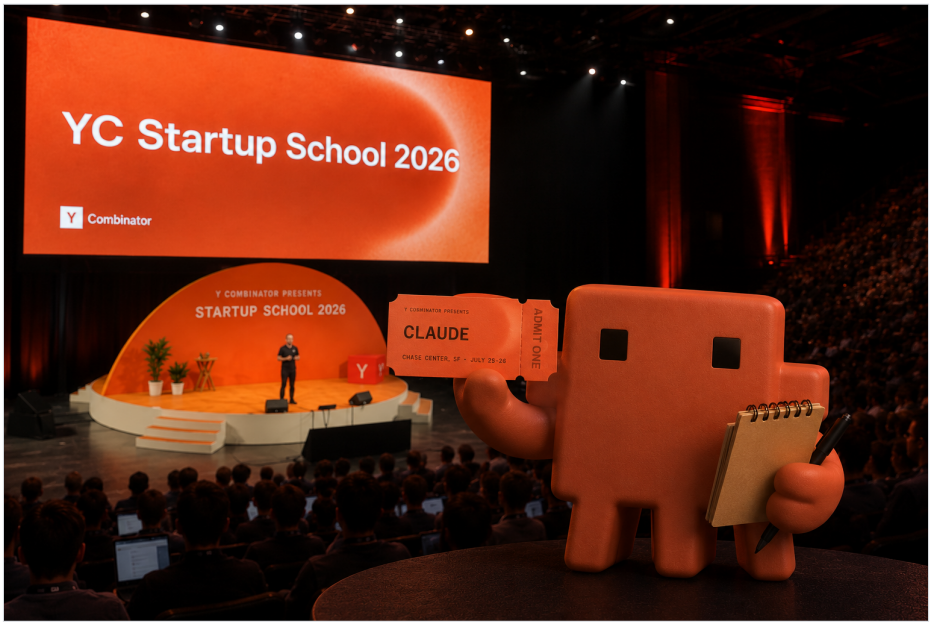
Y COMBINATOR

Startup School 2026

Notes on the talks

Jensen Huang · Sam Altman · Alexandr Wang
Blake Scholl · Boris Cherny



Claude attends YC Startup School

One chapter per speaker, in playlist order. Each chapter opens with the video's thumbnail and a link to the talk. Written from the transcripts, and kept to what each speaker actually argued, in plain language, with no buzzwords.

Source playlist: youtube.com/playlist?list=PLEb7ftOB0yf0

Compiled July 30, 2026

Contents

How to read these notes

The five talks were given at the same event and turn out to argue with each other very little. Read end to end, four claims recur across speakers who had no reason to coordinate:

1. **Conviction that nobody shares is the asset, not the risk.** Wang on unglamorous data work, Altman on being called irresponsible for a decade, Scholl on an idea everyone found laughable, Huang on a thesis about accelerating computation. All four describe years of being dismissed as the thing that gave them room to work.
2. **The constraint moved from capability to ambition.** Every speaker says some version of this. Wang calls the scarce resource vision. Altman says do three months of work in three months of work. Huang says the scale of a task has stopped being a constraint, so only the choice of problem is left.
3. **Systems thinking survives; specifics do not.** Huang and Wang say it outright. Cherny's whole method is a demonstration of it: everything concrete gets deleted every few months, and only the way of working persists.
4. **Be empirical.** Cherny states it as a technique; Huang states it as a personality; Scholl states it as a way to learn hard things. Try it, watch what happens, adjust. Nobody recommends reasoning it out in advance.

Boxes are used consistently throughout:

- **The one thing.** The point worth remembering if you remember nothing else.
- **Against the grain.** A belief the speaker held that the people around them did not.
- **Try this.** Something concrete enough to actually go do.
- **Watch out.** A trap the speaker explicitly warned about.
- **Message to my younger self.** Most speakers were asked a closing version of this question. Their answers are collected at the end of each chapter.

