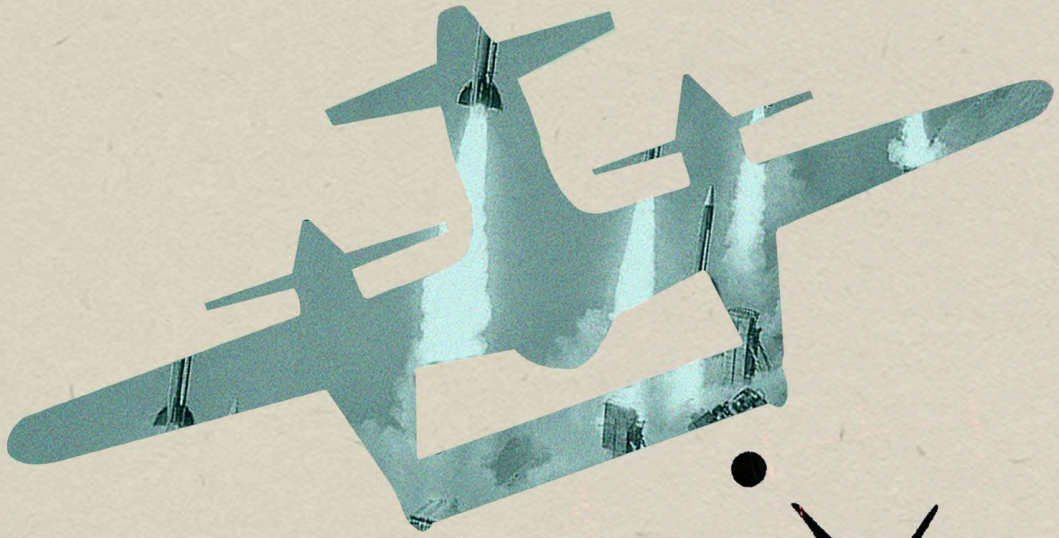
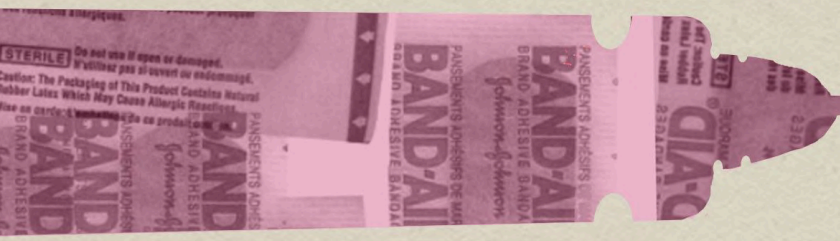


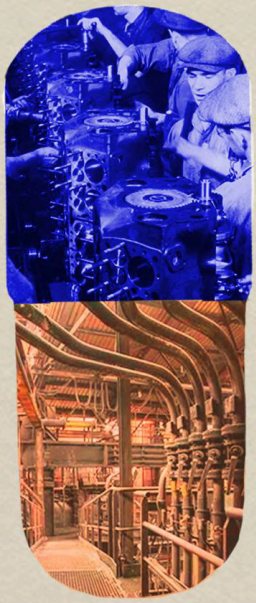
HEMANT TANEJA

QUARTERLY REVIEW

Q1: 2026

GENERAL CATALYST





I've been watching my peers process this quarter live, and while I've never really been a tweeter, I think the speed of the cycle demands slower, less reactive, more honest windows of reflection. The formats we have for publishing publicly (X, LinkedIn, the conference fireside) reward certainty and controversy. None of that helps one properly make sense of the world anymore.

A lot of large things happened simultaneously this quarter, none of them finished. The volume of happenings, the heat of the discourse, and the open question mark of what the very near future will look like can leave one feeling a bit confused.

But, I keep coming back to the simple conclusion that we can't lose the pillars of meaning that create our purpose. I think, if we can view the headlines as a complete arc, and keep a firm grasp on our purpose, the future starts to look optimistic again. The advances in technology feel like a historic, incredible feat of humanity, and the geopolitical reality feels like a recalibration toward a more resilient world order.

My thinking here draws from what I've seen and the conversations I've been part of over the last three months. I am attempting to write honestly about what this quarter actually felt like, and why GC has moved the way it has.

The world is crazy, but don't lose your head. Don't lose your soul.

AI, because I can’t start anywhere else

I have read pretty much everything the Collisons have published since I first met Patrick in 2008 at MIT. Patrick and John are not dramatic. So when I opened this year's letter, I noted the language they chose.

Stripe saw over 700 AI agent companies (debates about the term "agent" notwithstanding) launched on their platform in a single year. What Stripe sees, as the infrastructure running underneath a significant share of the global economy, is the founding rate itself changing with companies built by AI, for AI, going global on day one. The feedback loop (more capable models enabling more companies, enabling more data, enabling more capable models) is tightening with every quarter.

Their conclusion reads, "We write this letter at what may well turn out to be the advent of a different and hopefully much more beneficent singularity."

The word "hopefully" stands out. Patrick and John frame this as an aspiration and present possibility of the history we are already living. The word chosen as its pair, ‘beneficent’, is doing a lot of work.

Anthropic went from $1B to over $20B in annualized revenue run rate in fifteen months. The capabilities driving that growth are not approaching their ceiling. The speed is staggering, and I have stopped being surprised by how often I am surprised. The direction, however, is still an open question. If this goes well, what does "well" actually mean?

What the beneficent version of the singularity looks like: diseases that have been death sentences for millennia, solved. Poverty that has been a structural condition of human civilization, reduced. Energy abundance, within reach. The tedious and repetitive parts of knowledge work, automated and returning human attention to the creative, the interpersonal, the meaningful.

This means that the constraints for company-building have changed. The previous era was about capability: can we write the code, scale, etc.? The new constraint is trust: can we embed it deeply enough in the regulated, high-stakes environments of healthcare, education, defense, and financial systems, where the actual transformation of human life will happen? The companies that will matter in thirty years are not necessarily the most technically capable ones.

The anxiety on both sides

The acceleration is showing up in the texture of daily life. With Cowork's release, the excitement and anxiety around AI are both acute.

The anxiety has been talked about, but I still think we underplay just how limiting this anxiety could be for AI diffusion. For a few founders I've talked to, the excitement is also anxiety. It's so very Silicon Valley in the best way. I keep hearing from founders that every time they step away from the keyboard, there's an anxiety about an agent with GPU cycles somewhere that could be doing something useful. The fear of underutilizing AI. One group fears being replaced by agents. Another feels guilty for not deploying enough of them. Both are symptoms of a transition moving faster than our intuitions about work can absorb. Cowork, which brings AI agents to non-technical workers for the first time, is actually an attempt to close that gap by making the technology earn trust through actual usefulness.

A front row seat to LLM breakthroughs

Understanding how this actually plays out requires a front-row seat. That is why we built Percepta[1]: our AI engineering and research company inside General Catalyst, with talent at the boundary of AI research and real-world deployment in the industries we care most about. This quarter, Percepta had an AI research discovery that I want to share directly. It shows that we can turn LLMs into systems that can actually execute programs instead of just predicting text. I'll let it speak for itself.

1. Percepta is a General Catalyst operating company, not a portfolio company or similar entity in which any fund or vehicle managed or advised by General Catalyst has invested.

Please go to the appendix to read the Percepta white paper.

TLDR

Language models can solve tough math problems at research grade but struggle on simple computational tasks that involve reasoning over many steps and long context. Even multiplying two numbers or solving small Sudokus is nearly impossible unless they rely on external tools.

But what does it take for an LLM itself to be as reliable and efficient as a computer?

We answer this by literally building a computer inside a transformer. We turn arbitrary C code into tokens that the model itself can execute reliably for millions of steps in seconds.

Motivation: LLMs cannot compute

State-of-the-art language models can solve impressively hard mathematics, with systems able to reach gold-medal standard on the International Mathematical Olympiad or even tackle open Math and Science problems.

At the same time, a stubborn gap remains: even state-of-the-art models still stumble on tasks that should be purely computational. They make mistakes even on basic addition, and they fail to solve even simple Sudoku puzzles without outside help.

In practice, we bridge this gap using two families of workarounds. **Tool use:** the model writes code; an external interpreter executes it; the model reports the result. **Agentic orchestration:** an outer loop stores intermediate state, decomposes tasks, and repeatedly calls the model on short contexts.

These approaches are extremely useful but they highlight an important limitation: LLMs do not reliably perform long, exact computations on their own, so in practice we often delegate the execution to external tools or orchestration systems.

An analogy makes the distinction clear. Humans cannot fly. Building airplanes does not change that; it only means we built a machine that flies for us.

Today's language models are in the same situation. When a task requires exact computation, we attach external systems — interpreters, code runners, agent loops — and let those systems do the computing. But the capability still lives outside the model. The model itself is fundamentally handicapped: it cannot carry out the computation it is reasoning about, so it must repeatedly hand the task off to another system.

As a result, the model can describe algorithms, reason about them, or orchestrate tools that run them, but it cannot execute the steps itself. A system that cannot compute cannot truly internalize what computation is.

