FRED TALKS

5th April 2026

CONTENTS

[Daily article] April 5: Easter Oratorio

ENGLISH WIKIPEDIA ARTICLE OF THE DAY · 05 APR 2026 · SOURCE

The Easter Oratorio, BWV 249, is one of three oratorios composed by Johann Sebastian Bach for high holiday services of the Lutheran church in Leipzig. He wrote an autograph score (page pictured) in 1738, but had already composed the music in 1725 for two works: the congratulatory Shepherd Cantata and a church cantata for Easter. The text of the Shepherd Cantata was written by Picander, in his first documented collaboration with Bach. Picander may also have adapted the text for the Easter work, using unusually neither Biblical text nor chorales. Both works are musical dramas involving male and female characters, which, in the cantata, are from the Biblical Easter narratives. The music is structured in eleven movements, and scored for a festive Baroque instrumental ensemble of three trumpets, timpani, a variety of wind instruments, strings and continuo. Bach performed the Easter Oratorio in 1749, the year before his death. (Full article...).

Read more: https://en.wikipedia.org/wiki/Easter_Oratorio

Today's selected anniversaries:

1974:

Carrie, the first novel by American author Stephen King, was published. https://en.wikipedia.org/wiki/Carrie_%28novel%29

1983:

In China, the People's Armed Police, primarily responsible for internal security, was founded. https://en.wikipedia.org/wiki/People%27s_Armed_Police

2010:

Two bombings in Pakistan killed up to 50 people and injured around 100 others. https://en.wikipedia.org/wiki/5_April_2010_North-West_Frontier_Province_attacks

---

Wiktionary's word of the day:

unleavenedness: 1. The condition or state of being unleavened. 2. Of bread, etc.: the state of not being raised with a leaven or raising agent such as yeast. 3. (figurative) The condition or quality of being unaffected by something. 4. The condition or quality of not being controlled or moderated by some characteristic, such as courtesy, tact, etc.; immoderation, unrestraint. 5. (chiefly Christianity) The condition or quality or being unaffected by sin; purity, sinlessness. 6. About Word of the Day 7. Nominate a word 8. Leave feedback https://en.wiktionary.org/wiki/unleavenedness

---

Wikiquote quote of the day:

The glory which thou gavest me I have given them; that they may be one, even as we are one: I in them, and thou in me, that they may be made perfect in one; and that the world may know that thou hast sent me, and hast loved them, as thou hast loved me. --Jesus https://en.wikiquote.org/wiki/Jesus

---

Farzapedia, personal wikipedia of Farza, good example following my Wiki...

ANDREJ KARPATHY · 04 APR 2026 · SOURCE

Farzapedia, personal wikipedia of Farza, good example following my Wiki LLM tweet.

I really like this approach to personalization in a number of ways, compared to "status quo" of an AI that allegedly gets better the more you use it or something:

1. **Explicit**. The memory artifact is explicit and navigable (the wiki), you can see exactly what the AI does and does not know and you can inspect and manage this artifact, even if you don't do the direct text writing (the LLM does). The knowledge of you is not implicit and unknown, it's explicit and viewable.

2. **Yours**. Your data is yours, on your local computer, it's not in some particular AI provider's system without the ability to extract it. You're in control of your information.

3. **File over app**. The memory here is a simple collection of files in universal formats (images, markdown). This means the data is interoperable: you can use a very large collection of tools/CLIs or whatever you want over this information because it's just files. The agents can

apply the entire Unix toolkit over them. They can natively read and understand them. Any kind of data can be imported into files as input, and any kind of interface can be used to view them as the output. E.g. you can use Obsidian to view them or vibe code something of your own. Search "File over app" for an article on this philosophy.

4. **BYOAI**. You can use whatever AI you want to "plug into" this information - Claude, Codex, OpenCode, whatever. You can even think about taking an open source AI and finetuning it on your wiki - in principle, this AI could "know" you in its weights, not just attend over your data.

