. The third plan apparently was going to be three separate PRs. Dual write, read, path, flip, then the column drop, and then yada yada. You get the idea. I don't want to bore you with the details of Lakebed. And then we talked about the workflow to get this all done. Single workflow versus handmade work trees.

Neither, honestly. And you don't need to make a work tree by hand either. A single workflow is the wrong tool for the umbrella. Workflows are deterministic scripts that shine for fan out and verify. But this program is checkpoint driven. Each PR needs CI, your review, and a merge before the next rebase. In midstream, you'll make product calls like the open questions that are below that a script can't anticipate. One giant workflow would either barrel past those checkpoints or stall at the first one. So, it recommended that it orchestrate the streams from the session that we're already in, spawn work trees itself to actually do the implementations, and then use workflows only where they're strong, which is for multi- aent review passes one before it merges the PRs. It also identified that certain places had parallelism and possible like a plan two and three could be run in parallel so they don't touch each other's stuff as much and it wouldn't have too many merge conflicts. I then asked it to take all of this work and write it as a to-do list in the directory so it would be easier to just crank through all of it.

And then we get to the goal or more correctly put the insane world we now live in. I started a goal which basically tells the model keep going until the conditions pass. In this case the conditions were to complete all of the work that we have discussed here. You have permission to create work trees, rebase, branch, merge PRs, close PRs, etc. This is me explicitly telling the model, you have my thumbs up, merge away. This might sound insane. Kind of is, but I'll show you why it's not as bad as you think in a little bit. But if we're all going to go insane here, and we all very well might, I'm going to do a second ad break quick. Building apps with React is really easy. That's why agents are so good at it. It's never been easier to build for the web, but it's still not that

easy to build for mobile. Believe me, I've tried everything from Cotlin and Swift to React Native. It seems like AI just doesn't get it anywhere near as well.

Getting those things right is super difficult. And having a good codebase with a good team is essential. That's why knowing about Infinite Red is such a powerful hack right now. They can come in and modernize your codebase if you already have a mobile app. They can help start from scratch. They can onboard your team to do these things correctly. But in the end, what they're here to do is set you up to get your mobile app right. They obviously lean heavily into Expo and React Native, but if you want to build on Swift and Cotlin, they can help with that, too. They can even build the native bindings you need to get those parts right. And in a world where GBT55 can invent a cloud from scratch in a few minutes, but can't lay out a UI correctly on Xcode, yes, I'm speaking from experience, it's actually insane how bad 55 is at mobile without having the right guidance at least, which is what Infinite Red is here to provide.

Well, get your team set up with a good working codebase and more importantly, a good working understanding of how to work in that codebase. And then you can do the same with your agents. That's why everyone from Zoom and Microsoft to Domino's and Starbucks has worked with Infinite Red to polish up their mobile apps and get their teams ready to build better on React Native. So if you're curious about React Native or you're trying to fix an old broken app or you want to get things right from the start, hit them up at soyv.link/infinite red.

Okay, as I was saying, we create the goal to complete rebase, branch, merge, work tree, all of the things it has to do to deal with this insane amount of work I put in front of it. This is like a month of work in a to-do MD. Now, by the way, I tell it to go through all of the work step by step and mark each to-do in to-do.md done when completed. The to-do MD changes should be committed and merged as you go. Use the tools available to you to break this work up logically. Review it thoroughly and merge with confidence. Do not merge code until my automated code reviewers have approved it. In this case, Bugbot, Macroscope, and Code Rabbit because it's what I have set up in this or goal set.

Goal acknowledged. And then it ran for 5 hours. And every time I checked GitHub on my phone, more code had landed. And that's how we got here with this giant pile of work that actually landed. My whole month road map that I'd been running around in circles, not shipping, just thinking too much about now is in Lakebed. It did the thing. This might sound insane, like merging straight to Maine when the agent thinks it's good enough. But when you see all of the comments and all of the push back it got to clean up the implementations and keep them simple before merging and you combine that with the most important fact which is that production deployments are still a human in the loop. This is just the staging deployments that happen when you merge domain. So the model cannot ship or touch prod. The model can use staging for basically whatever it wants. I haven't done a prod deploy in a bit because also there were some things broken in my staging environment because of other models merging things they shouldn't have. And I had Fable clean all of that up before even doing this work. So I started by having Fable get staging to a place I liked enough that I would be happy merging main and shipping it to prod. But instead of just rushing there, I decided to clean up everything else. And it did it. It just went and went and went until everything was done.

