# SDS PODCAST

# EPISODE 1012: THE OPEN-WEIGHT 2.8-TRILLION PARAMETER COMPETING AT THE FRONTIER

Jon Krohn: 00:00 This is episode number 1010, 1010. It's a fun one on the agent advisor strategy. Welcome back to the SuperDataScience Podcast. I'm your host, Jon Krohn. Today's topic is a clever new pattern called the advisor strategy for building cheaper, faster AI agents without much extra cost. And in particular, we'll delve into the advisor tool that Anthropic shipped to make that strategy a simple on - line code change for you. If you've built or deployed an LLM powered agent, a coding agent, a research pipeline, a document processing workflow, anything that loops through many steps calling tools along the way, you've run into a fundamental tension. The most capable frontier models produce the best plans and recover from errors most gracefully, but they're expensive to run on every single turn. The smaller, faster models are dramatically cheaper per token, but they fumble exactly the decisions that matter most, choosing an architecture, recognizing when an approach isn't converging, knowing when a task is actually done.

01:10 Most teams resolve this by picking one point on the cost capability spectrum and living with the trade-offs or by building custom routing logic that sends hard tasks to a big model and easy tasks to a small one, which works, but requires a real engineering effort and an extra orchestration layer. Back in April, Anthropic formalized a third option. Anthropic's advisor tool available in beta today via the Claude API pairs two models inside a single API call. And of course, I've got a link to all the documentation on this in the show notes. The first model is called the executor. We're going to be talking about the executor a lot in this episode. Maybe I should pronounce it executor. Yeah, let's go with that. The first model is the executor. It's a fast, cost efficient model like Claude Sonnet or even tiny super fast and super cheap Claude Haiku that drives the entire agent loop.

02:07 The executor reads tool results, decides which tool to call next, writes the code, drafts the report and generates

essentially all of the final output. The second model, in addition to the executor, this is the advisor. And it's a higher intelligence model such as Claude Opus or Fable that the executor can consult mid-task the same way it would call any other tool such as web search or a bash terminal. To anthropomorphize, it's like having a lower level worker call their higher paid supervisor for guidance when they run into a situation they're not 100% sure how to deal with. Mechanically, it works like this. You declare the advisor as a tool in your API request specifying which model should play the advisor role. When the executor decides it needs help, say before committing to a design approach or after repeated errors suggesting it's stuck, the executor admits an advisor call.

03:03 Interestingly, that call takes no parameters at all. The executor only signals the timing and anthropic servers automatically hand the advisor the full conversation transcript, including your system prompt, tool definitions, every prior turn and tool result, and even the text the executor has generated so far in the current turn. The advisor then runs as a separate inference pass server side and runs its guidance, typically just 400 to 700 tokens of advice with all of the advisor's internal thinking stripped out and the executor then carries on. Critically, all of this happens inside one API request. No orchestration framework, no extra network round trips, and no context switching overhead because both models are looking at the same conversation state. So in terms of something concrete, what do you get for this? Anthropic published benchmark results alongside the launch of this advisor tool and with the usual caveat that these are the vendor's own numbers on the vendor's own product, they're worth digging into.

04:04 On SweeBench Multilingual, a benchmark of realistic software bug fixing tasks across nine programming languages, Claude Sonnet with a Claude Opus advisor scored 2.7 percentage points higher than Sonet running

alone while simultaneously reducing cost per agentic task by 12%. That's worth repeating. Better quality and lower cost at the same time. That result seems counterintuitive. How can adding calls to a much more expensive model make the whole thing cheaper? The answer is that the advisor's output is tiny relative to everything else and a good plan delivered early prevents waste. Without the advisor, Sonnet takes more failed attempts, makes more misguided tool calls and runs longer conversations to reach the same destination. The advisor shortens the whole journey and the bulk of token generation still happens at the executor's lower rates. If we use Haiku, Anthropic's smallest model instead of Sonnet as the executor, then the results are even more striking.

05:04 On BrowseComp, a difficult web research benchmark, Haiku alone scored just about 20%. Haiku with an Opus advisor scored 41% more than double. That configuration still trails Sonnet running solo on quality, but it costs 85% less per task. The pattern across the data is clear. The weaker the baseline executor, the bigger the lift from adding an advisor. And that gives you two distinct ways to use this. If you're currently running Sonnet on complex tasks, adding an OpusClass advisor buys you a quality bump at similar or even lower cost. If you're running Haiku at high volume, an advisor buys a large step up in intelligence while staying far below the cost of switching to a bigger executor outright. Those April results were generated with Opus in the advisor seat, but last week Anthropic's developer team shared updated numbers with its newest and most capable model, Fable five, advising the newest Sonnet.

06:05 And these data, though again, Anthropic Zone, are what put this topic on my radar for today's episode. On SweeBench Pro, a harder successor to the SweeBench family run acros 482 tasks. Sonnet five on its own passed over 75% of tasks at 75 cents per task. Add Fable five as an advisor, consulted on average roughly once per task,

and the pass rate jumps to 84% from 75% at a cost that's about double $1.40 per task. Fable five running solo top the table at 92%, but at over $2 per task. Put another way, the Sonnet plus Fable pairing captured about 92% of Fable's standalone performance at just 63% of its cost. Now, Anthropic's documentation for this feature is candid about where the pattern works and where it doesn't. And I want to pass along a few of those practical lessons because they'll save you real time if you experiment with this.