So this approach to personalization puts *you* in full control. The data is yours. In Universal formats. Explicit and inspectable. Use whatever AI you want over it, keep the AI companies on their toes! :)

Certainly this is not the simplest way to get an AI to know you - it does require you to manage file directories and so on, but agents also make it quite simple and they can help you a lot. I imagine a number of products might come out to make this all easier, but imo "agent proficiency" is a CORE SKILL of the 21st century. These are extremely powerful tools - they speak English and they do all the computer stuff for you. Try this opportunity to play with one.

Farza 🇵🇰🇺🇸@FarzaTV
[SVG OMITTED]

This is Farzapedia.

I had an LLM take 2,500 entries from my diary, Apple Notes, and some iMessage convos to create a personal Wikipedia for me.

It made 400 detailed articles for my friends, my startups, research areas, and even my favorite animes and their impact on me complete with backlinks.

But, this Wiki was not built for me! I built it for my agent!

The structure of the wiki files and how it's all backlinked is very easily crawlable by any agent + makes it a truly useful knowledge base.

I can spin up Claude Code on the wiki and starting at index.md (a catalog of all my articles) the agent does a really good job at drilling into the specific pages on my wiki it needs context on when I have a query.

For example, when trying to cook up a new landing page I may ask:

"I'm trying to design this landing page for a new idea I have. Please look into the images and films that inspired me recently and give me ideas for new copy and aesthetics".

In my diary I kept track of everything from: learnings, people, inspo, interesting links, images.

So the agent reads my wiki and pulls up my "Philosophy" articles from notes on a Studio Ghibli documentary, "Competitor" articles with YC companies whose landing pages I screenshotted, and pics of 1970s Beatles merch I saved years ago. And it delivers a great answer.

I built a similar system to this a year ago with RAG but it was ass.

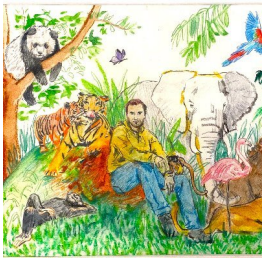
A knowledge base that lets an agent find what it needs via a file system it actually understands just works better.

The most magical thing now is as I add new things to my wiki (articles, images of inspo, meeting notes) the system will likely update 2-3 different articles where it feels that context belongs, or, just creates a new article.

It's like this super genius librarian for your brain that's always filing stuff for your perfectly and also let's you easily query the knowledge for tasks useful to you (ex. design, product, writing, etc) and it never gets tired.

I might spend next week productizing this, if that's of interest to you DM me + tell me your usecase!

[VIDEO OMITTED]



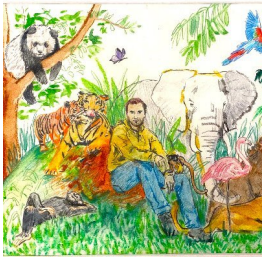
Andrej Karpathy @karpathy
[SVG OMITTED]

Wow, this tweet went very viral!

I wanted share a possibly slightly improved version of the tweet in an "idea file". The idea of the idea file is that in this era of LLM agents, there is less of a point/need of sharing the specific code/app, you just share the idea, then the other person's agent customizes & builds it for your specific needs.

So here's the idea in a gist format: https://t.co/NlAfEJjtJV

You can give this to your agent and it can build you your own LLM wiki and guide you on how to use it etc. It's intentionally kept a little bit abstract/vague because there are so many directions to take this in. And ofc, people can adjust the idea or contribute their own in the Discussion which is cool.



Andrej Karpathy @karpathy
[SVG OMITTED]

LLM Knowledge Bases

Something I'm finding very useful recently: using LLMs to build personal knowledge bases for various topics of research interest. In this way, a large fraction of my recent token throughput is going less into manipulating code, and more into manipulating knowledge (stored as markdown and images). The latest LLMs are quite good at it. So:

Data ingest:

I index source documents (articles, papers, repos, datasets, images, etc.) into a raw/ directory, then I use an LLM to incrementally "compile" a wiki, which is just a collection of .md files in a directory structure. The wiki includes summaries of all the data in raw/, backlinks, and then it categorizes data into concepts, writes articles for them, and links them all. To convert web articles into .md files I like to use the Obsidian Web Clipper extension, and then I also use a hotkey to download all the related images to local so that my LLM can easily reference them.

