SuperDataScience

SDS PODCAST

EPISODE 1020: HOW TO CHOOSE MODEL SIZE AND EFFORT LEVEL: THE TWO CRITICAL DIALS

Jon Krohn:	00:00	This is episode number 1020 on choosing the right model size and effort level. Welcome back to the SuperDataScience Podcast. I'm your host, Jon Krohn. Today's topic is the two dials that increasingly determine what you get out of a large language model. That's which model size you select and how much effort you tell that model to spend. If you've opened up Claude, ChatGPT or Gemini lately, you will have noticed that the model picker has been sprouting new options, not new models exactly, but new settings with names like effort level, reasoning, effort, and thinking level. In today's episode, I'll unpack what these settings do under the hood, when you should reach for them, and when you should leave them alone. My jumping off point is a blog post that Anthropic published in July written by Lydia Holly, a member of technical staff on the Claudecode team.
	00:56	Claudecode, if you haven't used it, is Anthropic's agentic coding tool. You delegate coding tasks to Claude from the command line or your IDE, and it goes off, reads your files, writes code, runs tests, and reports back. As usual, we've got a link to the full post from Lydia Hawley in the show notes. Although the post is framed around coding, the mental model it lays out applies to any AI platform you might use. So even if you never touch a terminal, stick with me here. Also, if you're listening to this episode around the time it's released, there's a good chance you'll hear a mid-roll ad from Anthropic in this episode. I'm delighted to have Anthropic as such a big supporter of the podcast this year, but they have no influence on the topics I cover on the show. And while this episode is largely inspired by an influential anthropic blog post, the guidance generalizes to any model family, not just Claude.
	01:48	And so later in the episode, I'll explicitly generalize my advice with examples involving other frontier labs like OpenAI and Google. Anyway, back to Claudecode for the moment. Claudecode exposes two settings that both

appear to make the answer better, the model setting and the effort level. A reasonable assumption would be that the model size setting controls how smart the response is while the effort level controls how long the model thinks before answering. The first assumption that model size controls how smart a model is holds up. The second one around effort level turns out to be incomplete in an interesting way. Let's start with model selection because Anthropic's explanation of what that dial does is one of the clearest walkthroughs of LLM inference I've seen aimed at practitioners. When you send a request, everything, your message, the system prompt, the tool definitions, any files in the context, the whole conversation history, all that stuff gets packed into a single API request.

02:49 The first thing that happens server side is tokenization. Your text is split into pieces and each piece is mapped to an integer from a fixed vocabulary the model was trained with. From that point on, your prompt is an array of integers. The model's job is to take that array of integers and predict which token comes next. It computes a probability for every token in its vocabulary and picks from the top. What turns your input tokens into those probabilities are the model weights, billions of tunable parameters organized into large matrices. Predicting one token means running your input through a long chain of matrix multiplications as we go through layers of a deep learning network to get into those kind of technical complexities a little bit. And then we read the probabilities out the other end. And the model doesn't generate a whole answer at once. It predicts one token, appends that one token to the sequence and runs the entire computation again for the next one.

03:46 So that means that a 200 token response is 200 separate passes through those deep learning matrices that make up the large language model. That loop is where most of your weigh time and most of your output costs come

from. Here's the key point. The model weights are set during training and by the time you are sending requests, they are read only. That's all inference means using the model after training is done with the weights frozen. Nothing in your prompt changes the weights. Your prompt and context can steer the prediction and steering works well, which is why putting your real code or your real documents in front of a model improves results so dramatically, but steering isn't teaching. If a library didn't exist when the model was trained, it isn't in the weights. Paste the docs into context and the model will use them for that request, but the underlying model retains nothing.

