

May 2025

Search posts from May 2025

Search

105 posts: [11 entries](#), [59 links](#), [14 quotes](#), [21 notes](#)

May 1, 2025

You also mentioned the whole Chatbot Arena thing, which I think is interesting and points to the challenge around how you do benchmarking. How do you know what models are good for which things?

One of the things we've generally tried to do over the last year is anchor more of our models in our Meta AI product north star use cases. The issue with open source benchmarks, and any given thing like the LM Arena stuff, is that they're often skewed toward a very specific set of uses cases, which are often not actually what any normal person does in your product. [...]

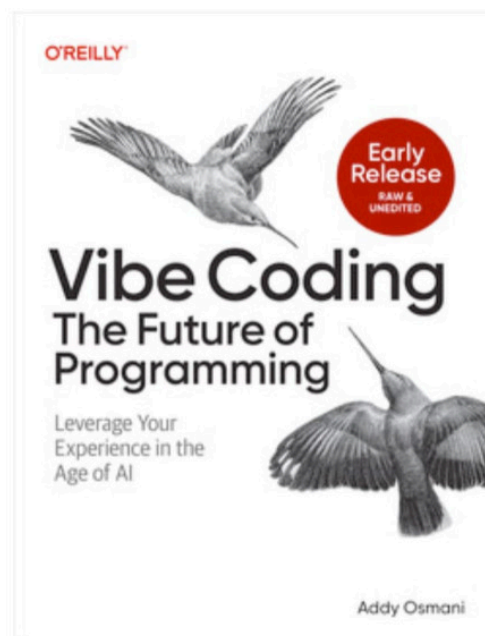
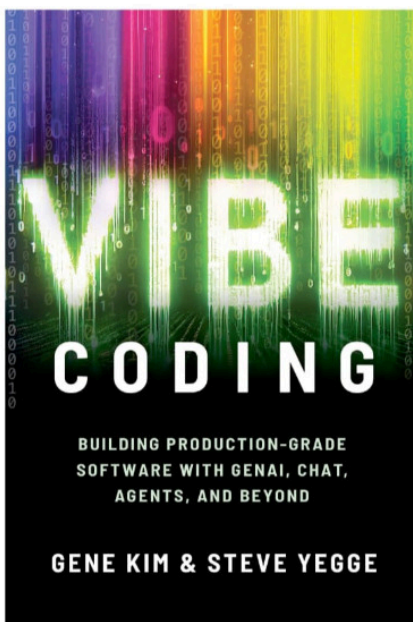
So we're trying to anchor our north star on the product value that people report to us, what they say that they want, and what their revealed preferences are, and using the experiences that we have. Sometimes these benchmarks just don't quite line up. I think a lot of them are quite easily gameable.

On the Arena you'll see stuff like [Sonnet 3.7](#), which is a great model, and it's not near the top. It was relatively easy for our team to tune a version of Llama 4 Maverick that could be way at the top. But the version we released, the pure model, actually has no tuning for that at all, so it's further down. So you just need to be careful with some of these benchmarks. We're going to index primarily on the products.

— [Mark Zuckerberg](#), on Dwarkesh Patel's podcast

[12:28 am](#) / [meta](#), [generative-ai](#), [llama](#), [mark-zuckerberg](#), [ai](#), [chatbot-arena](#), [llms](#)

Two publishers and three authors fail to understand what “vibe coding” means



Vibe coding does not mean “using AI tools to help write code”. It means “generating code with AI without caring about the code that is produced”. See [Not all AI-assisted programming is vibe coding](#) for my previous writing on this subject. This is a hill I am willing to die on. I fear it will be the death of me.

[... [908 words](#)]

[2:26 pm](#) / [books](#), [ai](#), [generative-ai](#), [llms](#), [ai-assisted-programming](#), [vibe-coding](#), [semantic-diffusion](#)

[Redis is open source again](#) ([via](#)) Salvatore Sanfilippo:

Five months ago, I rejoined Redis and quickly started to talk with my colleagues about a possible switch to the AGPL license, only to discover that there was already an ongoing discussion, a very old one, too. [...]

I'll be honest: I truly wanted the code I wrote for the new Vector Sets data type to be released under an open source license. [...]

So, honestly, while I can't take credit for the license switch, I hope I contributed a little bit to it, because today I'm happy. I'm happy that Redis is open source software again, under the terms of the AGPLv3 license.

I'm absolutely *thrilled* to hear this. Redis 8.0 is [out today under the new license](#), including a beta release of [Vector Sets](#). I've been watching Salvatore's work on those with [fascination](#), while sad that I probably wouldn't use it often due to the janky license. That concern is now gone. I'm looking forward to putting them through their paces!

See also [Redis is now available under the AGPLv3 open source license](#) on the Redis blog. An interesting note from that is that they are also:

Integrating Redis Stack technologies, including JSON, Time Series, probabilistic data types, Redis Query Engine and more into core Redis 8 under AGPL

That's a whole bunch of new things that weren't previously part of Redis core.

I hadn't encountered [Redis Query Engine](#) before - it looks like that's a whole set of features that turn Redis into more of an Elasticsearch-style document database complete with full-text, vector search operations and geospatial operations and aggregations. It supports search syntax that looks a bit like this:

```
FT.SEARCH places "museum @city:(san francisco|oakland) @shape:[CONTAINS $poly]" PARAMS 2 poly 'POLYGON((-122.5 37.7, -122.5 37.8, -122.4 37.8, -122.4 37.7, -122.5 37.7))' DIALECT 3
```

(Noteworthy that Elasticsearch chose the AGPL too when they switched back from the SSPL to an open source license [last year](#)).

[# 5:19 pm](#) / [open-source](#), [redis](#), [salvatore-sanfilippo](#), [vector-search](#)

[Making PyPI's test suite 81% faster](#) ([via](#)) Fantastic collection of tips from Alexis Challande on speeding up a Python CI workflow.