IDE:

I use Obsidian as the IDE "frontend" where I can view the raw data, the the compiled wiki, and the derived visualizations. Important to note that the LLM writes and maintains all of the data of the wiki, I rarely touch it directly. I've played with a few Obsidian plugins to render and view data in other ways (e.g. Marp for slides).

Q&A:

Where things get interesting is that once your wiki is big enough (e.g. mine on some recent research is ~100 articles and ~400K words), you can ask your LLM agent all kinds of complex questions against the wiki, and it will go off, research the answers, etc. I thought I had to reach for fancy RAG, but the LLM has been pretty good about auto-maintaining index files and brief summaries of all the documents and it reads all the important related data fairly easily at this ~small scale.

Output:

Instead of getting answers in text/terminal, I like to have it render markdown files for me, or slide shows (Marp format), or matplotlib images, all of which I then view again in Obsidian. You can imagine many other visual output formats depending on the query. Often, I end up "filing" the outputs back into the wiki to enhance it for further queries. So my own explorations and queries always "add up" in the knowledge base.

Linting:

I've run some LLM "health checks" over the wiki to e.g. find inconsistent data, impute missing data (with web searchers), find interesting connections for new article candidates, etc., to incrementally clean up the wiki and enhance its overall data integrity. The LLMs are quite good at suggesting further questions to ask and look into.

Extra tools:

I find myself developing additional tools to process the data, e.g. I vibe coded a small and naive search engine over the wiki, which I both use directly (in a web ui), but more often I want to hand it off to an LLM via CLI as a tool for larger queries.

Further explorations:

As the repo grows, the natural desire is to also think about synthetic data generation + finetuning to have your LLM "know" the data in its weights instead of just context windows.

TLDR: raw data from a given number of sources is collected, then compiled by an LLM into a .md wiki, then operated on by various CLIs by the LLM to do Q&A and to incrementally enhance the wiki, and all of it viewable in Obsidian. You rarely ever write or edit the wiki manually, it's the domain of the LLM. I think there is room here for an incredible new product instead of a hacky collection of scripts.

Posted Apr 2, 2026 at 8:42PM
Posted Apr 4, 2026 at 4:45PM

Posted Apr 4, 2026 at 10:55PM

Something I've been thinking about - I am bullish on...

ANDREJ KARPATHY · 04 APR 2026 · SOURCE

Something I've been thinking about - I am bullish on people (empowered by AI) increasing the visibility, legibility and accountability of their governments.

Historically, it is the governments that act to make society legible (e.g. "Seeing like a state" is the common reference), but with AI, society can dramatically improve its ability to do this in reverse. Government accountability has not been constrained by access (the various branches of government publish an enormous amount of data), it has been constrained by intelligence - the ability to process a lot of raw data, combine it with domain expertise and derive insights. As an example, the 4000-page omnibus bill is "transparent" in principle and in a legal sense, but certainly not in a practical sense for most people. There's a lot more like it: laws, spending bills, federal budgets, freedom of information act responses, lobbying disclosures... Only a few highly trained professionals (investigative journalists) could historically process this information. This bottleneck might dissolve - not only are the professionals further empowered, but a lot more people can participate.

Some examples to be precise: Detailed accounting of spending and budgets, diff tracking of legislation, individual voting trends w.r.t. stated positions or speeches, lobbying and influence (e.g. graph of lobbyist -> firm -> client -> legislator -> committee -> vote -> regulation), procurement and contracting, regulatory capture warning lights, judicial and legal patterns, campaign finance... Local governments might be even more interesting because the governed population is smaller so there is less national coverage: city council meetings, decisions around zoning, policing, schools, utilities...

Certainly, the same tools can easily cut the other way and it's worth being very mindful of that, but I lean optimistic overall that added participation, transparency and accountability will improve democratic, free societies.

(the quoted tweet is half-ish related, but inspired me to post some recent thoughts)

Harry Rushworth @Hrushworth
[SVG OMITTED]

The British Government is a complicated beast. Dozens of departments, hundreds of public bodies, more corporations than one can count...

Such is its complexity that there isn't an org chart for it.

Well, there wasn't...