04:35 This framing also demystifies hallucination a bit because when a model confidently calls an API that doesn't exist, that's the weights producing a token sequence that looks plausible from training patterns. So the model setting does exactly one thing. It swaps which set of frozen weights handles your request. In Claude Code's case, at the time of recording at least, that means choosing between Claude Sonnet, Claude Opus, and the newest and largest model claude fable. Bigger models encode more knowledge and capability in their weights and each of their output tokens typically costs more to use. It certainly does in the anthropic case. What the model size setting doesn't decide is how many tokens get generated. The same prompt can produce wildly different token counts depending on how much work the model decides to do. And that's what the second dial, effort level controls. This is where the blog post corrects a widespread misconception.

05:29 Effort isn't a thinking time slider. In an agentic tool like ClaudeCode, the tokens a model generates fall into a few categories, reasoning tokens, tool calls, and the text it writes to you like plans, progress updates, and summaries. All of these are ordinary output tokens from the same generation loop build at the same rate. Effort

level shapes all of them. At high effort, the model reads more files, verifies more of its work, and pushes further through a multi-step task before checking back in with you. At low effort, it would rather ask you for more context than burn tokens figuring something out on its own. Mechanically, the effort level is sent to the model as one more input alongside your prompt, and the model was trained to behave differently at each level. That learned behavior is baked into the frozen weights. It sets a bar for how thorough and how certain the model needs to be before it considers a task done.

06:20 Anthropic illustrates this with a same prompt comparison where the high effort path generates roughly seven times more tokens to reach a higher confidence answer. Importantly, effort sets how far the model is willing to travel, not how far it must travel. If step one of a three hypothesis debugging plan finds the bug, a well-trained model at high effort will say so and skip the remaining checks rather than patting out your bill. Anthropic notes their team watches for overthinking during training because that also, in addition to costing you more, degrades effectiveness. So how should you use these two dials, model size and effort in practice? Anthropics guidance boils down to a single diagnostic question you should ask whenever a model gets something wrong. Did it try hard enough or did it not know enough? If the model skipped a file, didn't run the tests, or bailed on a refactor partway through, raise the effort.

07:18 If the model had all the pertinent context, visibly tried and was still confidently wrong, that's a capability failure. So move up to a larger model. So if you were trying it with Sonnet and Bump Up to Opus, if you're trying with Opus, bump up to Fable. And critically, before touching either dial, you don't need to touch model size or the effort level. If the problem is that you provided a vague prompt or a missing context, and actually that's a more common culprit for the model not doing what you wanted and no

knob can fix that. Tying everything together, the post offers an analogy I found sticky. Sonnet, you can think of as a strong generalist. Opus is an expert and Fable is a deep specialist who's seen problems almost no one else has. Effort decides how much time each of those different kinds of person of model spends on your task.

08:18 So Opus at low effort is like five minutes with an expert, deep pattern recognition, but a skim of your code. Sonnet at high effort is a good generalist with the whole afternoon. They'll read everything, run everything, and end up understanding your specific code thoroughly. With less of that, I've seen this exactly before recognition. And fable, even at low effort, is the specialist who glances at the problem and spots the thing nobody else would. Neither dial is universally better. Model size is roughly how capable, while effort is roughly how thorough, and most real tasks need some of both. There's a cost wrinkle worth internalizing here. On routine work, a small and a large model both get it right. So the large model's extra verification at a higher per token price is wasted money. Drop down. So drop down from Fable to Opus or from Opus to Sonnet.

09:14 On hard multi-step work, the equation actually flips though. The small model grinds through iterations near the stealing of its ability while the large model reaches the same quality bar in fewer steps. So despite the higher per token price, total cost per task can actually come down and be lower on a bigger model as long as your task is tricky enough. This means that cheaper per token isn't always cheaper per task. And in anthropics testing, for example, Fable finished long multi-step jobs that Opus and Sonnet couldn't reach at any effort level, which together with its price is the argument for saving Fable for work that really needs it. Now, as promised earlier in the episode, let's zoom out beyond Claude because the whole industry has converged on some version of these two dials. OpenAI arguably went furthest toward hiding them.