I've used [pytest-xdist](#) to run tests in parallel (across multiple cores) before, but the following tips were new to me:

- `COVERAGE_CORE=sysmon` pytest `--cov=myproject` tells [coverage.py](#) on Python 3.12 and higher to use the new [sys.monitoring](#) mechanism, which knocked their test execution time down from 58s to 27s.
- Setting `testpaths = ["tests/"]` in `pytest.ini` lets pytest skip scanning other folders when trying to find tests.
- `python -X importtime ...` shows a trace of exactly how long every package took to import. I could have done with this last week when I was trying to [debug slow LLM startup time](#) which turned out to be caused by heavy imports.

I was grumbling to myself about how if we're going to [give in](#), ditch the proper definition and use "vibe coding" to refer to all forms of AI-assisted programming, where do we draw the line?

Is it "vibe coding" if my IDE suggests the completion of a single line of code? How about if I copy and paste in a three line "escape HTML characters" function from ChatGPT? What if I copy and paste some code from StackOverflow that it turns out was AI-generated by someone else? How much AI-assistance does it take to switch from programming to "vibe coding"?

Then I realized that the answer was staring me in the face. **There is no clear line. It's all in the vibes.**

May 2, 2025

[Expanding on what we missed with sycophancy](#). I criticized OpenAI's [initial post](#) about their recent ChatGPT sycophancy rollback as being "[relatively thin](#)" so I'm delighted that they have followed it with a much more in-depth explanation of what went wrong. This is worth spending time with - it includes a detailed description of how they create and test model updates.

This feels reminiscent to me of a good outage [postmortem](#), except here the incident in question was an AI personality bug!

The custom GPT-4o model used by ChatGPT has had five major updates since it was first launched. OpenAI start by providing some clear insights into how the model updates work:

To post-train models, we take a pre-trained base model, do supervised fine-tuning on a broad set of ideal responses written by humans or existing models, and then run reinforcement learning with reward signals from a variety of sources.

During reinforcement learning, we present the language model with a prompt and ask it to write responses. We then rate its response according to the reward signals, and update the language model to make it more likely to produce higher-rated responses and less likely to produce lower-rated responses.

Here's yet more evidence that the entire AI industry runs on "vibes":

In addition to formal evaluations, internal experts spend significant time interacting with each new model before launch. We informally call these "vibe checks"—a kind of human sanity check to catch issues that automated evals or A/B tests might miss.

So what went wrong? Highlights mine:

In the April 25th model update, we had candidate improvements to better incorporate user feedback, memory, and fresher data, among others. **Our early assessment is that each of these changes, which had looked beneficial individually, may have played a part in tipping the scales on sycophancy when combined.** For example, the update introduced **an additional reward signal based on user feedback—thumbs-up and thumbs-down data from ChatGPT.** This signal is often useful; a thumbs-down usually means something went wrong.

But we believe in aggregate, **these changes weakened the influence of our primary reward signal, which had been holding sycophancy in check.** User feedback in particular can sometimes favor more agreeable responses, likely amplifying the shift we saw.

I'm surprised that this appears to be first time the thumbs up and thumbs down data has been used to influence the model in this way - they've been collecting that data for a couple of years now.

I've been very suspicious of the new "memory" feature, where ChatGPT can use context of previous conversations to influence the next response. It looks like that may be part of this too, though not definitively the cause of the sycophancy bug:

We have also seen that in some cases, user memory contributes to exacerbating the effects of sycophancy, although we don't have evidence that it broadly increases it.

The biggest miss here appears to be that they let their automated evals and A/B tests overrule those vibe checks!

One of the key problems with this launch was that our offline evaluations—especially those testing behavior—generally looked good. Similarly, the A/B tests seemed to indicate that the small number of users who tried the model liked it. [...] Nevertheless, some expert testers had indicated that the model behavior “felt” slightly off.

The [system prompt change](#) I wrote about the other day was a temporary fix while they were rolling out the new model:

We took immediate action by pushing updates to the system prompt late Sunday night to mitigate much of the negative impact quickly, and initiated a full rollback to the previous GPT-4o version on Monday

They list a set of sensible new precautions they are introducing to avoid behavioral bugs like this making it to production in the future. Most significantly, it looks we are finally going to get release notes!

We also made communication errors. Because we expected this to be a fairly subtle update, we didn't proactively announce it. Also, our release notes didn't have enough information about the changes we'd made. Going forward, we'll proactively communicate about the updates we're making to the models in ChatGPT, whether “subtle” or not.

And model behavioral problems will now be treated as seriously as other safety issues.

We need to treat model behavior issues as launch-blocking like we do other safety risks. [...] We now understand that personality and other behavioral issues should be launch blocking, and we're modifying our processes to reflect that.

This final note acknowledges how much more responsibility these systems need to take on two years into our weird consumer-facing LLM revolution:

One of the biggest lessons is fully recognizing how people have started to use ChatGPT for deeply personal advice—something we didn't see as much even a year ago. At the time, this wasn't a primary focus, but as AI and society have co-evolved, it's become clear that we need to treat this use case with great care.

4:57 pm / [ai](#), [postmortem](#), [openai](#), [generative-ai](#), [chatgpt](#), [llms](#), [ai-ethics](#), [ai-personality](#), [system-prompts](#)

It's not in their [release notes](#) yet but Anthropic pushed some big new features today. [Alex Albert](#):

We've improved web search and rolled it out worldwide to all paid plans. Web search now combines light Research functionality, allowing Claude to automatically adjust search depth based on your question.

Anthropic announced [Claude Research](#) a few weeks ago as a product that can combine web search with search against your private Google Workspace - I'm not clear on how much of that product we get in this "light Research" functionality.

I'm most excited about [this detail](#):

You can also drop a web link in any chat and Claude will fetch the content for you.

In my experiments so far the user-agent it uses is Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko; compatible; Claude-User/1.0; +Claude-User@anthropic.com). It appears to obey robots.txt.

[# 6:45 pm](#) / [alex-albert](#), [anthropic](#), [claude](#), [llm-tool-use](#), [deep-research](#), [ai](#), [llms](#), [generative-ai](#), [user-agents](#)

Having tried a few of the [Qwen 3 models](#) now my favorite is a bit of a surprise to me: I'm really enjoying [Qwen3-8B](#).