So the real question is not whether a model can talk about computation, or even access it through tools. The real question is whether it can execute computation internally: reliably, efficiently, and over very long horizons. If it could, it would stop being merely a coordinator of computation and become a computer itself.

How we turned LLMs to computers

We implemented a WebAssembly interpreter inside the transformer weights. WebAssembly is a low-level instruction set designed for fast, deterministic execution — a universal target that languages such as C and C++ can compile to.

Our transformer emits a program but instead of pausing for an external tool, it executes that program itself, step by step, within the same transformer.

The key difference is that tool use is opaque: the model hands off control and receives a black-box answer. In-model execution is transparent: every intermediate step appears in the trace, and the model never leaves its own decoding loop.

The model does not call an external tool. Instead, it executes the program directly via its transformer weights, producing an execution trace token by token at more than 30,000 tokens per second on a CPU. The key technical idea is a new decoding path for execution traces that turns the model's attention lookups from linear scans into queries that take logarithmic time, enabling millions of correct execution steps inside a single transformer run.

More demos: Sudoku

Sudoku is a useful stress test for exact long-horizon computation. Learned neural approaches can achieve strong performance on easy or random Sudokus but completely stumble on the hard ones. Our fully autoregressive system achieves 100% accuracy on these benchmarks.

Our system executes a fully correct compiled Sudoku solver inside the transformer itself. There is no learned heuristic standing in for the algorithm and no gap between "the model suggested a solution" and "an external system verified it." The transformer executes the solver step by step.

The guarantee is universal rather than benchmark-specific: if the compiled solver is correct, the transformer's execution is correct as well. In practice, this lets the model solve even famously hard instances such as Arto Inkala's Sudoku, reaching the correct solution in under 3 minutes.

So what is next?

We showed that a transformer can become a computer, and that opens a new interface between software and neural networks. Once programs can run efficiently inside the model's own inference loop, a much larger design space opens up.

In our current prototype the model learns an interpreter whose behavior is encoded in its weights. But the compilation machinery we built for generating those weights can go further. In principle, arbitrary programs can be compiled directly into the transformer weights, bypassing the need to represent them as token sequences at all. That would make weights themselves a deployment target for software. Instead of merely learning software-like behavior, models could literally contain compiled program logic as part of their internal circuitry.

Finally, if software becomes part of the neural architecture, then AI systems need a way to grow over time much like software libraries do today. Modern software ecosystems evolve by accumulating modules, abstractions, and reusable components. A similar process may eventually occur inside AI systems, where new computational abilities are added incrementally to the model's internal execution engine.

Closing thoughts

Our work shows that transformers can execute programs efficiently inside their own inference loop, not as an external tool, but as part of the model itself. This opens a path toward AI systems that integrate learned representations with compiled algorithms inside a single computational substrate. In that world, software itself becomes part of the model.

We think this matters because the hardest problems we care about — sequential decision-making under uncertainty in healthcare, supply chains, and financial institutions — will require systems that can both reason flexibly and compute reliably.

All of this acceleration has us thinking about investing differently

The Percepta team is lean, agile, only fifteen months old. In January, I sat down with them to work through the state of coding models. One of the most technically sophisticated AI engineering teams I know was rethinking, from scratch, how software gets built and deployed. I left with the question, if software is now abundant, what actually becomes scarce? The public markets soon started asking the same.

This is not a surprise to us. In my 2018 book Unscaled, I argued that mass production was giving way to mass personalization, and that AI drives economies of "unscale" over economies of scale. Small, focused companies can now rent what incumbents spent decades building and serve narrower markets better. The coding model breakthroughs of the past eighteen months are the most dramatic acceleration of that thesis yet. Every company can now access bespoke software. AI becomes the world's new cognition capability, this will also accelerate the transformation of the workforce, with humans and AI agents increasingly working in tandem, and the boundary between the two shifting every quarter.

The Technological

A high volume of software companies were built in a hype cycle. Many of these products never solved a critical job to be done, or their job to be done has since been obsolesced by AI. Atlassian faces pressure as AI collapses the project management overhead its products were built around. Zendesk faces an existential challenge as AI handles the customer service interactions that justified its seat-based model. More broadly, SaaS customers are building their own solutions thanks to these model innovations, and AI-native competitors are providing cheaper, more customized alternatives. One of our portfolio companies has already eliminated five of their SaaS providers by rebuilding the functionality in-house, saving roughly $8M annually.

Point solutions are being absorbed in real time. The per-seat model, which assumed human headcount would grow indefinitely, is structurally impaired.

There is another side of software, perhaps the larger group, that will continue to have a healthy business. These are businesses deeply embedded in regulated workflows, sitting on proprietary operational data, where AI makes delivery more powerful, rather than less relevant. The healthcare system with ten million patient cases. The financial institution with default behavior data across a full credit cycle. Stripe's $1.9 trillion in 2025 payment volume is infrastructure built on an operational moat.

The Financial

The consequence is financial. Software as a store of value is no longer interesting when the return model depends on a terminal buyer paying a multiple of what you paid. That assumption is broken and will not correct itself. Software commoditized earlier than LBOs were modeled for. Paying 20–50× FCF implies a 20- to 50-year duration in a world where those cashflows would be steady, when in reality the competitive landscape shifts every quarter.

Take Anthropic, one of the fastest scaling businesses in history, whose implied revenue multiple has already compressed significantly as real revenue scaled. Legacy SaaS businesses with real but stagnant cash flows still carry multiples that haven't absorbed that lesson.

Let's say we bought a high-quality B2B software business a few years ago[2]: sticky enterprise contracts, 90% gross margins, 25% annual growth. The entry price was 25× EBITDA (~40-50x FCF), with a portion financed through private credit and underwritten to a five-year hold. EBITDA grew 50%. The debt came down. And yet the equity still doesn't work as software becomes commoditized.

EBITDA growth added $625M in enterprise value. Multiple compression from 25× to 12× destroyed $975M[3]. The deal returned 0.68× MOIC — roughly a third of LP capital lost on a deal where the underlying business succeeded. Annualized over five years, that is a deeply negative IRR of approximately -7%. The business performed. The model failed anyway. The return was so dependent on the exit multiple that no amount of operational improvement could compensate for a category repriced underneath it.

The exit environment offers little relief. Strategic acquirers are navigating their own AI transitions and have little appetite for large software integrations at premium prices. The public markets are effectively closed for most PE-backed companies: passive indexing has concentrated flows into mega-cap indices, and a sub-$10B company going public today faces minimal analyst coverage, no passive inflows, and chronic multiple compression.

2. *This hypothetical scenario is presented here solely for illustrative purposes and does not refer to any actual investment by any of GC's funds or vehicles.*
3. *The 25× entry multiple reflects the median software buyout transaction multiple for the 2019-2021 vintage, a period in which near zero interest rates, abundant private credit, and sustained SaaS growth drove aggressive underwriting across the market. The 12× exit multiple reflects current conditions: public market software multiples hover around 12.7× EBITDA as of 2025-2026. A 25× EBITDA entry on a typical SaaS business implies 35-50× FCF, underwriting decades of uninterrupted cash generation for an asset being repriced in real time.*

When the loans mature, the GP faces a choice with no good options: inject fresh capital into a business whose category is being repriced by AI, or hand over the keys to the lenders.

Software buyouts moving forward

The path forward for software buyouts is not complicated. Buyouts in mature categories like industrials, healthcare, professional services have always been grounded in cash flows, not exit multiples. Software buyouts need to be structured the same way. The businesses, cash flows, and customer relationships in the current vintage are largely real. The next stack has to be built by financing operating leverage and cash flows rather than terminal value.

The math changes dramatically when you stop betting on multiple expansion. Take the same business: $50M EBITDA, 25% annual revenue growth. Instead of entering at 25× on a terminal value thesis, a disciplined buyer enters at 10× — $500M enterprise value, modest leverage of 3× EBITDA, $350M equity check. Over five years, EBITDA grows to $75M through genuine AI-driven operational improvement. The exit clears at 14×, a modest re-rating for a business that has demonstrably transformed. Enterprise value: $1.05B. Equity recovered: $950M. That is a 2.7× MOIC and an IRR of approximately 22%.

The value creation playbook changes too. PE has historically created value around the product — cost cuts, sales capacity — while treating the technology as a fixed asset to be realized at exit. The next era requires tech transformation to be the investment thesis.

Talent will be the binding constraint. The best AI engineers are not looking for a head-of-AI role at a PE-backed enterprise still debating whether transformation is necessary. They go to the bleeding edge, where the mission is credible, and the mandate is real. Firms that signal ambivalence about change will not be able to hire for it, and without the talent, they cannot execute. That cycle is vicious in one direction and virtuous in the other.

The real and transformative promise of AI is not efficiency, but growth. While there is a hard floor on how much you can cut from any business, there is no ceiling on what intelligence-driven growth can produce: new revenue streams, new categories of output, capabilities that simply did not exist before. The shift from "AI is reducing our costs" to "AI is driving our top line" is where the next era of private markets returns will be made.