When GPT-5 launched in ChatGPT, OpenAI introduced an automatic router that decided on its own when a query warranted deeper reasoning, with paid users only able to force the issue via the model picker or by typing something like think hard about this into the prompt.

10:24 After user pushback, OpenAI surfaced explicit autofast and thinking modes and later added a reasoning effort selector with named levels ranging from a near instant setting up to an extended one alongside API parameters that let developers dial reasoning effort from none up through extra high. On the model size axis, OpenAI's current generation, GPT 5.6, comes in three tiers with celestial names. Soul, meaning sun, is the flagship for the hardest work. Terra Earth is a balanced everyday model at half Soul's price. And Luna, the moon, is the fastest and cheapest of the model family with the number denoting the generation and the name denoting a capability tier mirroring the Sonnet Opus Fable ladder over at Anthropic. And at the time of recording, an interesting little tidbit for you, OpenAI has confirmed a forthcoming model called Astra, another celestial word. This is a new class alongside those three existing Soul, Terra and Luna tiers.

11:29 And rather than being an upgrade to them, Astra is designed to coordinate multiple agents on long-running problems. An internal version reportedly cracked 10 open problems in mathematics and theoretical computer science, though there's no release date to the public, and even the shipping name may change when it is eventually released. Finally, Google took an evolutionary path with Gemini with respect to these dials. Their 2.5 generation models exposed a thinking budget, a raw token cap you set on the model's internal reasoning from zero up to tens of thousands of tokens. And with Gemini three, Google replaced that with a simpler thinking level parameter, more like what OpenAI and Anthropic are doing. And this just offers discrete settings like low and high, having

concluded that forcing developers to estimate token counts was the wrong abstraction, difficult to think through for us humans.

12:28 And then going out even a bit further beyond the proprietary closed models of the Frontier Labs, most reasoning focused open source models like the near frontier Quen models from Alibaba, which you can hear more about in last Friday's episode of this podcast in episode number 1018. Those reasoning focused open source models offer analogous reasoning effort toggles too. So whichever platform you build on, some flavor of the effort dial is waiting for you, and usually some flavor of the model size dial is as well. I find this convergence telling. Two years ago, the industry's answer to how do I get a better response was one dimensional. It was just use a bigger model. Today, the Frontier Labs are unanimous. The capability and diligence are separate axes, that they're priced differently, and that matching both to the task rather than maxing both out is what separates a savvy user, huh?

13:25 Maybe like you listener, from an expensive user. All right, before we wrap up here, let me leave you with three practical takeaways. First, start with the defaults. Every provider tunes the default effort to what most people would want to spend. And Anthropic, for example, explicitly recommends treating effort as a general preference for your kind of work rather than something to fiddle with task by task. Second, when output disappoints, fix context before touching dials. Then apply the diagnostic. If the model doesn't try hard enough, that means you need more effort. If it didn't know enough, upgrade to a bigger model. Third and finally, spend deliberately. Route routine work to smaller, cheaper models and reserve the frontier for problems that stretch it. Remembering that on the hardest tasks, the expensive model can be the cheaper one because it can take so

many fewer behind the scenes thinking tokens to come up with a solid answer for you.

14:23 Understanding these two dials, model size and inference time or thinking time will make you sharper at extracting value from every AI platform you touch. And given how much of data science workflow now runs through these models, that is leverage worth having for sure. So go match your dials to your tasks and make something great happen with the magical wizard powers we now all have. All right, and that's the end of today's episode. If you enjoyed it or know someone who might consider sharing this episode with them, leave a review of the show on your favorite podcasting platform or on YouTube. Tag me in a LinkedIn post with your thoughts and I'll respond to those. And of course, if you're not already subscriber, subscribe. Come on. The most important thing to me though is that you just keep on listening. I'm so grateful to have you listening and hope I can continue to make episodes you love for years and years to come till next time.

15:22 Keep on rocking it out there and I'm looking forward to enjoying another round of the SuperDataScience podcast with you, yes you, very soon.