CHAPTER 1

Alexandr Wang: A Once-in-a-Civilization Opportunity

"This is a Once-in-a-Civilization Opportunity"

Alexandr Wang Chief AI Officer, Meta. Previously founder of Scale AI. Interviewed by Gary Tan.
Video: https://www.youtube.com/watch?v=sJ4VJWycX9M

1.1 How he got started

He grew up in Los Alamos, New Mexico, won math and programming competitions, and knew he wanted to do something big without knowing what. Two things mattered before he started a company:

- **Working at a company first.** He took a gap year at Quora. From the outside you have no idea how a company actually works: how something gets built, how it gets improved, how a group of people makes a decision. You have to see it from the inside.
- **Going to school.** A year at MIT gave him room to poke at things. That is where he trained his first models and where the idea for Scale came from.

He applied to Y Combinator after one year of MIT and started Scale at 19. His summary of what YC was for: people who genuinely want you to win, and who will also tell you plainly when you are being an idiot.

1.1.1 The first idea was the wrong idea

He came in wanting to build a medical-help assistant. They worked on it for a month or two before their YC partner pulled them aside and said he did not think it was going anywhere. Wang's read on it years later: the idea was fine, the timing was wrong, and that product is being built now. Hearing it early was exactly what they needed.

1.1.2 Where Scale came from

Training a model at MIT needed three things: a cloud account for compute, code to run, and a dataset. Two of the three you could get by pressing a button. The third you could not get at all. That gap was the whole company.

THE ONE THING

He did not find the idea by reading about a hot market. He found it by noticing a specific thing that was missing while he was trying to do real work.

1.2 Conviction is the actual job

For years, data was an unglamorous business. Revenue was good and growing, and investors still doubted it. Was it durable, was it a real business? His explanation: none of them had ever trained a model, so they could not see it. The same investors who passed now write essays about how important data is.

AGAINST THE GRAIN

You need conviction in a set of beliefs nobody else agrees with. The best companies get started long before their core idea is popular, and then work in obscurity for years until it becomes obvious. If you follow the crowd you will end up confused and nowhere. Build your own compass for what the future looks like, because everyone else will just add noise.

He also flattens the idea that founders start out capable: nobody is good at something they have never done before. You start out bad at everything. The whole game is how fast you improve.

1.3 What is actually the bottleneck now

His view: the models are not the bottleneck. Spreading them through the rest of the world is. If model quality froze today, there would still be decades of change ahead in how the economy works. That makes this a good time to be someone with a vision.

The competitive picture has changed too. Ten years ago a startup was David against Goliath and had to be clever about finding an angle. Now it is closer to Goliath versus Goliath: a small team that really leans into agents can out-compete an incumbent.

1.3.1 Personal superintelligence

At Meta the bet is that every person ends up with an assistant tailored to them, that knows their context and expands what they can do. The framing he uses is *agency expansion*. What would everyone do if everything were easy? Explicitly not a single system that runs the world: billions of individual assistants, plus an explosion of small businesses, with business agents and personal agents interacting.

1.3.2 Running a frontier lab

About a year in, having rebuilt Meta's lab close to from scratch. What struck him:

- **Talent density is the thing to bet on**, and it compounds by itself: the more strong people you have, the more strong people want to join.
- **This is research, not product engineering.** It needs experimentation and science, which is a different operating model than an internet company.
- **Build the lab to grow.** Capability, compute, adoption and usage are all on steep curves at once, so the organization has to be built like something that grows rather than something that is finished.
- **Cheap matters.** He does not want a world where models are expensive enough to be rationed to the wealthiest developers.

1.4 Each wave is roughly ten times the last

Self-driving cars were genuinely exciting, and small next to chatbots. Chatbots were maybe ten times bigger. Coding agents were maybe ten times bigger again. His expectation is that this keeps going, with new shapes of product, each much bigger than the one before, and the best ones have not been built yet.

1.5 The most concrete advice in the talk

Most of the public argument is about *when*: two years or five, will progress hit a wall. He thinks that argument is close to a waste of time, because powerful models are coming either way. What will look obvious in hindsight is that intelligence became abundant, and so did the ability to get things done.