Introducing ⚙️Machinery of Government⚙️

Posted Apr 4, 2026 at 12:30PM

Brain Food: Credibility is Expensive

SHANE PARRISH (FS) · 04 APR 2026 · SOURCE

FS | BRAIN FOOD

April 5th, 2026 - #675 - read online - Free Version

Welcome to Brain Food, your weekly signal in a world full of noise.

Tiny Thoughts

*

One-sided deals are one-time deals.

**

A lot of problems persist because the solution is too simple to take seriously.

***

Credibility is expensive because the bills never stop.

You pay it in conversations no one overhears, in deals where you leave money on the table, in credit you give away. You pay for it every time you say the hard thing instead of the easy one.

The strange part is that you're paying for years before anyone notices, and you can lose it all in an afternoon.

Insights

*

Jim Rohn on why we set goals:

“ The ultimate reason for setting goals is to entice you to become the person it takes to achieve them. ”

**

Former Prime Minister of the United Kingdom Margaret Thatcher on consensus:

“Consensus ... is the process of avoiding the very issues that have to be solved. ”

***

Author Ryan Holiday on accomplishment:

“You'll never feel okay by way of external accomplishments. Enough comes from the inside. It comes from stepping off the train. From seeing what you already have, what you've always had. ”

The Knowledge Project

My guest this week is Joe Liemandt, the principal of Alpha School and the founder of Trilogy Software and ESW Capital.

Liemandt dropped out of Stanford to build Trilogy, became the youngest member of the Forbes 400, then vanished from public life for 25 years. But he didn’t stop building. Through ESW Capital, he quietly became one of the most prolific acquirers of software businesses in the world.

Now he’s back with a $1 billion bet that AI can help kids learn ten times faster, and that school as we know it is broken.

At Alpha School, students spend two hours a day on AI-driven instruction and score in the top 1% on standardized tests. The rest of the day is devoted to what Liemandt calls life skills: leadership, entrepreneurship, teamwork, and real projects that kids actually care about. There are no lectures, and kids don’t move forward until they master the material.

He argues the traditional classroom was designed for a narrow slice of students and wastes everyone else's time. The fix isn't more money or better teachers; it's rebuilding from scratch around mastery, motivation, and AI. The role of a 'teacher' dramatically changes.

This conversation covers everything from sleeping on the floor at Trilogy to being mentored by Jack Welch to the thinking behind Alpha School.

I hope you enjoy this conversation as much as I did.

+ Listen and Learn: Spotify | YouTube | Apple Podcasts | Web/Transcript | X

SPONSOR

Thanks for reading. I'll see you next week.

— Shane Parrish

P.S. I can't believe this was a real stunt .

Want to share this edition with a friend? Use this link: https://fs.blog/brain-food/april-5-2026/

Upgrade Yourself

Members get access to all my reading highlights, ad-free newsletters and podcasts, hand-edited transcripts, early access, AMAs, and so much more. Join us → here .

Absurd In Production

ARMIN RONACHER · 04 APR 2026 · SOURCE

About five months ago I wrote about Absurd, a durable execution system we built for our own use at Earendil, sitting entirely on top of Postgres and Postgres alone. The pitch was simple: you don’t need a separate service, a compiler plugin, or an entire runtime to get durable workflows. You need a SQL file and a thin SDK.

Since then we’ve been running it in production, and I figured it’s worth sharing what the experience has been like. The short version: the design held up, the system has been a pleasure to work with, and other people seem to agree.

A Quick Refresher

Absurd is a durable execution system that lives entirely inside Postgres. The core is a single SQL file (absurd.sql) that defines stored procedures for task management, checkpoint storage, event handling, and claim-based scheduling. On top of that sit thin SDKs (currently TypeScript, Python and an experimental Go one) that make the system ergonomic in your language of choice.

The model is straightforward: you register tasks, decompose them into steps, and each step acts as a checkpoint. If anything fails, the task retries from the last completed step. Tasks can sleep, wait for external events, and suspend for days or weeks. All state lives in Postgres.

If you want the full introduction, the original blog post covers the fundamentals. What follows here is what we've learned since.