I've been running prompts through the MLX 4bit quantized version, [mlx-community/Qwen3-8B-4bit](#). I'm using [llm-mlx](#) like this:

```
llm install llm-mlx
llm mlx download-model mlx-community/Qwen3-8B-4bit
```

This pulls 4.3GB of data and saves it to `~/.cache/huggingface/hub/models--mlx-community--Qwen3-8B-4bit`.

I assigned it a default alias:

```
llm aliases set q3 mlx-community/Qwen3-8B-4bit
```

I also added a default option for that model - this saves me from adding `-o unlimited 1` to every prompt which disables the default output token limit:

```
llm models options set q3 unlimited 1
```

And now I can run prompts:

```
llm -m q3 'brainstorm questions I can ask my friend who I think is secretly from Atlantis that will not tip her off to my suspicions'
```

Qwen3 is a "reasoning" model, so it starts each prompt with a `<think>` block containing its chain of thought. Reading these is always *really fun*. Here's the full response I got for [the above question](#).

I'm finding Qwen3-8B to be surprisingly capable for useful things too. It can [summarize short articles](#). It can [write simple SQL queries](#) given a question and a schema. It can [figure out what a simple web app does](#) by reading the HTML and JavaScript. It can [write Python code](#) to meet a paragraph long spec - for that one it "reasoned" for an unreasonably long time but it did eventually get to a useful answer.

All this while consuming between 4 and 5GB of memory, depending on the length of the prompt.

I think it's pretty extraordinary that a few GBs of floating point numbers can usefully achieve these various tasks, especially using so little memory that it's not an imposition on the rest of the things I want to run on my laptop at the same time.

[# 11:41 pm](#) / [llm](#), [models](#), [qwen](#), [mlx](#), [generative-ai](#), [ai](#), [local-llms](#), [llm-reasoning](#), [ai-in-china](#)

May 4, 2025

[DuckDB is Probably the Most Important Geospatial Software of the Last Decade](#). Drew Breunig argues that the ease of installation of DuckDB is opening up geospatial analysis to a whole new set of developers.

This inspired [a comment on Hacker News](#) from DuckDB Labs geospatial engineer Max Gabrielsson which helps explain why the drop in friction introduced by DuckDB is so significant:

I think a big part is that duckdb's spatial extension provides a SQL interface to a whole suite of standard foss gis packages by statically bundling everything (including inlining the default PROJ database of coordinate projection systems into the binary) and providing it for multiple platforms (including WASM). I.E there are no transitive dependencies except libc.

[...] the fact that you can e.g. convert too and from a myriad of different geospatial formats by utilizing GDAL, transforming through SQL, or pulling down the latest overture dump without having the whole workflow break just cause you updated QGIS has probably been the main killer feature for a lot of the early adopters.

I've lost count of the time I've spent fiddling with dependencies like GDAL trying to get various geospatial tools to work in the past. Bundling difficult dependencies statically is an under-appreciated trick!

If the bold claim in the headline inspires you to provide a counter-example, bear in mind that a decade ago is 2015, and most of the key technologies in the modern geospatial stack - QGIS, PostGIS, geopandas, SpatiaLite - predate that by quite a bit.

[12:28 am](#) / [geospatial](#), [gis](#), [sql](#), [duckdb](#), [drew-breunig](#)

Our local BBQ spot here in El Granada - [Breakwater Barbecue](#) - had a soft opening this weekend in their [new location](#).

Here's the new building. They're still working on replacing the sign from the previous restaurant occupant:



It's actually our old railway station! From 1905 to 1920 the [Ocean Shore Railroad](#) ran steam trains from San Francisco down through Half Moon Bay most of the way to Santa Cruz, though they never quite connected the two cities.

The restaurant has some photos on the wall of the old railroad. Here's what that same building looked like >100 years ago.



5:09 pm / [half-moon-bay](#), [photos](#), [history](#)

[Dummy's Guide to Modern LLM Sampling](#) (via) This is an extremely useful, detailed set of explanations by [@AlpinDale](#) covering the various different sampling strategies used by modern LLMs. LLMs return a set of next-token probabilities for every token in their corpus - a layer above the LLM can then use sampling strategies to decide which one to use.

I finally feel like I understand the difference between [Top-K](#) and [Top-P](#)! Top-K is when you narrow down to e.g. the 20 most likely candidates for next token and then pick one of those. Top-P instead "the smallest set of words whose combined probability exceeds threshold P" - so if you set it to 0.5 you'll filter out tokens in the lower half of the probability distribution.

There are a bunch more sampling strategies in here that I'd never heard of before - Top-A, Top-N-Sigma, Epsilon-Cutoff and more.

Reading the descriptions here of [Repetition Penalty](#) and [Don't Repeat Yourself](#) made me realize that I need to be a little careful with those for some of my own uses of LLMs.

I frequently feed larger volumes of text (or code) into an LLM and ask it to output subsets of that text as direct quotes, to answer questions like "which bit of this code handles authentication tokens" or "show me direct quotes that illustrate the main themes in this conversation".

Careless use of frequency penalty strategies might go against what I'm trying to achieve with those prompts.

9:13 pm / [ai](#), [prompt-engineering](#), [generative-ai](#), [llms](#), [tokenization](#)

[May 5, 2025](#)

[On using generative AI for work despite the risk of errors:]

- AI is helpful despite being error-prone if it is faster to verify the output than it is to do the work yourself. For example, if you're using it to find a product that matches a given set of specifications, verification may be a lot faster than search.
- There are many uses where errors don't matter, like using it to enhance creativity by suggesting or critiquing ideas.
- At a meta level, if you use AI without a plan and simply turn to AI tools when you feel like it, then you're unlikely to be able to think through risks and mitigations. It is better to identify concrete ways to integrate AI into your workflows, with known benefits and risks, that you can employ repeatedly.

— [Arvind Narayanan](#)

4:11 pm / [llms](#), [ai](#), [arvind-narayanan](#), [generative-ai](#)

[Feed a video to a vision LLM as a sequence of JPEG frames on the CLI \(also LLM 0.25\)](#)



The new [llm-video-frames](#) plugin can turn a video file into a sequence of JPEG frames and feed them directly into a long context vision LLM such as GPT-4.1, even when that LLM doesn't directly support video input. It depends on a plugin feature I added to [LLM 0.25](#), which I released last night.