For most of history, the limit on progress was getting a group of smart people together around a shared goal. That is what a country is, and what every company is. If intelligence stops being scarce, the scarce thing becomes vision and ambition: do you have a clear picture of how the world should be different in five to ten years, and the drive to push through the mess to get there.

TRY THIS

Look for feedback loops in a business and put agents on them. Every company is a big loop (get customers, make them happier, they spend more, you hire more, you get more customers) with smaller loops inside it. Pick a loop, define the number that says whether it is working, and let agents optimize against it. He says they have seen in-house cases where a swarm of agents with the right target beat what a hundred engineers would have done, comfortably.

Asked how this works mechanically, the answer was deliberately mundane: markdown files, scheduled jobs, and a clearly defined goal. Figure out the metric first. Most of it stops looking magic once you dig in.

WATCH OUT

Do not go all-in on words at the expense of technical depth. The abstraction layer keeps moving, from writing code to organizing people to orchestrating agents, and later to orchestrating enormous numbers of them. But rigorous, systematic thinking about systems never goes out of style. Also: ignore LinkedIn.

MESSAGE TO MY YOUNGER SELF

Build your own internal compass for how the future will go, and hold it with real conviction. You will be flooded with noise and it is hard to trust your own read when you are young and have no track record. But conviction is what lets you survive years of chaos. Then find the exponential with the steepest curve that will run the longest, and go there. It is fine if the starting point looks boring; when he started, the state of the art was spotting cats in videos.

CHAPTER 2

Blake Scholl: Breaking the Supersonic Ban

"Breaking the Supersonic Ban"

Blake Scholl Founder & CEO, Boom Supersonic. Talk followed by a long audience Q&A.
Video: https://www.youtube.com/watch?v=byAj35QlGbs

2.1 The problem he opens with

In 1969 we landed on the moon and flew Concorde in the same year. Half a century later we can do neither. Boeing's newest airliner was launched more than twenty years ago and does not fly any faster than the one that started the jet age. He reads out a recent news item that it takes Lockheed more than two years to build a single missile interceptor, and calls that no way to build a future.

Flying should be one of the most moving things a person can do, and most people now dread it.

2.1.1 Why speed mattered historically

The first jetliners doubled the speed of air travel, and the important changes were on the ground rather than in the air:

- Hawaii was not a mainstream tourist destination until you could get there in seven to nine hours.

- Nike started because its founder took a chance trip to Japan and fell in love with the running shoes there.
- Music went global partly because a worldwide concert tour became practical.

So his framing question is: what would another doubling of speed produce? What is the Hawaii, the Nike, and the global tour of a supersonic era?

2.2 Why nobody else was going to do it

He walks through each candidate and rules it out:

- **A private-jet maker.** Supersonic flight was banned over the US, and most private-jet miles are over the US. No business case without a rule change, and nobody wanted to build a plane that might be illegal.
- **A big airliner maker.** They were not building genuinely new airplanes at all, let alone radical ones.
- **A startup.** Widely considered laughable. One of his failed pitches was to Jeff Bezos in 2015, who then wrote in a shareholder letter that some things only big companies can do, citing airliners as an example.
- **Him personally.** He was a software engineer whose aviation knowledge was a private pilot's license.

What he took from that: the only way to find out what he was capable of was to pick a mission that inspired him and give it everything.

AGAINST THE GRAIN

The belief that almost stopped him: that every technology gets built at the earliest moment it possibly could, so if your idea is good and nobody is working on it, something must be wrong with the idea or with you. Combined with investors asking "why now?", this funnels founders into a small pool of recently-unlocked ideas.

He says it is simply false. Many great ideas are sitting in plain sight, solvable for years, unbuilt for no reason other than nobody went and did it. The technology for supersonic flight existed long before Boom. His answer to "why now?" is *because I started now.*

2.3 From cardboard to the sound barrier

Boom started in 2014 with a mockup made of cardboard, plywood and office-supply chairs. People made fun of the name. Roughly a decade later, their prototype became the first independently developed jet to break the sound barrier. Not a government, not a military.