Which brings me back to the question I started with: if software is abundant, what is actually scarce?

Define durable

On January 1, Warren Buffett retired after sixty years at Berkshire Hathaway. His parting claim: Berkshire has the best odds of lasting a century. This is the hardest moment in recent history to say that. I should disclose my bias: I am a deep admirer of Berkshire, and GC's Chairman, Ken Chenault, sits on its board. I can still say honestly that, by most tactical assessments, Berkshire is quite vulnerable to AI displacement. And I still think Buffett is right. He does not have the technology advantage, but he has trust. This is where we will see software companies remain resilient. Institutional relationships that compound over decades. Operational knowledge that deepens with use rather than degrades with competition. The kind of moat that does not get repriced by an AI model.

AI is deflationary for businesses whose moat was built on artificial scarcity. It is profoundly inflationary for businesses whose moats are institutional trust, deep customer relationships, and operational knowledge that compounds with increasing cognition. That is the answer to the question. That is what becomes scarce.

How we’ve positioned ourselves

We have long held that we are in a 30-year technology supercycle that began in 2007. The convergence of social, mobile, and cloud trends led to a digital representation of society that generated the data that would eventually train LLMs and bring AI to life. We continue on the path of technology-led abundance, each wave faster than the last.

It's easy in this environment to feel like the right move is to concentrate into winners, to pile into the obvious, the already-proven. This quarter, we had the privilege of participating in Anthropic's Series G, one of the largest venture rounds in history, because we believe it is one of the most important technology companies ever built. That investment has been one of our best – one of any investor’s best. We first invested in March 2025 at a $61.5B valuation. The March 2026 Series G valued Anthropic at $380B.

We have equal conviction at the seed stage and announced investments in 20 early-stage and seed companies this quarter. The full arc of venture exists and has value. The world needs a diversity of players: founders building outside of pure-play intelligence, finding new paths to intelligence outside of language and text, building the infrastructure and applications that will make this acceleration real for industries that have never seen it before.

And then there is the question of what comes after the full arc. This quarter, our Customer Value Fund (CVF) reached a milestone that, to the best of our knowledge, the venture asset class has never reached: an investment-grade credit rating. BBB or higher on roughly two-thirds of our capital, with ratings expected to grow as we scale. Apollo Global Management manages nearly one trillion dollars in AUM, roughly 80% of it in investment-grade credit strategies[4]. That is nearly $800 billion in capital at a single asset manager that venture has been unable to touch, because venture investments carry no defined duration or structure. CVF, which launched in 2021, can now bring this capital to bear for our companies. Achieving this at all is remarkable. Doing so in less than five years is groundbreaking.

What the rating represents is an entrepreneurial act as much as a financial one.

4. Apollo Global Management, Annual Report (Form 10-K), fiscal year ending December 31, 2025

We spotted a gap: venture has typically funded itself through equity commitments into limited partnerships and could never reach the investment-grade world. Thus, we built a new solution from scratch: an asset class that pays back on a predictable schedule, uniform across geographies and business types. We solved a problem for our companies and ended up creating a category.

I'm telling this story now because this is the environment we have been innovating toward. CVF gives us the ability to drive returns in software on a cash flow basis, without the terminal value dependency that has impaired so much of the current vintage. Creation, our capability for transforming and rolling up established assets, pairs great technical founders with incumbent distribution and startup-speed product reinvention, accumulating market share rather than betting on a single asset's terminal value. And Percepta, as you have already read, is building the AI orchestration layer that helps enterprises actually diffuse intelligence into their operations.

Our direct exposure to pure SaaS is approximately $4 billion out of $43 billion in AUM[5], much of which sits across strong pre-2022 companies in fintech, healthcare, and defense that have embraced AI and are scaling faster than ever. And the opportunity set for new power law companies to emerge is greater than it has ever been.

5. *As of 4/12/2026*

Future of Finance with Ramp and Stripe

In March, I sat with Patrick Collison and Eric Glyman at our first Future of Finance retreat, which we co-hosted with Stripe and Ramp, and convened 40 top CFOs, capital markets leaders, and fintech founders. We talked about how the best operating companies now treat their balance sheet as a weapon. Founders themselves are growing wary of unsustainable financing. It puts an additional premium on durable instruments.

I also heard ideas from attendees that pulled on my imagination of what the next chapter of finance looks like. Money might soon think. We could see programmable, intelligent money that carries context about its history, triggers real-time auditing, and ‘opines’ on the decisions of how it is used. I have more to think about here, but 5+ years ago, when we were first thinking about CVF, I had a similar feeling.

Durable value, to us, requires increasing resilience.

AI-enabled resilience requires a world stable enough to absorb the transformation: stable institutions, coherent policy, and governments willing to engage as genuine partners in figuring out how this technology gets deployed. Resilience — national, institutional, sectoral — is the precondition. The most important thing we can do, as stewards of AI diffusion, is keep technology and policy in genuine conversation with each other. That is what drives much of what GC does outside the portfolio.

$5B in India

In February, I flew to New Delhi. The India AI Impact Summit brought together Prime Minister Modi and a cohort of technology and AI leaders. GC announced a $5B commitment to India over the next five years. This, we believe, is one of the largest single-country investment commitments ever.

We think India will be one of the defining AI markets of the next decade. Three things are happening at once.

The first is systems. The pattern running through India's technology history is constraint: problems so complex that standard solutions don't apply, forcing answers better than anything built under easier conditions. India gave the world generic pharmaceuticals, reimagining how healthcare abundance gets manufactured and distributed at scale. When incumbent payment rails couldn't serve a billion people, India built UPI, which now processes more transactions than Visa and Mastercard combined. When no identity infrastructure existed, India built Aadhaar, the largest biometric database on earth that sophisticated markets are still trying to construct. Zepto, CRED, PBHealth are the current expression of the same logic: solve for India first, then reshape the category.

The second is scale. India will be the third-largest economy within the decade. It’s also incredibly young. Most advanced economies have an aging population. In India, one million young people enter the workforce every month, more than the entire workforce of many European countries. That is a demographic opportunity for innovation, and a massive data set.

The third is resilience. India is managing one of the most pressured strategic positions of any major democratic economy due to its proximity to China, while actively diversifying away from its longstanding reliance on Russian defense procurement. Democratic governance and a diaspora that bridges East and West have created a US-India corridor that is becoming one of the most durable strategic relationships in the world. The trade agreement finalized this quarter is the latest expression. Underneath it, India is building its industrial base more aggressively than perhaps any country on earth right now: 75% of modernization spend going domestic, up from roughly 60% five years ago; defense exports up 34x in 11 years[6]. The infrastructure and the industrial base is there, but India still needs to build the AI-native companies to fully utilize them.

The relationship between tech and policy

In addition to meeting with Prime Minister Modi this quarter, I also met with the Irish Deputy Prime Minister, and separately with Australia's Ambassador to the US (former PM). In the past year, I've had substantive conversations with heads of state from France, the UK, Singapore, Greece, and others. National resilience, stacked across regions, creates global resilience.

6. *India Ministry of Defense, Make in India Powers Defence Growth, March 2025*

Small countries that are US allies have a disproportionate role to play in the AI era. Ireland, for one, has produced nearly ten unicorns in seven years,[7] appointed its first AI Minister, and is thinking carefully about becoming the connective tissue between US technology and European governance as it takes on the EU Council Presidency. Singapore is rethinking how its defense posture can support stability in the South China Sea, in an era when military capability is no longer purely a function of population size. And Gulf States (which I'll get to in a minute) are realizing that their speed-to-market can create a new tech hub.

Our founders need these conversations too. This quarter, we brought our 'fly in' model to the UK, having previously run it in DC and Brussels. GC Institute brought 16 founders and portfolio company leaders across AI, defense, healthcare, and cybersecurity for two days of meetings with 27 UK government officials.

GCI serves to mutually educate. We bring our founders up to speed on the language and priorities of government; we bring officials into honest contact with how fast this technology is actually moving. An example of this: GCI published a defense policy report this quarter, making the case for how procurement and deployment frameworks need to evolve.

7. *Irish Times, There's nothing mythical about Ireland's growing unicorn herd, April 2025*

This work unblocks regulation, it opens government as a customer, and it shapes the policy environment that every company in this space will eventually have to navigate. The goal is to build the relationship between policymakers and tech.

The grey: Anthropic, Mythos, the DoW, and Iran

This February, that very relationship was tested directly with Anthropic and the Department of War. The story simplified fast: morally superior AI vs. unbridled acceleration. Neither accurately captures what was actually happening or the position of either side. We are investors in Anthropic and believe deeply in their mission, technology, and vision for the highest pursuit of AGI. I also know the complicated language of deals.

The reaction, predictably, was black and white. The reality is grey.

What happened is less interesting to me than what it opens up. This is a chance to actually design what responsible public-private collaboration looks like with a technology unlike anything that has come before. What do government contracts and decision-making powers look like with such rapidly accelerating capabilities? The questions Anthropic was forced to articulate are the same questions that every domain will have to face in the next decade. Where should AI have no role, and why, and who decides?