[... [1,600 words](#)]

Two things can be true simultaneously: (a) LLM provider cost economics are too negative to return positive ROI to investors, and (b) LLMs are useful for solving problems that are meaningful and high impact, albeit not to the AGI hype that would justify point (a). This particular combination creates a frustrating gray area that requires a nuance that an ideologically split social media can no longer support gracefully. [...]

OpenAI collapsing would not cause the end of LLMs, because LLMs are useful *today* and there will always be a nonzero market demand for them: it's a bell that can't be unrung.

— [Max Woolf](#)

I'm disappointed at how little good writing there is out there about effective prompting.

Here's an example: what's the best prompt to use to summarize an article?

That feels like such an obvious thing, and yet I haven't even seen *that* being well explored!

It's actually a surprisingly deep topic. I like using tricks like "directly quote the sentences that best illustrate the overall themes" and "identify the most surprising ideas", but I'd love to see a thorough breakdown of all the tricks I haven't seen yet.

[May 6, 2025](#)

[What people get wrong about the leading Chinese open models: Adoption and censorship](#) ([via](#)) While I've been enjoying [trying out Alibaba's Qwen 3](#) a lot recently, Nathan Lambert focuses on the elephant in the room:

People vastly underestimate the number of companies that cannot use Qwen and DeepSeek open models because they come from China. This includes on-premise solutions built by people who know the fact that model weights alone cannot reveal anything to their creators.

The root problem here is the closed nature of the training data. Even if a model is open weights, it's not possible to conclusively determine that it couldn't add backdoors to generated code or trigger "indirect influence of Chinese values on Western business systems". Qwen 3 certainly has baked in opinions about the status of Taiwan!

Nathan sees this as an opportunity for other liberally licensed models, including his own team's OLMo:

This gap provides a big opportunity for Western AI labs to lead in open models. Without DeepSeek and Qwen, the top tier of models we're left with are Llama and Gemma, which both have very restrictive licenses when compared to their Chinese counterparts. These licenses are proportionally likely to block an IT department from approving a model.

This takes us to the middle tier of permissively licensed, open weight models who actually have a huge opportunity ahead of them: OLMo, of course, I'm biased, Microsoft with Phi, Mistral, IBM (!?!), and some other smaller companies to fill out the long tail.

That's it. I've had it. I'm putting my foot down on this craziness.

1. Every reporter submitting security reports on #Hackerone for #curl now needs to answer this question:

"Did you use an AI to find the problem or generate this submission?"

(and if they do select it, they can expect a stream of proof of actual intelligence follow-up questions)

2. We now ban every reporter INSTANTLY who submits reports we deem AI slop. A threshold has been reached. We are effectively being DDoSed. If we could, we would charge them for this waste of our time.

We still have not seen a single valid security report done with AI help.

— [Daniel Stenberg](#)

3:12 pm / [ai](#), [llms](#), [ai-ethics](#), [daniel-stenberg](#), [slop](#), [security](#), [curl](#), [generative-ai](#)

[Gemini 2.5 Pro Preview: even better coding performance](#). New Gemini 2.5 Pro "Google I/O edition" model, released a few weeks ahead of that annual developer conference.

They claim even better frontend coding performance, highlighting their #1 ranking on the [WebDev Arena leaderboard](#), notable because it knocked Claude 3.7 Sonnet from that top spot. They also highlight "state-of-the-art video understanding" with a 84.8% score on the new-to-me [VideoMME benchmark](#).

I rushed out a [new release of llm-gemini](#) adding support for the new `gemini-2.5-pro-preview-05-06` model ID, but it turns out if I had read to the end of their post I should not have bothered:

For developers already using Gemini 2.5 Pro, this new version will not only improve coding performance but will also address key developer feedback including reducing errors in function calling and improving function calling trigger rates. The previous iteration (03-25) now points to the most recent version (05-06), so no action is required to use the improved model

I'm not a fan of this idea that a model ID with a clear date in it like `gemini-2.5-pro-preview-03-25` can suddenly start pointing to a brand new model!

I used the new Gemini 2.5 Pro to summarize the conversation about itself on Hacker News using the latest version of [my hn-summary.sh script](#):

```
hn-summary.sh 43906018 -m gemini-2.5-pro-preview-05-06
```

Here's [what I got back](#) - 30,408 input tokens, 8,535 output tokens and 3,980 thinknig tokens for a total cost of 16.316 cents.

8,535 output tokens is a *lot*. My system prompt includes the instruction to "Go long" - this is the first time I've seen a model really take that to heart. For comparison, here's [the result](#) of a similar experiment against the previous version of Gemini 2.5 Pro two months ago.

Update: The *one time* I forget to run my "Generate an SVG of a pelican riding a bicycle" test is the time that the model turns out to produce one of the best results I've seen yet!



Here's [the transcript](#) - 11 input tokens and 3,281 output tokens and 1,558 thinking tokens = 4.8404 cents.

I asked Gemini to describe that image:

```
llm -m gemini-2.5-pro-preview-05-06 \  
-a https://static.simonwillison.net/static/2025/gemini-latest-pelican.jpg \  
'describe image for alt text'
```

Here's [what I got back](#). Gemini thought it had drawn a duck:

A cartoon illustration of a white duck with an orange beak riding a blue bicycle.

The duck has a large, oval white body and a smaller round head with a black dot eye. Its thin black wings act as arms, gripping the blue handlebars. One yellow-orange leg is visible, bent and pushing a grey pedal.

The bicycle has a blue frame with a distinctive cross-brace, a brown oval seat, and dark grey wheels with silver spokes. The entire image is set against a plain white background.

[Saying “hi” to Microsoft’s Phi-4-reasoning](#)

Microsoft released a new sub-family of models a few days ago: Phi-4 reasoning. They introduced them in [this blog post](#) celebrating a year since the release of Phi-3:

[... [1,498 words](#)]

6:25 pm / [microsoft](#), [generative-ai](#), [local-llms](#), [llms](#), [llm](#), [phi](#), [qwen](#), [ollama](#), [llm-reasoning](#), [llm-release](#), [ai-in-china](#)

[What’s the carbon footprint of using ChatGPT?](#) Inspired by Andy Masley's [cheat sheet](#) (which I [linked to](#) last week) Hannah Ritchie explores some of the numbers herself.