What Changed

The project got multiple releases over the last five months. Most of the changes are things you'd expect from a system that people actually started depending on: hardened claim handling, watchdogs that terminate broken workers, deadlock prevention, proper lease management, event race conditions, and all the edge cases that only show up when you're running real workloads.

A few things worth calling out specifically.

Decomposed steps. The original design only had `ctx.step()`, where you pass in a function and get back its checkpointed result. That works well for many cases but not all. Sometimes you need to know whether a step already ran before deciding what to do next. So we added `beginStep()` / `completeStep()`, which give you a handle you can inspect before committing the result. This turned out to be very useful for modeling intentional failures and conditional logic. This in particular is necessary when working with "before call" and "after call" type hook APIs.

Task results. You can now spawn a task, go do other things, and later come back to fetch or await its result. This sounds obvious in hindsight, but the original system was purely fire-and-forget. Having proper result inspection made it possible to use Absurd for things like spawning child tasks from within a parent workflow and waiting for them to finish. This is particularly useful for debugging with agents too.

absurdctl. We built this out as a proper CLI tool. You can initialize schemas, run migrations, create queues, spawn tasks, emit events, retry failures from the command line. It's installable via `uvx` or as a standalone binary. This has been invaluable for debugging production issues. When something is stuck, being able to just `absurdctl dump-task --task-id=` and see exactly where it stopped is a very different experience from digging through logs.

Habitat. A small Go application that serves up a web dashboard for monitoring tasks, runs, checkpoints, and events. It connects directly to Postgres and gives you a live view of what's happening. It's simple, but it's the kind of thing that makes the system more enjoyable for humans.

Agent integration. Since Absurd was originally built for agent workloads, we added a bundled skill that coding agents can discover and use to debug workflow state via `absurdctl`. There's also a documented pattern for making pi agent turns durable by logging each message as a checkpoint.

What Held Up

The thing I'm most pleased about is that the core design didn't need to change all that much. The fundamental model of tasks, steps, checkpoints, events, and suspending is still exactly what it was initially. We added features around it, but nothing forced us to rethink the basic abstractions.

Putting the complexity in SQL and keeping the SDKs thin turned out to be a genuinely good call. The TypeScript SDK is about 1,400 lines. The Python SDK is about 1,900 but most of this comes from the complexity of supporting colored functions. Compare that to Temporal's Python SDK at around 170,000 lines. It means the SDKs are easy to understand, easy to debug, and easy to port. When something goes wrong, you can read the entire SDK in an afternoon and understand what it does.

The checkpoint-based replay model also aged well. Unlike systems that require deterministic replay of your entire workflow function, Absurd just loads the cached step results and skips over completed work. That means your code doesn't need to be deterministic outside of steps. You can call `Math.random()` or `datetime.now()` in between steps and things still work, because only the step boundaries matter. In practice, this makes it much easier to reason about what's safe and what isn't.

Pull-based scheduling was the right choice too. Workers pull tasks from Postgres as they have capacity. There's no coordinator, no push mechanism, no HTTP callbacks. That makes it trivially self-hostable and means you don't have to think about load management at the infrastructure level.

What Might Not Be Optimal

I had some discussions with folks about whether the right abstraction should have been a durable promise. It's a very appealing idea, but it turns out to be much more complex to implement in practice. It's however in theory also more powerful. I did make some attempts to see what absurd would look like if it was based on durable promises but so far did not get anywhere with it. It's however an experiment that I think would be fun to try!

What We Use It For

The primary use case is still agent workflows. An agent is essentially a loop that calls an LLM, processes tool results, and repeats until it decides it's done. Each iteration becomes a step, and each step's result is checkpointed. If the process dies on iteration 7, it restarts and replays iterations 1 through 6 from the store, then continues from 7.

But we've found it useful for a lot of other things too. All our crons just dispatch distributed workflows with a pre-generated deduplication key from the invocation. We can have two cron processes running and they will only trigger one absurd task invocation. We also use it for background processing that needs to survive deploys. Basically anything where you'd otherwise build your own retry-and-resume logic on top of a queue.

What's Still Missing

Absurd is deliberately minimal, but there are things I'd like to see.