A combination of geopolitical pressure, administration priorities, and the high-stakes nature of defense made this the first real test. But the template being worked out in this conflict will extend beyond defense. It is a governance template for the age of AI. Workforce. Online safety. Education. Healthcare. The debate is inevitable. What concerns me is how quickly the public reached for the exit in the face of complexity.

Around the same time this was unfolding, I heard about Anthropic's Mythos. My initial reaction was excitement about what it could mean for vulnerability debt, the compounding backlog of security flaws that critically important organizations have never fully caught up on, and the possibility of genuine zero-day prevention for companies of all sizes.

Anthropic could have released this to gen pop. Their restraint, and the intentionality with which they are releasing via Project Glasswing, are admirable. It's the opposite of 'move fast and break things.'

The restraint also isn't anti-acceleration. Anthropic is bringing people along and building the trust that allows a model like this to eventually reach scale.

I'm very grateful Dario has his personal moral framework staked so clearly in the ground. But individual values are not a governance system. What the industry needs is the architecture to make responsible deployment practice rather than leadership preference.

Glasswing is an attempt at that system, and I’m very glad that Nikesh Arora, who is joining GC's Board, is part of the “Glasswing 40”. Nikesh has led some of the most consequential technology organizations in the world, and I've long admired the clarity and ambition he brings to everything he does. His presence reflects the caliber of judgment being brought to bear on a model that could either do serious damage to or rigorously armor the world's critical systems.

The world is grey. Mythos is grey. We need systems that let us exist in and navigate the grey.

I cannot write about grey without acknowledging the most consequential and tragic expression of that dynamic unfolding right now with Iran. The same instinct that failed the Anthropic confrontation – the retreat from complexity, the reach for a clean narrative – is failing here too. The world is grey.

When the lives of millions, the future of a region, and the stability of global energy markets all hang on the same set of decisions, the test is whether the most capable people, under the most pressure, can keep their heads cool enough to design toward something better, rather than just react to what's in front of them. History suggests that when they do, these moments produce architectures of stability that would have been impossible without the pressure that preceded them.

What would need to be true: a deterrence framework between Israel and the Gulf Cooperation Council that creates actual peace and prosperity for the people of the region. Energy markets that stabilize so the global AI buildout can proceed. AI diffusing locally across the Middle East, built for the region rather than imported into it. The small nations of this region have an extraordinary role to play in this, sitting atop much of the world's accessible energy, with young populations and governments that can move quickly. Singapore made itself indispensable to global flows of capital by betting on institutional excellence over natural resources. The Gulf states have the same opportunity, with intelligence instead of capital.

“Waste no more time arguing what a good man should be. Be one.” - Marcus Aurelius

In late March, a jury in Los Angeles found Meta and YouTube negligent in the design of their platforms, ordering $6 million in damages to a twenty-year-old woman who alleged that near-constant use of Instagram and YouTube had caused her body dysmorphia and depression. This is tech-lash 1.0 reaching its legal conclusion. Tech-lash 2.0 is already building, and it is building around AI. The question is whether we will have learned anything from the first one.

“The Most Joyless Tech Revolution Ever.”

Tech-lash 2.0 is more about how we're talking about AI, and how we're treating the people whose lives we claim we want to improve, than it is about AI itself.

The Wall Street Journal recently described the AI era as "The Most Joyless Tech Revolution Ever." It is right. The anxiety spreading across our society about displacement, the erosion of human connection, and what intelligent machines mean for ordinary lives is fully rational. It’s a response to who Silicon Valley has become in public. It stems from viral marketing campaigns that brag about fraud, from tech-fluencers laughing at Luddites who don't want to use AI because of energy costs, or from the way we announce the obsolescence of entire professions on conference stages.

Stop laughing. Start being a steward of the very thing on which you spend your well-publicized 9-9-6.

People don't hate progress. They hate snobs, jerks, people who look down on them; why would communities want us solving their problems when they don't see us as humans worth rooting for?

I have been through almost 25 years of Silicon Valley hype cycles. I have seen it at its best. I am often afraid I am now seeing it at its worst. Our collective, sector-wide ambition is being constrained by our own behavior. Tech-lash is the consequence of a lack of kindness and caring too little about the humans we are building for. Transforming healthcare, defense, energy, and education at the scale we envision requires trust from governments, regulators, legacy institutions, and communities who have no obligation to extend it to us. We are burning that trust for engagement metrics and personal brand. AI can't cure cancer if we've convinced everyone it is a cancer. Just look at nuclear as an example of how widespread fear of poorly branded technology can foreclose opportunity.

We are making the hardest thing we have ever attempted harder by treating basic human decency as a competitive disadvantage. Building companies is brutally hard. Why make it harder with a zero-sum culture where respect is seen as weakness? If you're a jerk, people won't help you, no matter how smart you are, and you need help to transform industries.

Evolution of tech lash

Four forces have brought us here, and understanding them matters.

The first is the social-media optimization of our minds and the false permission it has granted us to believe that we can decouple our social presence from our real-life presence, that what we say and what we do are separable. Investors who reason carefully in private perform extremism on X. Builders who move thoughtfully in practice declare absolute positions for the algorithm. Moderation does not get engagement. Nuance does not go viral. And so we have collectively abandoned both, racing to stake the most provocative position rather than the most considered one, telling ourselves it is “just the game.”

It is not just the game.

We are living in orthogonal black-and-white lanes of our own construction while the world we are trying to change remains irreducibly gray. It's destroying our ability to reason together, debate with nuance, and engage with challenging ideas. And it makes us look detestable.

The second is the politics contamination. When Silicon Valley entered the political arena in recent years, we had the opportunity to export our best qualities: the bias for action, the intellectual honesty, the genuine meritocracy we, at our best, actually practice. Instead, we imported the qualities of politics: tribal sorting, ideological litmus tests, zero-sum conflict performed for audiences rather than resolved for outcomes. We had the institutional credibility to elevate public discourse. We degraded it, and our own industry, instead.

The third is the genius asshole myth. The truth is that most of us do not qualify for the exemption it purports to grant. The theory holds that brilliance excuses behavior, and that the magnitude of what you are building licenses how you treat people in the building. This is among the most thoroughly researched areas in organizational behavior: one genuinely toxic person costs an organization hundreds of thousands annually in lost productivity, turnover, and diminished performance.

When someone succeeds and happens to be cruel, we assume causation when the evidence shows coincidence. They succeed despite it. They would succeed more without it.

The fourth force is the most dangerous, and one I have made my personal mission to try and counteract: contempt for legacy. In the culture Silicon Valley has created, “legacy” is a synonym for irrelevance. Legacy companies, legacy institutions, legacy people. All shorthand for things that no longer matter. This is precisely backward.

Legacy is a beautiful word and a beautiful thing to have built. The word itself tells you what matters: you endured. You were trusted long enough, by enough people, to leave a mark. The industries we most want to transform are governed, operated, and trusted by people with decades of institutional knowledge we do not have. We need their cooperation. We need them to want us to succeed. The world has no obligation to route around us.

If we treat its institutions as obstacles rather than partners, the transformation we envision will not happen. It will simply stop.

Some industries, like certain corners of finance and extractive industries, can sustain a jerk culture because they are not building the future. They are extracting value from existing systems. They don't need societal buy-in. Tech doesn't have that luxury. We need government, regulators, incumbent leaders, and the general public to adopt what we build. That requires us to be models for the future.

Adam Smith, before he wrote *The Wealth of Nations*, wrote *The Theory of Moral Sentiments*. His argument, which he considered foundational, was that self-interest produces prosperity only when constrained by sympathy, propriety, and genuine concern for others. Markets require moral foundations to function. We are building an economy organized around intelligence, and intelligence without character, without soul, is simply efficient destruction.

Legacy companies are stupid. I want to build a legacy company.

It is my greatest hope to build a legacy company. I would be honored to do something important enough to be braided into the historical retrospect of business.

I make it my responsibility to be around legacy leaders I genuinely respect. Ken Chenault — arguably one of the kindest and most ambitious leaders of our era — is General Catalyst's Chairman and my mentor. Former industry CEOs, including Ken Frazier, Kate Walsh, Steve Klasko, and Daryl Tol, help run our company. I place them alongside tech-native leaders in their thirties, like Hirsh Jain. That is a deliberate act of faith: that legacy and the future are not opposites.

We bought a hospital in Akron, Ohio. I was motivated by genuine respect for what the people at Summa Health had spent careers building, and by enough humility to concede that the professionals in the field might know things I don't. It was also deeply ambitious. No venture capital firm has ever bought a hospital. We take big swings here, but we take them in partnership with the people who already know the ground.

Kind != nice

This is not an argument for niceness. Niceness is often its own form of contempt: conflict-avoidant, performative, unwilling to engage seriously with ideas that challenge. Patrick Collison delivers difficult feedback and disagrees directly. Satya Nadella inherited a Microsoft defined by stack ranking and internal warfare, and he disbanded the toxic culture while raising expectations. Paul Graham described his purpose at YC as "trying hard to prevent assholes from getting funded." Roger Federer was the most ferociously competitive player of his generation and never once diminished an opponent. The sport grew because of how he carried himself.