And then I had it on main. I had it in staging. And I went and personally stress tested it and also spun up other agents to go try out all the new features and also try out all the old features. Maybe try some apps that were built the old way and see how they port over to the new way. And then I had other agents look over all the changes between prod and main because a lot had changed on this branch since and see what things needed to be derisk and then spin up more tests to go check those things. And there was basically nothing that needed to be fixed. I'm still kind of blown away. I burned way more tokens trying to verify the work that Fable did here than I burned getting Fable to do the work here. And it was all good.

There was nothing to change, which means again, I'm not pushing it hard enough. And I'm still looking for more ways to do that. Believe me, there will be plenty as I go. But god damn, Lake Bed is now a month ahead of schedule when it had fallen admittedly a month behind. I was really hyped. And this workflow was just so cool to watch as it went. Like truly unbelievable. I wanted to keep working, but this one was running on my main laptop cuz I didn't think I would get this far. If I did, I would have done it on a different computer. But I was just playing around with the new quad code and the new model and got way further than expected. But I wanted to do other work on other projects. So that's when I started connecting to my Mac Mini, to my other Linux boxes, and to the other machines I have. Check out my Linux video if you haven't. It covers a lot of this stuff. And I started setting up Claude on those. And as usual, I got really frustrated trying to use cloud over SSH because all of these nice things I rely on like image pasting, decent select behaviors, scroll that works just weren't really viable.

And I did what I always do. I went back to T3 Code. And god damn has T3 Code helped me maximize my utilization of this model a ton. The cool thing with T3 Code is that you don't have to use it on the same machine that's running your agents. It's very easy to set up something like tail scale and connect to your other computer on the T3 code website or on the T3 code app on your computer. Or if you're a little bold and you're down to try and build it yourself, the T3 Code mobile app, which is also open source, which by the way, all this is open source. We have no way to charge for it yet. We don't make money. I've probably spent over 250 grand both in tokens and salaries for the people working on T3 Code. This is a gift. Enjoy it before I have to start charging. That all said, the T3 Code mobile app is in the repo and you can build it yourself for your phone and it's really goddamn cool. I had no idea how much I was underrating it because I tried the first version when Julius was like 3 days in. He's been grinding on it for a month and it has gotten absurdly good. So, I set that up on my phone, connected to Tail Scale, connected to my computers, instantly blown away. I was so blown away that we started working on T3 Connect, which is a service we'll hopefully ship in the near future that lets you not need to set up tail scale to get all these benefits. You're a nerd though. You can set up tail scale. Just do that. It's totally fine. Don't pay us money or wait for us when you could just have it yourself. The reason I bring all of this up is because it made parallelizing my work trivial. Every time I had an idea for a thing or even when I was just on my phone and running into problems, I would just spin up a workflow for it or spin up a work tree or just get something done. When I was working from my phone, I kept noticing small bugs. So, I kept spinning up new work trees to fix the bugs. Like I wanted to have the repo for T3 code with a favicon at the root level because then it would have the little icon in the corner here like other projects do as you see with T3 chat lake bed and other things that we have a easily findable favicon in T3 codes was a little too hard to find. So I bumped it up higher so T3 code would see it. I then asked it to commit and make a PR following the repo guidelines and it did. And it looks like it had a transient failure because testing is hard and the cursor adapter is not as reliable as

we want it to be.

Cool. You know what this is? This is an opportunity to show you guys how I work. Here's what I do. First, I take a screenshot of the problem. Then I go to something like T3 Code. I open this and I can pick which machine and which T3 Code directory to use because this is a repo I have in a lot of different places. I have it on four different machines and one of them has it in three different places. I'm just going to do this in the standard T3 code repo on this project. I'm going to pick whichever of my favorite models. In this case, I'm just going to ask Fable to do it. I'm going to paste the image. So, I'm going to say figure out why this test fails randomly. It just failed for no reason on PR number 3683.

And then a link to the bad run. It is on a work tree, so this won't affect other work. It has now been spawned. And now I'm not as worried clicking the rerun job button. And hopefully in just a moment, this PR will pass. And as you've probably noticed, the slowest part of this workflow by far is waiting for GitHub load times. I cannot believe how bad of a state GitHub is in. I would gladly pay a hundred bucks a month for something just like GitHub with all the core features we need. That is way faster. Ideally, I would just pay to GitHub and they would make their suck less, but that's never going to happen at this point. You get the idea, though. And now I have this work going.