There's no built-in scheduler. If you want cron-like behavior, you run your own scheduler loop and use idempotency keys to deduplicate. That works, and we have a documented pattern for it, but it would be nice to have something more integrated.

There's no push model. Everything is pull. If you need an HTTP endpoint to receive webhooks and wake up tasks, you build that yourself. I think that's the right default as push systems are harder to operate and easier to overwhelm but there are cases where it would be convenient. In particular there are quite a few agentic systems where it would be super nice to have webhooks

natively integrated (wake on incoming POST request). I definitely don't want to have this in the core, but that sounds like the kind of problem that could be a nice adjacent library that builds on top of absurd.

The biggest omission is that it does not support partitioning yet. That's unfortunate because it makes cleaning up data more expensive than it has to be. In theory supporting partitions would be pretty simple. You could have weekly partitions and then detach and delete them when they expire. The only thing that really stands in the way of that is that Postgres does not have a convenient way of actually doing that.

The hard part is not partitioning itself, it's partition lifecycle management under real workloads. If a worker inserts a row whose `expires_at` lands in a month without a partition, the insert fails and the workflow crashes. So you need a separate maintenance loop that always creates future partitions far enough ahead for sleeps/retries, and does that for every queue.

On the delete side, the safe approach is `DETACH PARTITION CONCURRENTLY`, but getting that to run from `pg_cron` doesn't work because it cannot be run within a transaction, but `pg_cron` runs everything in one.

I don't think it's an unsolvable problem, but it's one I have not found a good solution for and I would love to get input on.

Does Open Source Still Matter?

This brings me a bit to a meta point on the whole thing which is what the point of Open Source libraries in the age of agentic engineering is. Durable Execution is now something that plenty of startups sell you. On the other hand it's also something that an agent would build you and people might not even look for solutions any more. It's kind of … weird?

I don't think a durable execution library can support a company, I really don't. On the other hand I think it's just complex enough of a problem that it could be a good Open Source project void of commercial interests. You do need a bit of an ecosystem around it, particularly for UI and good DX for debugging, and that's hard to get from a throwaway implementation.

I don't think we have squared this yet, but it's already much better to use than a few months ago.

If you're using Absurd, thinking about it, or building adjacent ideas, I'd love your feedback. Bug reports, rough edges, design critiques, and contributions are all very welcome—this project has gotten better every time someone poked at it from a different angle.

The Axios supply chain attack used individually targeted social engineering

SIMON WILLISON'S WEBLOG: ENTRIES · 03 APR 2026 · SOURCE

The Axios supply chain attack used individually targeted social engineering

The Axios team have published a full postmortem on the supply chain attack which resulted in a malware dependency going out in a release the other day, and it involved a sophisticated social engineering campaign targeting one of their maintainers directly. Here's Jason Saayman'a description of how that worked:

> so the attack vector mimics what google has documented here: https://cloud.google.com/blog/topics/threat-intelligence/unc1069-targets-cryptocurrency-ai-social-engineering
>
> they tailored this process specifically to me by doing the following:
>
> - they reached out masquerading as the founder of a company they had cloned the companys founders likeness as well as the company itself.
> - they then invited me to a real slack workspace. this workspace was branded to the companies ci and named in a plausible manner. the slack was thought out very well, they had channels where they were sharing linked-in posts, the linked in posts i presume just went to the real companys account but it was super convincing etc. they even had what i presume were fake profiles of the team of the company but also number of other oss maintainers.
> - they scheduled a meeting with me to connect. the meeting was on ms teams. the meeting had what seemed to be a group of people that were involved.
> - the meeting said something on my system was out of date. i installed the missing item as i presumed it was something to do with teams, and this was the RAT.
> - everything was extremely well co-ordinated looked legit and was done in a professional manner.

A RAT is a Remote Access Trojan—this was the software which stole the developer's credentials which could then be used to publish the malicious package.

That's a *very effective* scam. I join a lot of meetings where I find myself needing to install Webex or Microsoft Teams or similar at the last moment and the time constraint means I always click "yes" to things as quickly as possible to make sure I don't join late.

Every maintainer of open source software used by enough people to be worth taking in this way needs to be familiar with this attack strategy.