I have always impressed upon my children that I want them to be ambitious and do incredible things. But above that, I want them to be genuinely kind and respectful of others. I believe they can be both. You can be kind and competitive. Kind and exacting. Kind and completely unwilling to compromise on what matters. Kind and ambitious.

Silicon Valley is also the cultural center of gravity for how the world imagines progress. That is an extraordinary privilege, and we must earn it every day. We are positioned to demonstrate what becomes possible when ambition is paired with genuine decency, when competitiveness does not require cruelty, when the gap between what we say and what we do closes rather than widens. It will be a privilege to bring about the AI transformation that architects society for the better. The only question is whether we will have been worth rooting for, whether we built it with our souls intact.

In late March, I brought together a group from GC to my farm in the mountains, above Santa Cruz, for our annual Expand Possibilities retreat. Each year I select a dozen promising talent from GC, from admin to investing, based on a 150 word application they submit. We handed out hats that say 'default optimist': the positive version of the future is achievable, and the people in the room play a role in making it happen.

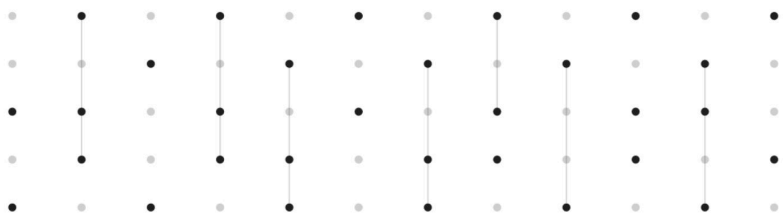
The quarter ended on a hillside above Santa Cruz. The questions followed us there. That is as it should be.

— Hemant Taneja, April 2026


DEFAULT OPTIMIST

Can LLMs Be Computers?

Executing programs inside transformers with exponentially faster inference



TL;DR

Language models can solve tough math problems at research grade but struggle on simple computational tasks that involve reasoning over many steps and long context. Even multiplying two numbers or solving small Sudokus is nearly impossible unless they rely on external tools.

> But what does it take for an LLM itself to be as reliable and efficient as a computer?

We answer this by literally building a computer inside a transformer. We turn arbitrary C code into tokens that the model itself can execute reliably for millions of steps in seconds.

Here is how it works when solving an optimization problem that proceeds in many steps, namely min-cost perfect matching[1] via the Hungarian algorithm.

USER

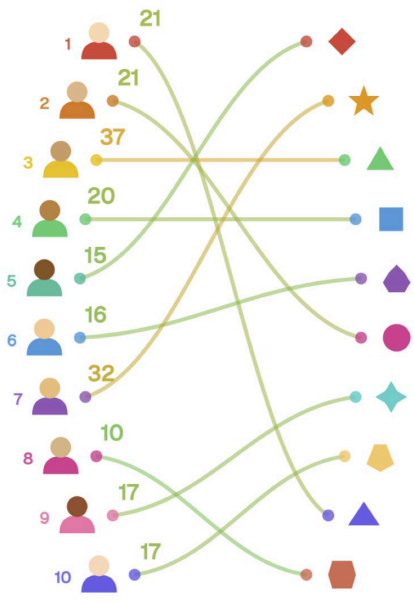
Solve the min-cost perfect matching for this 10×10 cost matrix:

```
61 58 35 86 32 39 41 27 21 42
59 77 97 99 78 21 89 72 35 63
88 85 37 57 59 97 37 29 69 94
32 82 53 20 77 96 21 70 50 61
15 44 81 10 64 36 56 78 20 69
76 35 87 69 16 55 26 37 30 66
86 32 74 94 32 14 24 12 31 70
97 63 20 64 90 21 28 49 89 10
58 52 27 76 61 35 17 91 37 66
42 79 61 26 55 98 70 17 26 86
```

ASSISTANT

```
row 5 -> col 1 (cost 15).
row 7 -> col 2 (cost 32).
row 3 -> col 3 (cost 37).
row 4 -> col 4 (cost 20).
row 6 -> col 5 (cost 16).
row 2 -> col 6 (cost 21).
row 9 -> col 7 (cost 17).
row 10 -> col 8 (cost 17).
row 1 -> col 9 (cost 21).
row 8 -> col 10 (cost 10).
```

Done.



Optimal cost: 206

The model does not call an external tool. Instead, it executes the program directly via its transformer weights, producing an execution trace token by token and streaming results at more than 30k tokens/sec on a CPU.

The key technical idea is a new decoding path for execution traces that turns the model's attention lookups from linear scans into queries that take logarithmic time, enabling millions of correct execution steps inside a single transformer run.

Motivation: LLMs cannot compute

State-of-the-art language models can solve impressively hard mathematics, with systems able to reach gold-medal standard on the International Mathematical Olympiad[2] or even tackle open Math and Science problems[3].

At the same time, a stubborn gap remains: even state-of-the-art models still stumble on tasks that should be purely computational. They make mistakes even on basic addition, and they fail to solve even simple Sudoku puzzles without outside help. Benchmarks such as Sudoku-Bench make this failure mode especially visible, reporting low unaided solve rates.[4]

In practice, we bridge this gap using two families of workarounds:

- **Tool use:** the model writes code; an external interpreter executes it; the model reports the result. Tool-integrated approaches measurably improve math reasoning.[5]
- **Agentic orchestration:** an outer loop stores intermediate state, decomposes tasks, and repeatedly calls the model on short contexts, effectively bolting a state machine onto the outside.

These approaches are extremely useful but they highlight an important limitation: LLMs do not reliably perform long, exact computations on their own, so in practice we often delegate the execution to external tools or orchestration systems.

An analogy makes the distinction clear. Humans cannot fly. Building airplanes does not change that; it only means we built a machine that flies for us.

Today's language models are in the same situation. When a task requires exact computation, we attach external systems (interpreters, code runners, agent loops) and let those systems do the computing.

But that means the capability still lives outside the model. The model itself is fundamentally handicapped: it cannot carry out the computation it is reasoning about, so it must repeatedly hand the task off to another system.

As a result, the model can describe algorithms, reason about them, or orchestrate tools that run them, but it cannot execute the steps itself. A system that cannot compute cannot truly internalize what computation is.

So the real question is not whether a model can talk about computation, or even access it through tools. The real question is whether it can execute computation internally: reliably, efficiently, and over very long horizons. If it could, it would stop being merely a coordinator of computation and become a computer itself.

How we turned LLMs to computers

On the surface, nothing is broken. The transformer architecture that powers LLMs is incredibly capable. Multiple works have shown that different variants of transformers can carry out complex computations because they can simulate Turing machines and hence any effective computer[6]. Recent work shows that they can even achieve these computational capabilities via training[7].

But theoretical universality is not the same as practical execution. A model can be expressive enough to simulate a computer and still be a terrible computer in practice. Classical universality results often rely on constructions where simple operations in a real machine correspond to long sequences of steps in the simulated model. In other words, representational capability alone does not guarantee practical execution efficiency.

> We tackle this by implementing a modern RAM computer inside the transformer rather than a purely theoretical model of computation. In our construction, each instruction maps to only a handful of tokens (at most 5).

But even that is not enough. The deeper problem is the decoding process itself.

Transformers have a structural handicap as executors: standard autoregressive decoding makes each step interact with the full accumulated history. A real computer updates a compact state with roughly constant work per instruction. A transformer instead generates one token at a time while continually interacting with a prefix that keeps growing. KV caching avoids recomputing past key/value projections, but decoding still requires attending over the cached prefix, so work per step remains coupled to sequence length.[8]

> We remove this handicap by showing that the same transformer architecture admits an efficient decoding scheme for execution-style traces. The key technical unlock is to restrict lookup heads to head dimension 2, which enables a decoding path where the dominant retrieval/update operations can be computed in log time in the sequence length (for this structured executor regime), rather than by a full prefix-sized attention sweep.

This is powerful enough to let us execute arbitrary programs inside a transformer for millions of steps.

The rest of this post walks through how this works. Feel free to skip ahead:

- What does computation mean? — How in-model execution differs from tool use: the transformer runs the program itself, step by step.
- More demos: Sudoku — Executing a compiled solver inside the transformer to solve the Arto Inkala Sudoku, regarded as the hardest Sudoku in the world.
- How can computation be encoded? — Execution as a trace that only appends, and how a transformer reconstructs state by looking back at prior steps.
- Exponentially Fast Attention — The core unlock: 2D heads turn attention into a convex-hull query, enabling log-time decoding over million-step traces.

- What is next? — Richer attention, training at scale, compiling programs into weights, and growing AI systems like software.

What does computation mean?

These days, when a model needs to compute something, it writes code. For example, to add 3 + 5, a model might output:

```
python -c "print(3+5)"
```

Generation then pauses. An external tool-use mechanism intercepts the code block, sends it to a sandboxed interpreter, and injects the result (`8`) back into the token stream. The model resumes, now knowing the answer.