Hannah is Head of Research at Our World in Data, a Senior Researcher at the University of Oxford ([bio](#)) and maintains a [prolific newsletter](#) on energy and sustainability so she has a *lot* more credibility in this area than Andy or myself!

My sense is that a lot of climate-conscious people feel guilty about using ChatGPT. In fact it goes further: I think many people judge others for using it, because of the perceived environmental impact. [...]

But after looking at the data on individual use of LLMs, I have stopped worrying about it and I think you should too.

The inevitable counter-argument to the idea that the impact of ChatGPT usage by an individual is negligible is that aggregate user demand is still the thing that drives these enormous investments in huge data centers and new energy sources to power them. Hannah acknowledges that:

I am *not* saying that AI energy demand, on aggregate, is not a problem. It is, even if it’s “just” of a similar magnitude to the other sectors that we need to electrify, such as cars, heating, or parts of industry. It’s just that individuals querying chatbots is a relatively small part of AI’s total energy consumption. That’s how both of these facts can be true at the same time.

Meanwhile Arthur Clune [runs the numbers](#) on the potential energy impact of some much more severe usage patterns.

Developers burning through \$100 of tokens per day (not impossible given some of the LLM-heavy development patterns that are beginning to emerge) could end the year with the equivalent of a short haul flight or 600 mile car journey.

In the panopticon scenario where all 10 million security cameras in the UK analyze video through a vision LLM at one frame per second Arthur estimates we would need to duplicate the total usage of Birmingham, UK - the output of a 1GW nuclear plant.

Let’s not build that panopticon!

7:47 pm / [ai](#), [generative-ai](#), [chatgpt](#), [llms](#), [ai-assisted-programming](#), [vision-llms](#), [ai-ethics](#), [ai-energy-usage](#)

[May 7, 2025](#)

[astral-sh/ty](#) (via) Astral have been working on this "extremely fast Python type checker and language server, written in Rust" [quietly but in-the-open](#) for a while now. Here’s the first alpha public release - albeit [not yet announced](#) - as [ty](#) on PyPI (nice [donated](#) two-letter name!)

You can try it out via [uvx](#) like this - run the command in a folder full of Python code and see what comes back:

uvx ty check

I got zero errors for my recent, simple [condense-json](#) library and a *ton* of errors for my more mature [sqlite-utils](#) library - [output here](#).

It really is *fast*:

```
cd /tmp
git clone https://github.com/simonw/sqlite-utils
cd sqlite-utils
time uvx ty check
```

Reports it running in around a tenth of a second (0.109 total wall time) using multiple CPU cores:

```
uvx ty check 0.18s user 0.07s system 228% cpu 0.109 total
```

Running `time uvx mypy .` in the same folder (both after first ensuring the underlying tools had been cached) took around 7x longer:

```
uvx mypy . 0.46s user 0.09s system 74% cpu 0.740 total
```

This isn't a fair comparison yet as ty still isn't feature complete in comparison to mypy.

6:37 pm / [pypi](#), [python](#), [rust](#), [mypy](#), [uv](#), [astral](#)

[llm-prices.com](#). I've been maintaining a simple LLM pricing calculator since [October last year](#). I finally decided to split it out to its own domain name (previously it was hosted at `tools.simonwillison.net/llm-prices`), running on Cloudflare Pages.

LLM pricing calculator

Number of input tokens (aka prompt tokens):

1233

Number of output tokens (aka completion tokens):

83

Cost per million input tokens (\$):

0.075

Cost per million output tokens (\$):

0.3

Total cost: \$0.000117

Total cost: 0.0117 cents

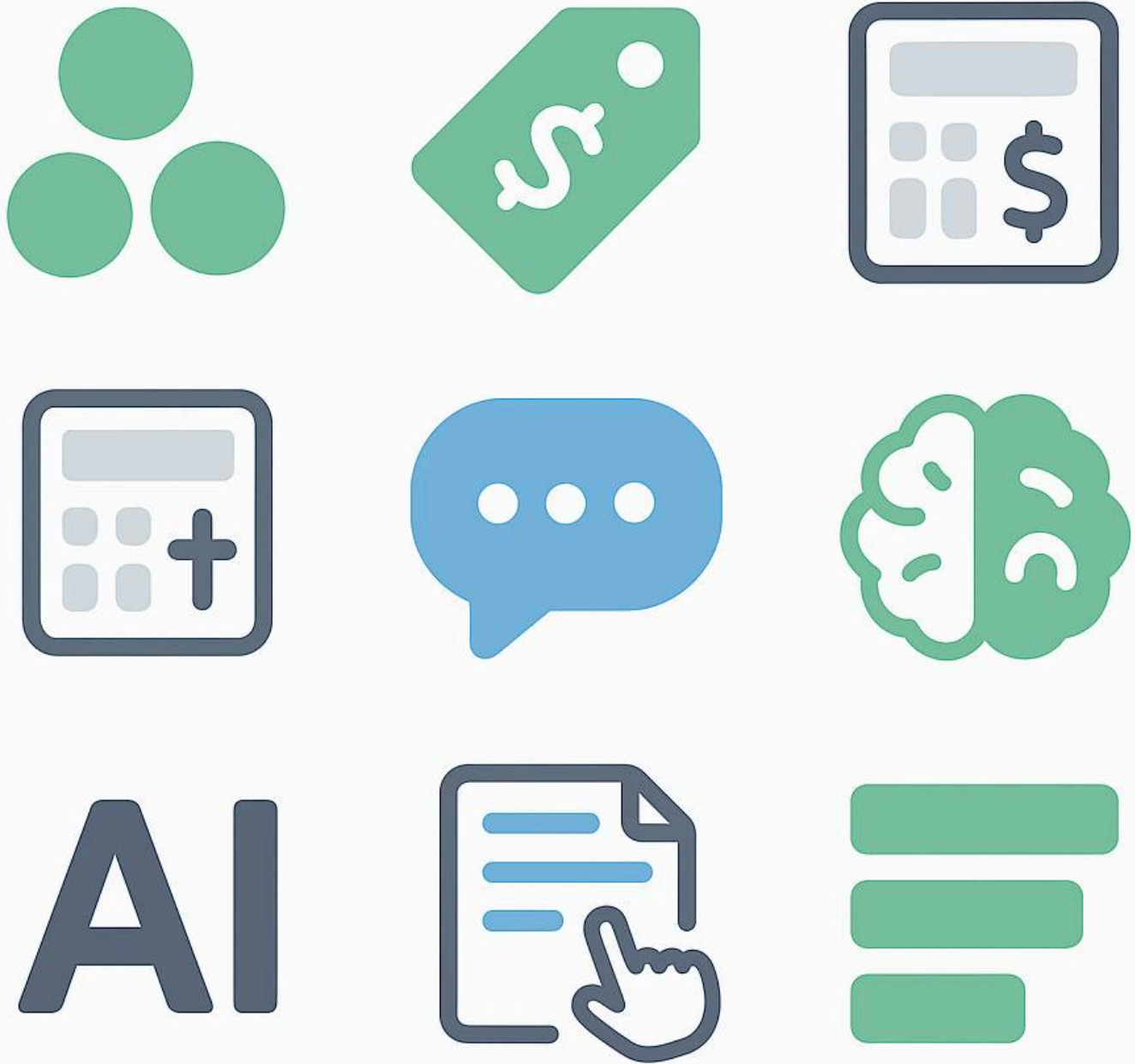
Model prices (per million tokens)