2.3.1 Optimize for two days

His framing for being a founder: you optimize for the worst day and the best day.

- **The worst day.** Boom nearly died several times. At one point they were down to seven days of cash and the board told him to shut it down. He said no; they just had not found the way through yet.
- **The best day.** Watching the team roll the airplane out and go supersonic.

The reason the worst days were survivable is that he wanted the thing to exist in the world. That is the argument for building something that matters beyond yourself.

2.3.2 The accidental breakthrough

The prototype went supersonic on two flights, six times through the sound barrier, and never made an audible boom on the ground. They demonstrated a long-discussed principle: break the sound barrier high enough, at the right speed for that day's weather, and the boom bends back up and never reaches the ground.

That sidestepped the argument that had blocked supersonic flight for fifty years. The question had always been "how quiet is quiet enough?", and with no boom at all there is nothing left to argue about.

THE ONE THING

It was about 24 hours from breaking the sound barrier to being invited to the West Wing, and 115 days to an executive order making supersonic flight legal again. His line: people ask whether it is a bad idea to build in a regulated industry. It turns out you can change the regulations, by building the thing and explaining clearly why it is fine.

2.4 How 50 people did what used to take thousands

The core move: make hardware development look more like software development by driving down the cost of each iteration, in both bits and atoms.

2.4.1 Iterating in bits

Good software is modular: change one part and other parts do not change. Good airplanes are the opposite: everything affects everything. Add a row of seats and the body gets heavier, the engines need more power, the wings need redesigning. Classically this meant many specialists each working in their own spreadsheet and handing results to each other, so iteration was slow and eventually you gave up and built whatever you had.

Their fix was to move all of that engineering out of spreadsheets and into real software. Spreadsheet engineering, as he puts it, is code being treated as a second-class citizen, with no automated integration, no automated testing, none of it.

- One tool lets you define an airplane in a configuration file and simulate the whole aircraft in minutes: how far it flies, its fuel burn, how many passengers, what the trade-offs are.
- That turned into a search problem. You cannot find product-market fit on a supersonic jet by building one and seeing if people like it, so they explored the space of buildable airplanes digitally and narrowed in on the right one.
- A second tool does the same for engine blades: change a blade design and see the structural and aerodynamic result in real time. That is work which used to take dozens of engineers months.

He notes that falling software costs make far more of these custom tools affordable than before.

2.4.2 Iterating in atoms

The hard-won lesson was that iterating through outside suppliers is extremely difficult, so they built their own machine shop and their own test facilities.

- Digital design to prototype part in about 24 hours, with engineers working alongside the technicians who make the parts.
- Their own engine test stand, half an hour from the engineers, so they can test whenever they want rather than booking a slot in someone else's facility.

2.4.3 How they funded a decade of R&D

Because they had chosen to build their own engine, they could sell that engine separately as a ground-based power turbine, shipping without the rest of the airplane. That generates test data, reliability learning, and the capital to keep going.

> **THE ONE THING**
> The vertical integration they did for control turned into the financing strategy. A component built for the hard long-term goal became a product that pays for the hard long-term goal.

2.4.4 Building in public

They chose to livestream the supersonic flight: the first supersonic satellite-link installation and the first supersonic air-to-air livestream, filmed on a phone. Millions watched. His favorite clips were from classrooms of kids who were not previously into aviation. Somewhere in one of them, he says, is the next chief engineer of another supersonic jet.

2.5 From the Q&A

2.5.1 Competing with China on manufacturing

Yes, but not by trying to bring back the manufacturing that left. You do not beat China at its own game. You invent the next wave of manufacturing and build that here.

His worked example is tooling. Building hardware normally requires custom precision molds and fixtures, such as a wing-sized mold to lay up carbon fiber in, or steel tools for metal parts. Much of that trade left the US and the skills went with it. So Boom is trying to eliminate tooling entirely: an all-digital process going from a digital design to an advanced turbine blade in 24 hours with no tooling at all. The point is to compete on invention rather than on labor or on an industry the US no longer has.