This works, but the actual execution happened outside the model. The model specified the computation, then waited for an external system to carry it out.

Our transformer also emits a program, but instead of pausing for an external tool, it executes that program itself, step by step, within the same transformer.

To achieve this, we implemented a WebAssembly interpreter inside the transformer weights.[9] WebAssembly is a low-level instruction set designed for fast, deterministic execution and a universal target that languages such as C and C++ can compile to. To compute 3 + 5, the model would write:

```
{
i32.const 03 00 00 00
i32.const 05 00 00 00
i32.add   00 00 00 00
output    00 00 00 00
}
```

The model then switches to fast decoding mode and executes the program itself, step by step within the same transformer, producing an execution trace of tokens:

```
03 00 00 00  commit(+1,sts=1,bt=0)
05 00 00 00  commit(+1,sts=1,bt=0)
08 00 00 00  commit(-1,sts=1,bt=0)
out(08)
halt
```

Each line contains tokens the model generates. The stack grows, the `add` fires, the result is output, and halts, all within the model's own output stream, with no external round-trip.

The key difference is that tool use is opaque: the model hands off control and receives a black-box answer. In-model execution is transparent: every intermediate step appears in the trace, and the model never leaves its own decoding loop.

More demos: Sudoku

Sudoku is another useful stress test for exact long-horizon computation. Learned neural approaches can achieve strong performance on easy or random Sudokus but completely stumble on the hard ones.[10] A common explanation is that autoregressive models are fundamentally ill-suited for constraint-satisfaction problems because they commit to answers token-by-token and cannot revise early mistakes.[11] But our fully autoregressive system achieves 100% accuracy on these benchmarks, suggesting that the real bottleneck is not the autoregressive paradigm itself — it is that solving hard Sudokus requires very long execution traces, and standard attention makes long-context generation prohibitively expensive. This is exactly the problem our fast attention path addresses.

Our system executes a fully correct compiled Sudoku solver inside the transformer itself. There is no learned heuristic standing in for the algorithm and no gap between "the model suggested a solution" and "an external system verified it". The transformer executes the solver step by step.

The guarantee is universal rather than benchmark-specific: if the compiled solver is correct, the transformer's execution is correct as well. In practice, this lets the model solve even famously hard instances such as Arto Inkala's Sudoku, reaching the correct solution in under 3 minutes.

8	1	2	7	5	3	6	4	9
9	4	3	6	8	2	1	7	5
6	7	5	4	9	1	2	8	3
1	5	4	2	3	7	8	9	6
3	6	9	8	4	5	7	2	1
2	8	7	1	6	9	5	3	4
5	2	1	9	7	4	3	6	8
4	3	8	5	2	6	9	1	7
7	9	6	3	1	8	4	5	2

How can computation be encoded?

A useful way to picture an autoregressive transformer is as a machine that lives inside its own history. A traditional computer has editable memory that it updates as a result of operations. But in a transformer there is no such thing.

There is a fixed prompt (the input or the program). Then there is a trace that only grows (the tokens the model generates). At each step, the model can only "look back" through a small number of queries (attention heads), and must then append one more token. Through this process we need to find a way to encode a working machine.

To build intuition about this model and how the encoding works, it is helpful to think about the following.

Computation as a trace that only appends

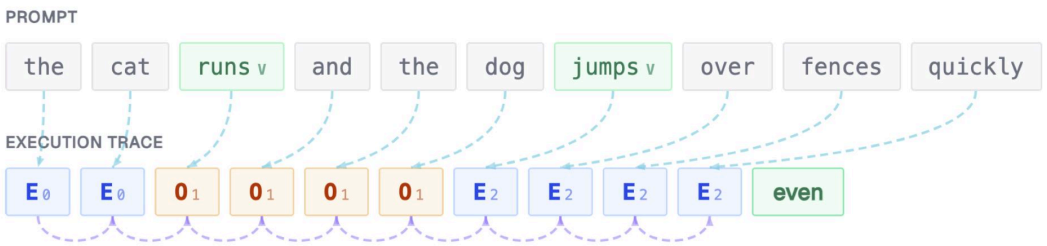
To understand how a transformer can execute a program internally, it helps to think of computation in a slightly unusual way.

Imagine a notebook where every step of a computation is written on the next line. Once written, earlier lines cannot be changed; the notebook only grows.

- The first few lines contain the input (the prompt).
- Each new line records the next step of the computation.
- At every step you are allowed to look back at earlier lines, but you cannot edit them.

This is surprisingly close to how an autoregressive transformer operates: the prompt is the input, the generated tokens form a trace that keeps growing, and each new token is produced after looking back at a small number of positions via attention.

Here is a toy example. Given a sentence, count whether the number of verbs is odd or even. Each trace token attends to exactly two positions: the corresponding input word (to check if it is a verb) and the previous trace token (to read the running parity).



Notice that each step only needs two lookbacks (one into the prompt and one into the trace), regardless of how long the sentence is. This is the key insight: many algorithms can be expressed as a trace that only appends, where each step reads a small, fixed number of earlier positions.

> Can a computation be represented as a trace that only appends, where each step only needs to look back a small number of times?

The answer is yes. In our system the model generates such a trace explicitly. The tokens it produces represent the evolving state of a virtual machine: instruction pointer, memory and stack operations, arithmetic, control flow, and outputs. The model reconstructs the current state simply by looking back to the relevant earlier steps.

> **Technical Note: What can a transformer look back at?**
>
> In a transformer, the look-back operations are more expressive than simply inspecting past tokens. Each step can also read intermediate states computed while deciding which token to produce — these correspond to the values stored in different attention layers. Think of it this way: each attention head acts like a shared one-dimensional array. When processing a token, the head first writes a single value at some index, then may read a single value at a (potentially different) index — one write followed by at most one read, in that order. Each token reads values published by earlier tokens and publishes values to inform future decisions.
>
> In addition to this read/write primitive, attention can also compute *cumulative sums*. This lets us track quantities like the instruction pointer, operand-stack depth, and call-stack depth as running sums of delta increments. Together, index lookup and summing are enough to run complex computation.

The rest of this post focuses on the efficiency challenge: even if a transformer can represent such an execution trace, standard decoding still pays a growing cost as the trace becomes longer. Our fast decoding path removes that handicap, and the 2D head restriction is the key unlock that makes the fast path possible.

The key unlock: Exponentially Fast Attention

A real computer runs long programs by updating a compact state (registers, stack, memory) with near-constant work per instruction.

A standard transformer "advances state" by generating tokens, and each next token is computed by attention over a prefix that grows forever. KV caching reuses previously computed keys/values, but it does not remove the basic scaling: at each step the query still has to interact with a cache whose size grows with the number of generated tokens. This is why decoding research often becomes an IO problem: the bottleneck becomes "make the KV cache fast enough and score against it."[12]

But no matter how fast one makes KV caching, the fundamental scaling remains: the t-th decoding step still interacts with a prefix of length t. This means work per step grows linearly with the trace length, and the total cost of generating t tokens grows quadratically. This is a well-known bottleneck of transformers and it is an active area of research how to design faster alternatives.[13]

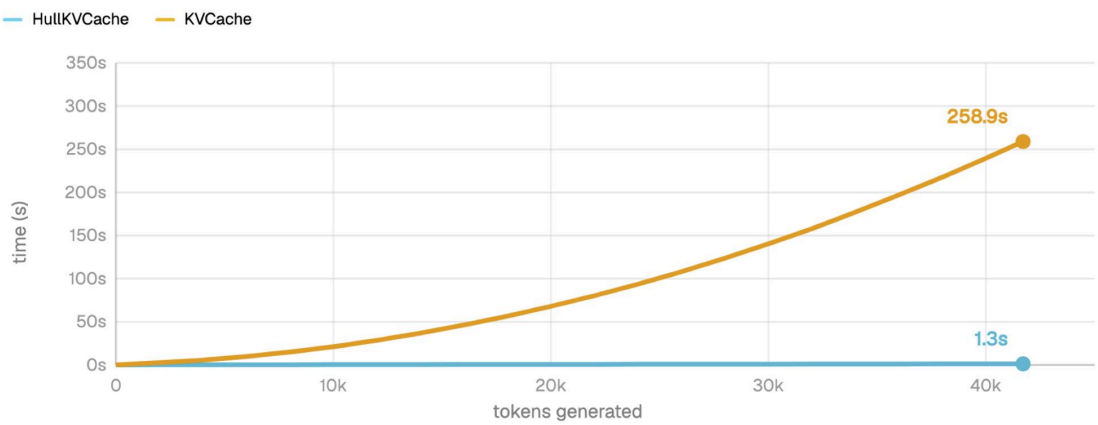
As we'll explain below, our approach addresses the quadratic blow-up and yields exponentially faster attention lookups. Instead of spending $O(t)$ time per step, our method requires $O(\log t)$ time. Here is how it looks in practice, when using our HullKVCache compared to a standard KVCache:

HullKVCache

```
25 00 00 00 commit(+1,sts=1,bt=0)
00 00 00 00 commit(+1,sts=1,bt=0)
ee fe ff ff commit(-1,sts=0,bt=1)
8d 00 00 00 commit(+0,sts=0,bt=0)
64 00 00 00 commit(+1,sts=1,bt=0)
02 00 00 00 commit(-1,sts=1,bt=0)
halt
```

KVCache

```
25 00 00 00 commit(+1,sts=1,bt=0)
00 00 00 00 commit(+1,sts=1,bt=0)
ee fe ff ff commit(-1,sts=0,bt=1)
8d 00 00 00 commit(+0,sts=0,bt=0)
64 00 00 00 commit(+1,sts=1,bt=0)
02 00 00 00 commit(-1,sts=1,bt=0)
halt
```



In the workloads we care about (long execution traces where the model repeatedly performs a small number of structured lookups per step), the difference in scaling per step compounds.