Model	Input cost	↑	Output cost
Amazon Nova Micro	\$0.035		\$0.14
Gemini 1.5 Flash-8B ≤128k	\$0.0375		\$0.15
Ministral 3B 24.10	\$0.04		\$0.04
Amazon Nova Lite	\$0.06		\$0.24
Gemini 2.0 Flash Lite	\$0.075		\$0.30
Gemini 1.5 Flash ≤128k	\$0.075		\$0.30
Gemini 1.5 Flash-8B >128k	\$0.075		\$0.30
Gemini 2.0 Flash	\$0.10		\$0.40
Mistral Small 3.1	\$0.10		\$0.30
Ministral 8B 24.10	\$0.10		\$0.10

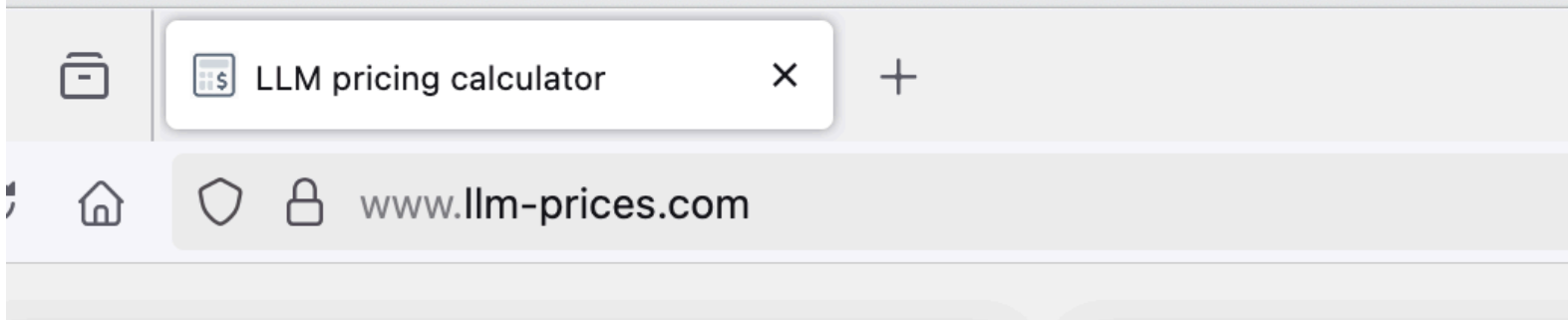
The site runs out of my [simonw/llm-prices](#) GitHub repository. I ported [the history](#) of the old `llm-prices.html` file using a vibe-coded bash script that I forgot to save anywhere.

I rarely use AI-generated imagery in my own projects, but for this one I found an excellent reason to use GPT-4o image outputs... to generate the favicon! I dropped a screenshot of the site into ChatGPT (o4-mini-high in this case) and asked for the following:

design a bunch of options for favicons for this site in a single image, white background



I liked the top right one, so I cropped it into Pixelmator and made a 32x32 version. Here's what it looks like in my browser:



LLM pricing calculator

Number of input tokens (aka prompt tokens):

Model

I added a new feature just now: the state of the calculator is now reflected in the #fragment-hash URL of the page, which means you can link to your previous calculations.

I implemented that feature using [the new gemini-2.5-pro-preview-05-06](#), since that model boasts improved front-end coding abilities. It did a pretty great job - here's how I prompted it:

```
llm -m gemini-2.5-pro-preview-05-06 -f https://www.llm-prices.com/ -s 'modify this code so that the state of the page is reflected in the fragment hash URL - I want to capture the values filling out the form fields and also the current sort order of the table. These should be respected when the page first loads too. Update them using replaceHistory, no need to enable the back button.'
```

Here's [the transcript](#) and [the commit updating the tool](#), plus [an example link](#) showing the new feature in action (and calculating the cost for that Gemini 2.5 Pro prompt at 16.8224 cents, after [fixing the calculation](#).)

[# 8:15 pm](#) / [favicons](#), [projects](#), [ai](#), [cloudflare](#), [generative-ai](#), [llms](#), [ai-assisted-programming](#), [gemini](#), [llm-pricing](#), [text-to-image](#), [vibe-coding](#)

[Medium is the new large](#). New model release from Mistral - this time closed source/proprietary. Mistral Medium claims strong benchmark scores similar to GPT-4o and Claude 3.7 Sonnet, but is priced at \$0.40/million input and \$2/million output - about the same price as GPT 4.1 Mini. [For comparison](#), GPT-4o is \$2.50/\$10 and Claude 3.7 Sonnet is \$3/\$15.

The model is a vision LLM, accepting both images and text.

More interesting than the price is the deployment model. Mistral Medium may not be open weights but it is very much available for self-hosting:

Mistral Medium 3 can also be deployed on any cloud, including self-hosted environments of four GPUs and above.