2.5.2 What AI has and has not changed for physical products

- **Changed:** the cost of building custom software, which means custom engineering tools that would previously have been unaffordable. He draws the analogy to early Amazon, which built its own recommendation and warehouse software and fit it to the business like a glove, but needed enormous scale to afford that. The other half is that software is usually built by people in a different company from the people using it, and now any user can become a tool builder.
- **Not changed:** there is still no good AI for the physical world. Programming a machine tool is still a manual job. He would love a machine that goes from a design straight to an automatically inspected part with no programmer.

2.5.3 Regulators: engage or launch anyway?

Not one-size-fits-all. It depends on whether you have an entrenched opponent.

- Ride-hailing launched first and used its customers as allies, because the incumbent taxi industry was actively using regulators to block competition. He notes this is a common pattern: much regulation comes not from someone protecting the public but from an incumbent running to a regulator to shut down a competitor.
- Aviation was the opposite. Safety-critical, and no entrenched enemy: the big manufacturers thought Boom was cute and would never succeed. So he went to the regulator early, explained what they were doing and why, and asked for help and feedback, showing plans before they were final.

The result: an approvals process that can stretch for months took 90 minutes, and they got the first-ever permit to fly a civil airplane supersonic before the airplane had flown at all. His summary: they made it *us plus the regulator versus the problem*, not us versus the regulator.

2.5.4 Hiring

His industry has both a total absence of startups and a set of large companies with cultures he does not admire, which creates constant tension between technical skill, culture fit, and experience.

- Value early-career people: young, ambitious, hands-on, optimistic, and not yet shaped by a legacy company.
- Promote from within where you can, especially for senior roles.
- The one thing you cannot teach is caring. You can teach any skill or any body of knowledge; you cannot make someone care who does not.

TRY THIS

Their rule about experience: it is fine for a young engineer to do something they have never done, just as it is fine for them to do something nobody has ever done. But if somebody in the world has done it before, you have to find that person, call them, and ask their advice. You do not have to take it; you just have to hear it. He finds accomplished people are rarely unwilling to take that call.

WATCH OUT

His nomination for worst common advice: "work on something you know about." His first company was mobile commerce because that is what his resume said he was good at. He could build it, but he had no idea what it *should* be and did not care about any of it. So: high-quality output, no product vision, no fun. Knowledge and skills you can change; what you love you cannot. Especially now, when anyone can learn anything with a patient tutor.

2.5.5 On self-belief

He did not have it on day one, and friends' eyes glazed over when he described the idea. What shifted it was thinking about the founders he admired, and in particular the goal of putting a personal computer on every desk, stated at a time when personal computers mostly did not exist. Nobody taps you on the shoulder and tells you that you are the one who gets to do the big thing. That conversation never happens; no one grants permission.

So he stopped putting energy into *whether* he could and put all of it into *how*. Some mornings he still wakes up doubting it, and treats that as normal.

2.5.6 Teaching yourself hard things

Know the difference between knowing enough to pass a test and actually understanding something. Learning aerospace engineering, he bought textbooks, went back and redid high-school physics and calculus, and did the problem sets.

TRY THIS

Keep a *confusion list*: things you cannot honestly say you understand, even if you could answer a test question about them. His goal was to remove one item a week. In practice the list grew without bound, but it built the self-awareness to tell the difference between really getting something and waving your hands, which is what lets you stay on a topic until it is actually clear.