A linear scan decoder pays a cost that keeps growing as the trace grows. A hull-based decoder keeps the cost per step essentially tied to a retrieval primitive that runs in logarithmic time. Over long horizons, that changes what is feasible: computations that would be impractically slow under standard decoding become runnable for millions of steps.

This is also why the payoff shows up most clearly on "boring" deterministic spans: exact copying, state-machine stepping, or long mechanical traces. These are precisely the steps where we do not want to spend full attention budget.

The fast path: 2D attention

We obtain this speed-up by reframing the question. We are not attempting to speed up transformers in full generality, nor do we want to introduce yet another architecture.

Instead, we focus on vanilla transformers but under a tractable parameterization. We specifically target the case where the dimension of the heads is small: 2D.

This does not mean that the whole transformer is small. You can still have an arbitrarily large number of layers, arbitrarily many heads and arbitrarily large embeddings. It just means that the embeddings used across layers of a transformer are broken into smaller chunks resulting in more heads n_heads = d_model / 2.

Of course, this restriction can come at a cost for certain tasks and is not the magic answer to everything. While we are still exploring how limiting this is in practice for training, say, large models, we find that it is still flexible enough to train efficiently and can capture very complicated logic.

In fact, for Turing completeness, 2D attention is all you need! And it is still flexible enough to represent a whole RAM computer as we show in this blog. It is a crucial enabler of building a fast-path in transformers. While abstract reasoning and planning remain part of the original path, heavy computational tasks can go through the fast path.

The model itself is a completely standard PyTorch transformer, nothing exotic:

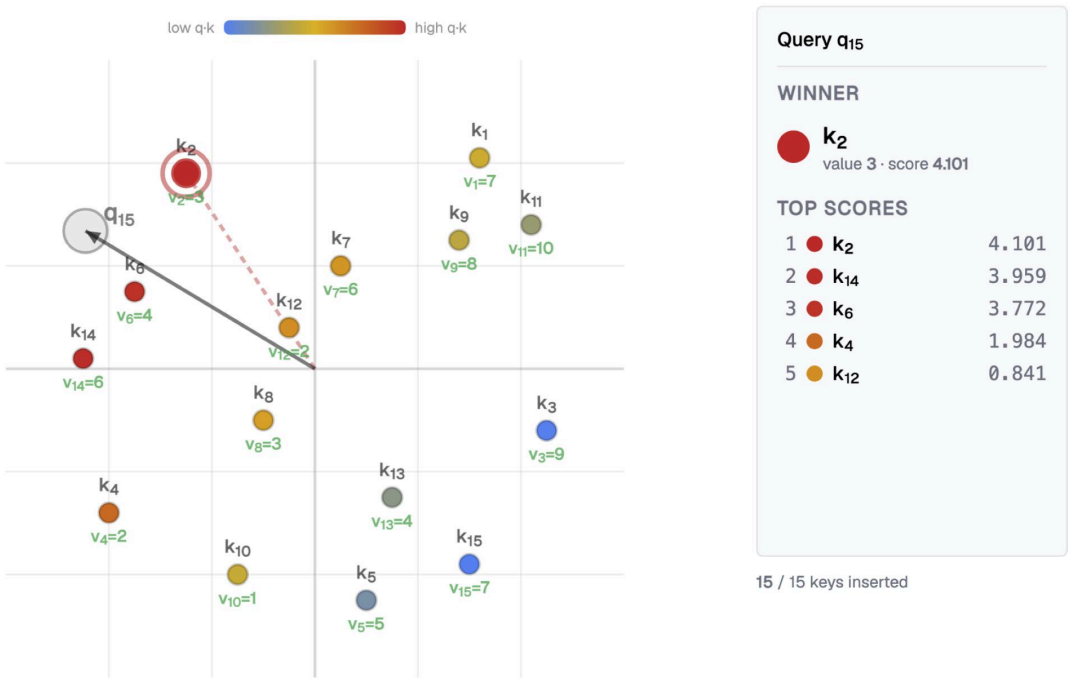
```
class VanillaTransformer(nn.Module):
    def __init__(self, vocab, d_model=36, n_heads=18, n_layers=7, d_ffn=36):
        super().__init__()
        self.tok = nn.Embedding(vocab, d_model)
        self.attn = nn.ModuleList([
            nn.MultiheadAttention(d_model, n_heads, batch_first=True, bias=False)
            for _ in range(n_layers)
        ])
        self.ff_in  = nn.ModuleList([nn.Linear(d_model, 2*d_ffn, bias=False) for _ in
        self.ff_out = nn.ModuleList([nn.Linear(d_ffn, d_model, bias=False) for _ in r
        self.head = nn.Linear(d_model, vocab, bias=False)

    def forward(self, idx):
        T = idx.shape[1]
        x = self.tok(idx) + pos_emb(T)
        causal = torch.triu(torch.ones(T, T, device=idx.device, dtype=torch.bool), di
        for attn, ff_in, ff_out in zip(self.attn, self.ff_in, self.ff_out):
            y, _ = attn(x, x, x, attn_mask=causal, need_weights=False)
            x = x + y
            gate, val = ff_in(x).chunk(2, dim=-1)
            x = x + ff_out(F.relu(gate) * val)
        return self.head(x)
```

Notice: `d_model = 36` with `n_heads = 18` gives exactly 2D per head. The architecture uses 7 layers, standard `nn.MultiheadAttention`, and a gated feed-forward network. No custom attention kernels, no sparse masks; just vanilla PyTorch. The only thing that makes it special is the weights.

Let's look at how 2D attention gets us these impressive speed-ups.

The geometric view of 2D attention



The attention mechanism works as follows:

- each past token contributes a key vector k_j and a value v_j,
- the current step forms a query vector q,
- relevance scores are computed for every key q . k_j, and
- the head returns the sum of values of all keys reweighted by the softmax of their scores.

In the special case that matters for our fast path:

- each key is 2D, k_j in R^2,
- the query q in R^2 can be thought of as a direction in the plane,
- and we want the value of the point that maximizes the dot product in that direction (hard-max attention). In case of ties, we compute the average.

Despite the maximization queries being global over the whole history, there exist efficient data-structures that can answer such 2D attention queries in logarithmic time in the number of points. That is exactly the classic "supporting point" query in computational geometry: given a direction q, find the point on the convex hull furthest in that direction. We speed up decoding by maintaining such a data-structure as tokens get generated which lets us efficiently search over the whole set of past points.

Restricting the head dimension to 2 is what enables the fast path: during inference we can replace a brute force scan ("score against every key") with a structure where the maximum can be found by looking only at a tiny subset of points in the hull.

> **Technical Note: Why do two dimensions suffice?**
>
> To implement memory and stack operations, we need each attention head to answer the query "give me the value most recently stored at index *i*." This index lookup is what requires 2D keys.
>
> Store every index *j* as the 2D key k_j = (2j, -j^2). Looking up index *i* then amounts to querying in direction q = (i, 1), since
>
> argmax_j { (2j, -j^2) . (i, 1) } = i
>
> The quadratic term -j^2 acts as a penalty that grows for any j != i, ensuring only the exact match wins the argmax.
>
> Attention can also compute cumulative sums, which we use to track quantities like the instruction pointer and stack depth. If all keys are set to the same value, attention averages all values uniformly, and multiplying by the current token position *t* recovers the actual sum. This only requires 1D (or even 0D) keys — it is index lookup that forces us to 2D.

So what is next?

We showed that a transformer can become a computer, and that opens a new interface between software and neural networks. Once programs can run efficiently inside the model's own inference loop, a much larger design space opens up.

Richer attention mechanisms

For simplicity, our construction uses hard-max attention. But that is not a fundamental restriction. While we do not yet know whether exact softmax attention can be maintained with the same efficiency, it is easy to approximate it with k-sparse softmax attention: retrieve the top-k keys and perform the softmax only over those. By storing points across nested convex hulls, this yields a decoding cost of O(k + log n).

This means the geometric fast path is not limited to our executor construction. In principle, it can accelerate any transformer with 2D heads at decoding time, replacing full linear scans with efficient geometric retrieval. The same machinery also extends naturally to 3D heads via 3D convex hulls, although higher dimensions quickly become less efficient. The key question is whether 2D already captures most of the speedup, or whether slightly larger heads unlock substantially more capability.