Mistral's other announcement today is [Le Chat Enterprise](#). This is a suite of tools that can integrate with your company's internal data and provide "agents" (these look similar to Claude Projects or OpenAI GPTs), again with the option to self-host.

Is there a new open weights model coming soon? This note tucked away at the bottom of the Mistral Medium 3 announcement seems to hint at that:

With the launches of [Mistral Small](#) in March and Mistral Medium today, it's no secret that we're working on something 'large' over the next few weeks. With even our medium-sized model being resoundingly better than flagship open source models such as Llama 4 Maverick, we're excited to 'open' up what's to come :)

I released [llm-mistral 0.12](#) adding support for the new model.

9:14 pm / [ai](#), [generative-ai](#), [llms](#), [llm](#), [mistral](#), [vision-llms](#), [llm-pricing](#), [llm-release](#)

[Create and edit images with Gemini 2.0 in preview](#) (via) Gemini 2.0 Flash has had image generation capabilities for a while now, and they're now available via the paid Gemini API - at 3.9 cents per generated image.

According to [the API documentation](#) you need to use the new `gemini-2.0-flash-preview-image-generation` model ID and specify `{"responseModalities":["TEXT","IMAGE"]}` as part of your request.

Here's an example that calls the API using `curl` (and fetches a Gemini key from the `llm keys get store`):

```
curl -s -X POST \
  "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash-preview-image-generation:generateContent?
key=$(llm keys get gemini)" \
  -H "Content-Type: application/json" \
  -d '{
    "contents": [{
      "parts": [
        {"text": "Photo of a raccoon in a trash can with a paw-written sign that says I love trash"}
      ]
    }],
    "generationConfig":{"responseModalities":["TEXT","IMAGE"]}
  }' > /tmp/raccoon.json
```

Here's [the response](#). I got Gemini 2.5 Pro [to vibe-code me](#) a new [debug tool](#) for visualizing that JSON. If you visit that tool and click the "Load an example" link you'll see the result of the raccoon image visualized:

Render JSON from Gemini Image Generation

Paste Gemini JSON here:

```
{
  "candidates": [
    {
      "content": {
        "parts": [
          {
            "inlineData": {
              "mimeType": "image/png",
              "data":
                "iVBORw0KGgoAAAANSUgEUGAAAj4AAQACAIAAABu8MQcAAAAiXpUWHRSYXcgCHJvZmlsZSB0eXB1IGlwdGMAAAiZTYwxDgIxDAT7v0KekD
                jrtV1T0VHwgbtcIiEhgfh/QaDgmGLWw0w6X66n5fl6jNu9p+ULkapDENGzpj+KL5aFfa6KnYWgSjZjG0iSYRXTY/v8KIijI/
                rXyc236kHdAK22RvHVummEN+91ML0BQ+sioU79WmMAAAKHavRYdFhNTDpjB20uYWRvYmUueG1wAAAAA8P3hwYWNrZXQgYmVnaW49Iu+7v
                yIgaWQ9Iic1TTBNCENlaGlIenJlU3p0VGN6a2M5ZCI/"
            }
          ]
        }
      }
    ]
  }
```

Load [an example](#) or [a really big \(40MB\) example](#) or

Render JSON

Rendered Content

Candidate 1





The other prompt I tried was this one:

Provide a vegetarian recipe for butter chicken but with chickpeas not chicken and include many inline illustrations along the way

The result of that one was a [41MB JSON file](#)(!) containing 28 images - which presumably cost over a dollar since images are 3.9 cents each.

Some of the illustrations it chose for that one were somewhat unexpected:

* ½ teaspoon Kashmiri chili powder (or paprika for milder flavor)



* ½ cup heavy cream (or coconut cream for vegan option)



If you want to see that one you can click the "Load a really big example" link in [the debug tool](#), then wait for your browser to fetch and render the full 41MB JSON file.

The most interesting feature of Gemini (as with GPT-4o images) is the ability to accept images as inputs. I tried that out with [this pelican photo](#) like this:

```
cat > /tmp/request.json << EOF
{
  "contents": [{
    "parts": [
      {"text": "Modify this photo to add an inappropriate hat"},
      {
        "inline_data": {
          "mime_type": "image/jpeg",
          "data": "$(base64 -i pelican.jpg)"
        }
      }
    ]
  }],
  "generationConfig": {"responseModalities": ["TEXT", "IMAGE"]}
}
EOF

# Execute the curl command with the JSON file
curl -X POST \
  'https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash-preview-image-generation:generateContent?
key=$(llm keys get gemini) \
  -H 'Content-Type: application/json' \
  -d @/tmp/request.json \
  > /tmp/out.json
```

And now the pelican is wearing a hat:



[Introducing web search on the Anthropic API](#) (via) Anthropic's [web search](#) (presumably still [powered by Brave](#)) is now also available through their API, in the shape of a new [web search tool](#) called `web_search_20250305`.

You can specify a maximum number of uses per prompt and you can also pass a list of disallowed or allowed domains, plus hints as to the user's current location.

Search results are returned in a format that looks similar to the [Anthropic Citations API](#).

It's charged at \$10 per 1,000 searches, which is a little more expensive than what the [Brave Search API](#) charges (\$3 or \$5 or \$9 per thousand depending on how you're using them).

I couldn't find any details of additional rules surrounding storage or display of search results, which surprised me because both [Google Gemini](#) and [OpenAI](#) have these for their own API search results.

11:25 pm / [search](#), [ai](#), [generative-ai](#), [llms](#), [anthropic](#), [claude](#), [llm-tool-use](#)

May 8, 2025

[llm-gemini 0.19.1](#). Bugfix release for my [llm-gemini](#) plugin, which was recording the number of output tokens (needed to calculate the price of a response) incorrectly for the Gemini "thinking" models. Those models turn out to return `candidatesTokenCount` and `thoughtsTokenCount` as two separate values which need to be added together to get the total billed output token count. Full details in [this issue](#).

I spotted this potential bug in [this response log](#) this morning, and my concerns were confirmed when Paul Gauthier wrote about a similar fix in Aider in [Gemini 2.5 Pro Preview 03-25 benchmark cost](#), where he noted that the \$6.32 cost recorded to benchmark Gemini 2.5 Pro Preview 03-25 was incorrect. Since that model is no longer available (despite [the date-based model alias persisting](#)) Paul is not able to accurately calculate the new cost, but it's likely a lot more since the Gemini 2.5 Pro Preview 05-06 benchmark cost \$37.