2.5.7 Careers, cities, and balance

- **Job worries.** He thinks the doom is confused: Boom needs *more* software engineers now, not fewer, because the cost of writing software fell, anyone including hardware engineers can now produce code, and someone has to make sure the architecture is coherent.
- **Start a company or join one?** Not one-size-fits-all. If there is a company you cannot get out of your head, start it now. Otherwise go work with the best people you can find on a problem you are excited about. The magic is the intersection of what you love, what you can get very good at, and what matters to the organization around you. Hit all three early and your career goes vertical. He was given a large budget on a project the CEO cared about at 23, which meant that when he messed up, the most senior people were invested in helping him not fail.
- **Do not start a company just to start one.** The genuinely bad outcome is running a successful company you do not love, feeling responsible to employees and investors, and being effectively stuck. Only start a company you want to keep.
- **San Francisco?** Not required. He left it to build Boom in Denver, because building physical things affordably at scale there was not feasible. But unless you have a specific reason not to be, be in an ecosystem of ambitious people.
- **No work experience yet?** Side projects. Show the most impressive thing you have built, with pictures. The best people find ways to do things at unusually young ages, and the idea that you have to be a junior in college to get an internship is nonsense.
- **Work-life balance.** He argues for harmony rather than balance. If the work is meaningful it is part of your life, not the opposite of it. He started Boom with three children under two. He finds parenting lessons and leadership lessons cross over constantly.

MESSAGE TO MY YOUNGER SELF

Pick a problem you want solved in the world, and stop worrying about whether the timing is right or whether you are the right person. Build things that matter to you and to other people, because that is what carries you through the bad days. And build in public, because it recruits the next generation of people who will build the thing after yours.

CHAPTER 3

Sam Altman: Never a Better Time to Do a Startup

"Never a Better Time to Do a Startup"

Sam Altman Co-Founder & CEO, OpenAI. Formerly president of Y Combinator. Interviewed by Gary Tan.

Video: https://www.youtube.com/watch?v=ZIaOBAjvc38

3.1 2005, and how far away it feels

He was in the very first YC batch, with a location-sharing company. His description of the era: startups were not cool at all, eight companies hiding out in a small building, their YC partner cooking them dinner every week. The thing each of those companies spent three months building could now be done in minutes by a coding agent, and that was only twenty years ago.

His point is that you can read that fact two ways. You can be sad that a whole startup is now one prompt, or you can notice that you are now able to start the most ambitious company you can imagine, with expert-level help in every field, and take on hard technical problems that were simply impossible before.

3.1.1 On Paul Graham

His memory is of walking in every week feeling hopeless and dejected, and walking out having been convinced that his startup was about to take over the world. Creating optimism, momentum and belief out of nothing was the particular skill.

He also passes on the framing he found most useful: a good startup investor is a kind of teacher, but not the lecture kind. The flight-instructor kind: someone sitting next to you saying *do this, not that, you missed that, you are making these mistakes.* That hands-on redirection of undirected energy is what he needed at twenty.

3.2 Why startups matter, in his view

- They are the main thing that keeps an economy from going stagnant, because companies drift toward being bad over time.
- If AI is as big a change as he thinks, startups become *more* important, not less: they are how the power of the technology gets spread through the economy rather than concentrated in a few companies or models.

He describes running YC largely as being the unofficial flag-bearer for that argument, fighting for why startups matter and putting relatively more power in founders' hands, which he thinks turned out better for everyone, investors included.

3.3 When startups win

His general observation: startups win when the technology landscape is moving fast, costs are falling, and cycle times are short. Looking at history, great startups cluster when the ecosystem shifts and incumbents lose their advantage: the arrival of the internet, of platform apps, of the phone app store.

He thinks all of those conditions hold now, plus a new one: the expertise that used to be hard to acquire (hiring excellent people who could do specific things you needed) has shifted. He has seen small teams who grew up using these tools automate most of a startup with four people and a lot of compute.

THE ONE THING

Taste, agency, and an understanding of how businesses actually work (where value accrues, what a real advantage looks like versus a fake one) still matter. But he would bet this moment cuts against many years of experience and in favor of people who are fluent with the tools.

3.4 The core advice: be right and disliked for a long time

Ten years ago, the reaction to OpenAI was not just that they were wrong about whether this was buildable, but that they were being irresponsible and would single-handedly set the field back.

AGAINST THE GRAIN

His highest-order piece of advice: find something you can develop reasonable conviction in that the conventional wisdom says is wrong. Then be okay with it taking a long time and with people being dismissive throughout.