07:06 First, the advisor is a poor fit for single turn question answering. If there's nothing to plan, there's nothing for an advisor to advise on. It shines instead on long horizon agentic work, coding agents, computer use, multi-step research where most turns are mechanical, but having an excellent plan is crucial. Second, executors tend to undercall the advisor by default, especially on coding tasks. So Anthropic publishes suggested system prompt language to encourage calls at the two moments that matter most early before the executor commits to an approach and late before it declares the task complete. Anthropic even quantifies the interventions by model. A simple mid-conversation nudge reminding the executor to consult the advisor, raised task pass rates by roughly seven percentage points on HighQ executors, had no measurable effect on Sonnet, and slightly lowered pass rates on Opus executors, which apparently already know when to ask for help.

08:05 Third and finally, there are levers for keeping the advisor's cost contained. Setting a token cap of 2048 on the advisor's output reduced mean advisor output by roughly sevenfold in anthropics testing with essentially no truncation and no detectable quality loss. And advisor side prompt caching breaks even at around three advisor calls per conversation. So it's worth enabling for long agent loops, but not for short tasks. One more important implementation note that's probably obvious, but I'll

make explicit anyway. The advisor must be at least as capable as the executor, right? That's so obvious. You can't have Haiku advising fable.

08:48 And this pattern isn't confined to the API. It has rolled into Claude Code as a simple toggle and within days of launch, feature requests to support the advisor tool were popping up across developer ecosystems, which tells you the pattern resonated immediately. So beyond anthropic, how does this compare with what the rest of the industry is doing? The most instructive comparison is with OpenAI, which has been attacking the same cost capability tension from a different angle. Since GPT-5, ChatGPT has used a real-time router, a system that examines each incoming query and decides upfront whether to send it to a fast high throughput model or a deeper reasoning model based on the conversation type, the complexity, tool needs, and explicit user intent. OpenAI trains the router continuously on signals like when users manually switch models and which responses they prefer. It's a smart approach for a consumer chat product, but the philosophy is quite different.

09:42 OpenAI's router makes a dispatch decision before generation begins and the platform makes that decision for you. Anthropics advisor inverts this. The cheap model runs the show and escalates itself mid task on its own judgment with the full shared context of everything it has already tried. Routing decides who answers. Advising lets the worker phone a more intelligent, higher paid supervisor halfway through the job using that same anthropomorphic analogy that I provided earlier in the episode. Overall, the pattern, this idea of having an advisor is also spreading beyond any single vendor. So you're not limited to doing this with just anthropic models or just OpenAI models. In June, OpenRouter, the popular multi-provider model gateway, which we use at my consulting firm, Y Carat, for tons of use cases, OpenRouter launched its own cross-provider advisor tool

where any model on the platform can serve as the executor and any model from any provider can serve as the advisor as well.

10:44 So you could run a Gemini executor that consults Claude or a GPT executor that consults a DeepSeek. Their pitch, Open Routers pitch, leans on the enormous price spreads between model tiers. They cite a 67X per token gap between a budget model and a frontier advisor because most requests don't need frontier level reasoning. So why pay frontier prices for anything but the moments that do? And of course, plenty of teams have been hand rolling this kind of escalation logic for months. The advisor tool essentially productizes it with the orchestration handled server side. The takeaway for you if you're building with LLMs, this is one of the cheapest experiments in AI engineering right now with one of the largest potential payoffs, close to a one line change to an existing API call. Anthropic's recommendation is to run your evaluation suite three ways, executor solo, executor with advisor and advisor solo, and let your own data, which of course vary by workload, tell you where the quality cost frontier sits for your use case.

11:48 We've got links to Anthropics announcement, the advisor tool documentation and open router's cross provider take in the show notes for you to check out. Zooming out, I think the deeper significance here is that frontier AI progress is no longer only about training ever bigger models. It's increasingly about smarter economics and how we compose the models we already have. A system where a fast model does the work and a brilliant model supplies judgment at exactly the right moments looks a lot like how well-run human teams operate, and it's now available to you as a simple API parameter. Cool. Another powerful quiver for your bow. All right, that's it for today's episode. If you enjoyed it or know someone who might, please share this episode with them. Leave a review of the show on your favorite podcasting platform or on YouTube.

I can't tell you how important that is for spreading this show.

12:43 If you can write an Apple Podcast review, that's especially helpful for getting the word out about this podcast. If you have any thoughts about this episode that you'd like me to see and respond to, tag me in a LinkedIn post and obviously subscribe to this show if you're not already a subscriber. Most importantly, however, we hope you'll just keep on listening. Until next time, keep on rocking it out there and I'm looking forward to enjoying another round of the SuperDataScience podcast with you very soon.