I've gone through my [gemini tag](#) and attempted to update my previous posts with new calculations - this mostly involved increases in the order of 12.336 cents to 16.316 cents ([as seen here](#)).

5:49 am / [ai](#), [generative-ai](#), [llms](#), [llm](#), [gemini](#), [aider](#), [llm-pricing](#), [paul-gauthier](#)

But I've also had my own quiet concerns about what [vibe coding] means for early-career developers. So much of how I learned came from chasing bugs in broken tutorials and seeing how all the pieces connected, or didn't. There was value in that. And maybe I've been a little protective of it.

A mentor challenged that. He pointed out that debugging AI generated code is a lot like onboarding into a legacy codebase, making sense of decisions you didn't make, finding where things break, and learning to trust (or rewrite) what's already there. That's the kind of work a lot of developers end up doing anyway.

— [Ashley Willis](#), What Even Is Vibe Coding?

12:10 pm / [vibe-coding](#), [ai-assisted-programming](#), [ai](#), [generative-ai](#)

Microservices only pay off when you have real scaling bottlenecks, large teams, or independently evolving domains. Before that? You're paying the price without getting the benefit: duplicated infra, fragile local setups, and slow iteration.

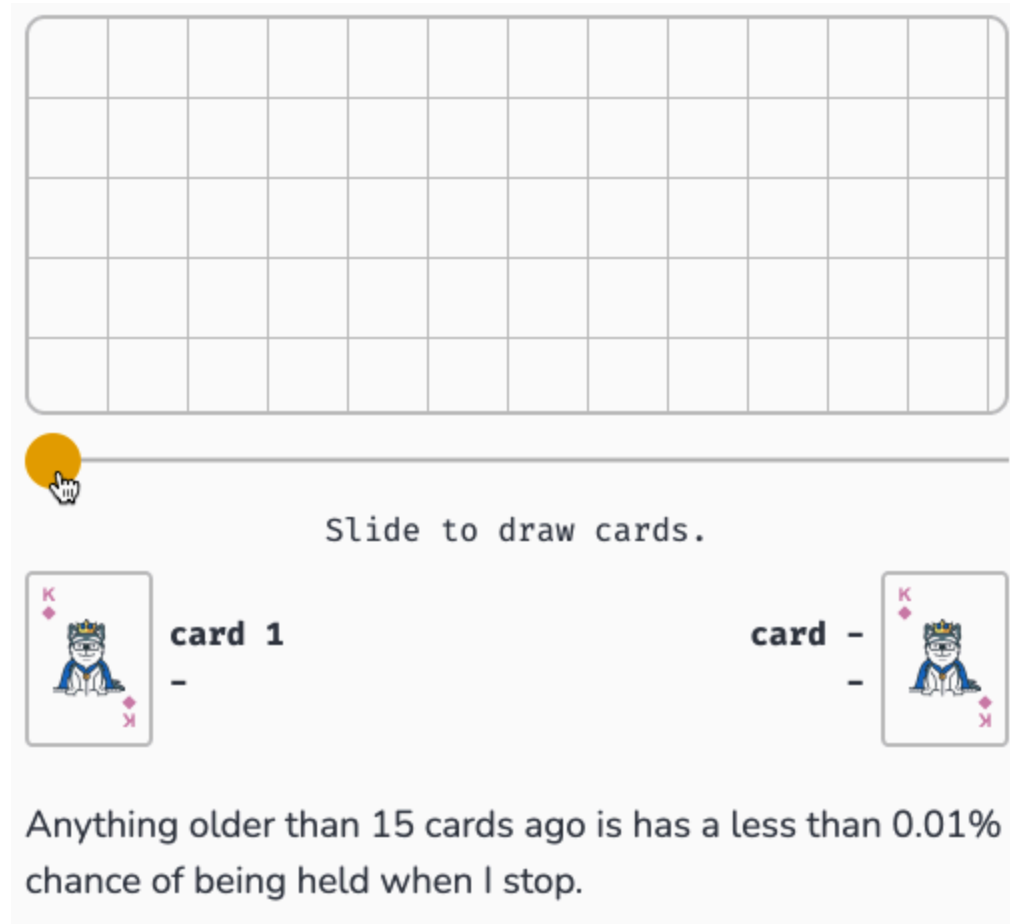
— [Oleg Pustovit](#), Microservices Are a Tax Your Startup Probably Can't Afford

7:30 pm / [software-architecture](#), [startups](#), [microservices](#)

[Reservoir Sampling](#) ([via](#)) Yet another outstanding interactive essay by Sam Rose ([previously](#)), this time explaining how reservoir sampling can be used to select a "fair" random sample when you don't know how many options there are and don't want to accumulate them before making a selection.

Reservoir sampling is one of my favourite algorithms, and I've been wanting to write about it for years now. It allows you to solve a problem that at first seems impossible, in a way that is both elegant and efficient.

I appreciate that Sam starts the article with "No math notation, I promise." Lots of delightful widgets to interact with here, all of which help build an intuitive understanding of the underlying algorithm.



Sam shows how this algorithm can be applied to the real-world problem of sampling log files when incoming logs threaten to overwhelm a log aggregator.

The dog illustration is [commissioned art](#) and the MIT-licensed code is [available on GitHub](#).

9 pm / [algorithms](#), [logging](#), [rate-limiting](#), [explorables](#), [sam-rose](#)

If Claude is asked to count words, letters, and characters, it thinks step by step before answering the person. It explicitly counts the words, letters, or characters by assigning a number to each. It only answers the person once it has performed this explicit counting step. [...]

If Claude is shown a classic puzzle, before proceeding, it quotes every constraint or premise from the person's message word for word before inside quotation marks to confirm it's not dealing with a new variant. [...]

If asked to write poetry, Claude avoids using hackneyed imagery or metaphors or predictable rhyming schemes.

— [Claude's system prompt](#), via Drew Breunig

[2025](#) » May

M	T	W	T	F	S	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	