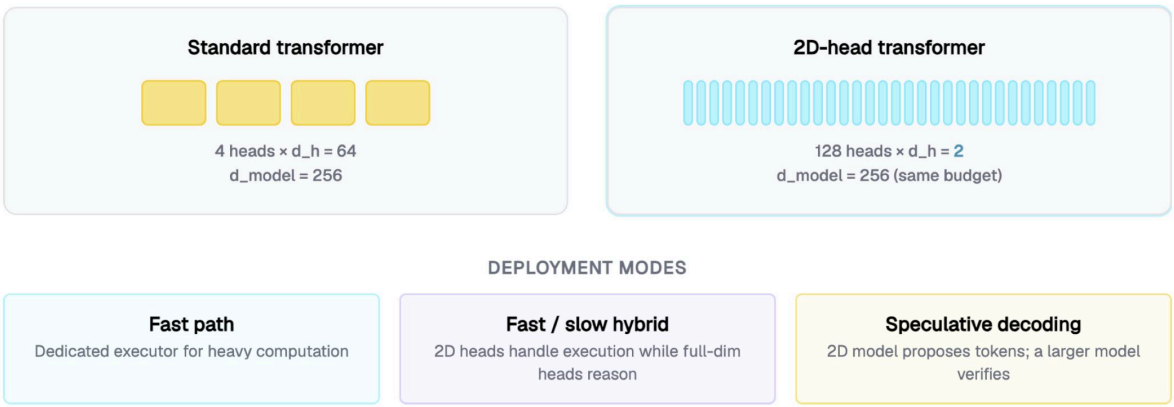
k-sparse softmax retrieves the top-k keys from **nested convex hulls** and applies softmax only over those — cost **O(k + log n)**.

The same geometric machinery extends to **3D heads** via 3D convex hulls.

Training large models with 2D heads

The 2D-head parameterization used here is not inherently small. The total parameter count can remain comparable to standard transformers because the model can simply use more heads and layers. The real question is how capable such models become when trained at scale.

That question matters even beyond pure capability. These models could be useful in several modes: as a dedicated fast path paired with a slower, more general model; as part of a fast/slow hybrid architecture inside a single system; or as a speculative execution model that proposes tokens quickly while a regular-attention model verifies and accepts them. Regardless of their eventual capability ceiling, they already suggest a powerful systems primitive for speeding up larger models.



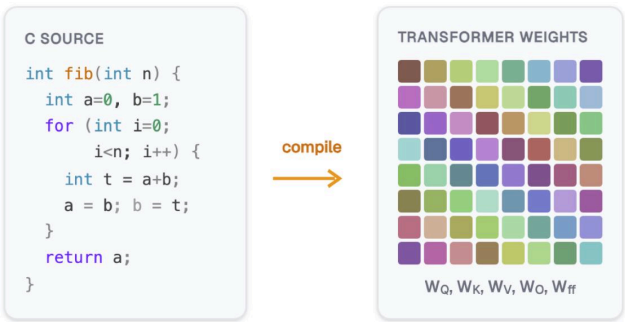
The system presented here is a standalone transformer designed as an executor. A natural next step is to combine it with a large language model and train the model to invoke this execution pathway when exact computation is required. In such a hybrid system the language model would plan and reason, while the execution component would run algorithms.

Because the execution trace is part of the forward pass, the whole process remains differentiable: we can even propagate gradients through the computation itself. That makes this fundamentally different from an external tool. It becomes a trainable computational substrate that can be integrated directly into a larger model.

Programs into weights & training beyond gradient descent

In our current prototype the model learns an interpreter whose behavior is encoded in its weights. But the compilation machinery we built for generating those weights can go further. In principle, arbitrary programs can be compiled directly into the transformer weights, bypassing the need to represent them as token sequences at all.

That would make weights themselves a deployment target for software. Instead of merely learning software-like behavior, models could literally contain compiled program logic as part of their internal circuitry.



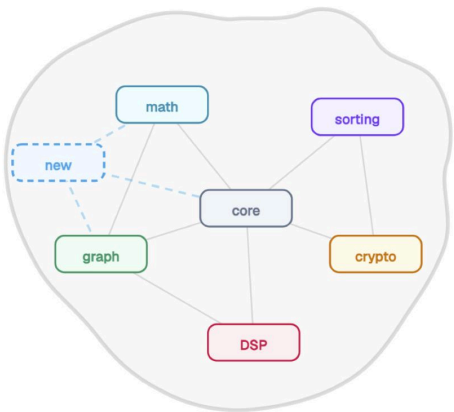
Weights become a deployment target: instead of learning software-like behavior, models **contain** compiled program logic.

If logic can be compiled into weights, then gradient descent is no longer the only way to modify a model. Weight compilation provides another route for inserting structure, algorithms, and guarantees directly into a network.

Taken seriously, this suggests a different picture of training altogether: not just optimizing weights with data, but also writing parts of the model directly. Push that idea far enough and you get systems that do not merely learn from experience, but also modify or extend their own weights, effectively rewriting parts of their internal machinery.

Growing AI systems like software

Finally, if software becomes part of the neural architecture, then AI systems need a way to grow over time much like software libraries do today. Modern software ecosystems evolve by accumulating modules, abstractions, and reusable components. A similar process may eventually occur inside AI systems, where new computational abilities are added incrementally to the model's internal execution engine.



Just as software ecosystems grow by accumulating **modules**, AI systems can add computational abilities incrementally — each one compiled into the model's internal execution engine.

New capabilities connect to the existing circuit without retraining the whole system.

Our work shows that transformers can execute programs efficiently inside their own inference loop. The broader vision is that future AI systems will not just use software; they will contain it, integrating learned representations with compiled algorithms inside a single computational substrate. In that world, software itself becomes part of the model.

Closing thoughts

We showed that transformers can execute programs efficiently inside their own inference loop, not as an external tool, but as part of the model itself. This opens a path toward AI systems that integrate learned representations with compiled algorithms inside a single computational substrate.

We think this matters because the hardest problems we care about, sequential decision-making under uncertainty in healthcare, supply chains, and financial institutions, will require systems that can both reason flexibly and compute reliably.

We are building these systems now, and we are hiring. If you want to work on problems at the intersection of language models, reinforcement learning, and real-world decision systems, join us.

References

1. Min-cost matching is a challenging optimization problem where the goal is to assign people to tasks minimizing the cost of the assignment. wikipedia.org
2. Google DeepMind, Gemini with Deep Think officially achieves gold-medal standard at IMO, deepmind.google
3. OpenAI, Early science acceleration experiments with GPT-5, cdn.openai.com
4. Sudoku-Bench Leaderboard, pub.sakana.ai
5. MuMath-Code: Combining Tool-Use Large Language Models with Multi-perspective Data Augmentation for Mathematical Reasoning, aclanthology.org
6. Attention is Turing Complete, jmlr.org
7. Can You Train a Computer?, dimitrisp.substack.com
8. FlashAttention: KV cache explained, flashattn.dev
9. WebAssembly, developer.mozilla.org
10. Teaching Transformers to Solve Combinatorial Problems through Efficient Trial & Error, arxiv.org
11. Kona vs LLMs on Sudoku, logicalintelligence.com
12. PyTorch: Flash-Decoding, pytorch.org
13. Reformer: The Efficient Transformer, arxiv.org

IMPORTANT DISCLOSURES

This letter (this "Letter") is provided by General Catalyst Group Management, LLC ("General Catalyst" or the "Adviser"), a registered investment adviser with the U.S. Securities and Exchange Commission ("SEC"). Registration with the SEC does not imply a certain level of skill or training, nor does it constitute an endorsement of the Adviser by the SEC.

NOT AN OFFER OR SOLICITATION

This Letter does not constitute an offer to sell, or a solicitation of an offer to buy, any security, investment product, or interest in any fund managed or sponsored by General Catalyst (each, a "Fund" and collectively, the "Funds"). Any such offer or solicitation will be made only by means of a confidential private placement memorandum, limited partnership agreement, subscription agreement, or other definitive offering documents (collectively, "Offering Documents") delivered to qualified prospective investors and in accordance with applicable law. This Letter is not intended to be, and should not be construed as, investment advice tailored to the needs of any specific investor.

FORM ADV AND ADDITIONAL INFORMATION

General Catalyst's Form ADV Part 2A (the "Brochure") contains additional information about the Adviser's business practices, fee arrangements, potential conflicts of interest, and disciplinary history, if any. A copy of General Catalyst's Brochure is available upon request or may be obtained through the SEC's Investment Adviser Public Disclosure website at https://adviserinfo.sec.gov.

NO WARRANTY; LIMITATION

The information contained in this Letter is provided "as is" and without warranty of any kind, either express or implied. General Catalyst expressly disclaims all warranties, including warranties of merchantability and fitness for a particular purpose. General Catalyst shall not be liable for any direct, indirect, incidental, consequential, or other damages arising out of or in connection with the use of or reliance on the information contained herein.

General Catalyst reserves the right to amend, supplement, or replace any information contained in this Letter at any time and without prior notice.