And while that's going, I can go see the status of other work I'm doing, like my attempts to improve the connection experience when you're using SSH into a box and setting up T3 code on it so another machine can control it. I want that to be buttery smooth and I'm really hoping we can get it there soon. For now, just ask your agents. It'll figure it out, but I want it to be one command. I then was working on a new machine I just set up and I wanted to see if it could get T3 Code and the mobile app working properly so I could not have to rely on this laptop as much for all of my iOS work and it got pretty far. Had some computer use issues. I'm going to fix that after I finish recording. This one was actually really fun though. This is an experiment I was trying. I had all of these work trees I spawned on my phone and most of them were further along than expected and relatively simple changes. And rather than make Julius deal with like five different PRs, I decided instead to just ask Babel to combine them because some of them had overlap, some of them would have conflicted. I didn't want to deal with it. So I made an executive decision. I asked, "Can you access all these work trees on my machine? I'd love to pull the changes from all of them into one branch with conflicts handled so I can test them all at once." Five minutes later, it did. And now I have this branch I can pull down on my laptop in order to test all the changes. It's great. It handled the conflicts that existed because of multiple things that touched on each other's stuff. And now I have all of this in one place where I can file a single PR and easily document everything that changed. I could even pull this onto a different computer like a Mac and have it go verify all the changes, too. I do want to be really clear about something. I don't expect the majority of the code I spawn this way to be ever used or merged. Part of this is just seeing how complex the problem is. Like if I ask the model to go solve this problem and it takes under 3 minutes, it was probably a simple fix and I won't feel bad filing that PR and telling Julius to get it merged. If it takes 15 minutes, that's a little scary.

I might want to pay more attention. If it takes an hour or more, oh, something's wrong with our architecture. We need to go deeper and figure out what's up. The amount of time it takes and the amount of changes it has to

make in order to make any of these things happen is a really good indicator for where the good and bad parts of your codebase are. And for a lot of these, the changes were so simple, it's like an easy merge. Like obviously you want to go put that up and do it. But for some of them, it took long enough that I'm a little bit concerned. For example, with the mobile thread scroll jumping stuff, this one it took a while on. It took over an hour and a half. That scares me.

I'm not going to blindly merge this code. I'm going to go put a lot more time in here. Especially compared to the issue that annoyed me more, which was sliding back not behaving properly, which it fixed in 2 minutes and 20 seconds. It was so simple it concerned me cuz it talked about a drawer behavior. We didn't have a drawer. I think we might have in the past though. So, I just asked, "Do we even have a drawer anymore?" Apparently, we do, but the edgewipe gesture is the only entry point. No, we don't then. But now, I know like it was kind of wrong about that. So, I have to put a little more time in. You get the idea. You got to think with your brain a bit. Not just like reading the code thinking, but thinking about the architecture, thinking about the request, thinking about how fast or slow it was, how many things it touched and then make a good decision accordingly. And if you don't know, just ask. You'll be amazed how useful the models are when you ask them questions about things. It's almost like that's what they originally built for or something crazy. Hopefully, this has been a good overview of how I've been working with Fable. As you can see, the sheer volume of work I'm doing in all of my projects has massively ramped up. I am more ambitious than ever. I'm having more fun than ever, and I'm pushing myself harder than ever. I do have one last pro tip for you, though. Vibe Proxy. If you're scared of hitting your limits, this might be worth setting up.

It will autosplit your traffic across multiple different accounts. I had heard about this before and I thought it would be a little sketchier or harder to set up. It really wasn't too bad. It is using the API key version of Claude Code, so you do lose like the built-in / remote control and a few other features like that. I haven't really missed it that much. I've actually found this pretty pleasant. As you can see, I'm not getting close to my limits yet, but I do also have another 4 days before they take Fable from us in the subs. So, I'm probably going to get a lot closer, especially once I finish recording. I thought this video would be short. That was foolish of me. I'm going to go back to coding. I hope this inspires you to do the same. Let me know what you think of my workflow. Am I insane for even sharing this, much less doing it? Or is this actually inspiring and helpful for you as you try to go build using these models yourself? I have had so much fun.

I hope you do, too. So until next time, peace nerds.