For years it felt like they knew the biggest secret in the world while being called idiots, with slowly accumulating evidence that they were not deluded. It was extremely frustrating, and in hindsight a gift, because it meant no serious competitors and time to do the research and build the company.

The flip side: if you are starting the same company as everyone else, you get plenty of attention and can raise plenty of money, but those are much less often the big outcomes.

He connects this to a favorite point about exponentials: people are bad at intuiting them, which is how the world missed this one. There are new exponentials forming right now. He does not know what they are. If you can identify one and keep building conviction as more data arrives, be grateful that nobody else understands it yet.

3.4.1 You need a few people, not many

They used to joke that only fifty people in the world believed this was possible, and that was fine because forty-five of them worked there. Saying the heretical thing is probably *why* those people wanted to be in one place.

> **WATCH OUT**
>
> If you cannot find anybody who shares your belief, pay attention to that. Doing a startup alone is also very hard and very lonely. But you do not need many people, and if everybody believes it, that is a bad sign too.

3.4.2 Finding your people

He says the limit on how many good startups YC could fund used to be co-founder matching: lots of talented people who could not find their tribe. His practical advice is almost anticlimactic: find a way to be mildly helpful to a lot of people. It is enjoyable, you see interesting things, and it compounds over a very long horizon.

His own example: he was an early investor in a payments company at 22, and they asked him to drive down and have dinner with a candidate to convince them to drop out and join. Eight years later that person co-founded OpenAI with him. He says they could spend the whole session on stories like that: unpredictable at the time, decisive later.

> **WATCH OUT**
>
> His extended aside, delivered as a rant: it is very easy to take shots at people trying to do hard things, get thousands of likes, and feel like you did something meaningful. It is very hard to build something of value. Looking back at a decade of people mocking YC and startups while he ran it, he figures many of them felt like they got him on any given day, and over the decade none of them did. His warning is not about the target. It is that it will have an effect on *you*, in an insidious way. Blowing off steam is fine; hold yourself to a higher bar than sarcasm.

3.5 Safety, honestly stated

Asked about a recent AI security incident, he does not downplay it or inflate it.

- It is not a large loss-of-control event, and he does not want to claim it is.
- But if you had asked people ten years ago where on the spectrum from nothing to superintelligence you would place *a system escaping its sandbox and getting into another company's systems*, most would have put it much closer to the far end. The goalposts have moved.
- He calls it both an alignment failure and a security failure, says OpenAI made some big mistakes, and treats it as a real reminder that loss-of-control accidents are not purely theoretical.

3.5.1 Concentration of power

The thread he keeps returning to. Concentrated power has been bad in essentially every period of human history. He can imagine AI producing the widest distribution of power ever seen, and he can imagine it producing the narrowest.

THE ONE THING

He does not think anyone should be locked into one company's or one person's moral worldview, including his own. That is the argument he gives for OpenAI being reserved about imposing its views on what people should build, while still holding a safety floor. And it is why he thinks startups have a real role: economic concentration in one company or one model is one of the ways this goes wrong.

Asked what someone at the start of their career can do about any of this, his answer is direct: start a successful startup. That alone, multiplied across many people, keeps the economy distributed and is a large contribution.

3.5.2 The dystopia he names

Not the obvious one. The version he says he is particularly nervous about is over-correcting on safety: you get a cure for cancer and material abundance, and in exchange you get no freedom, no privacy, a perfect surveillance state, and nothing left that really matters to do.

His proposed test for whether the next ten years went well: every year, do people have more freedom and agency, more control over their time, more of their days spent on things they actually want to do, and more long-term fulfillment? If that number goes up annually, you probably avoided the catastrophes.

3.6 Scale, and the demand for intelligence

- He expects the next six months of model progress to feel like roughly the last two years.
- He thinks the world will effectively never be out of a compute shortage. He has never seen a commodity like it, and demand for good-enough intelligence at a low-enough price looks essentially uncapped.
- His contrast is electricity: demand rises as price falls, but eventually it gets hard to think of new things to do with more electricity. With intelligence there is always something else to make better.

His illustration: six and a half years ago the heaviest user of these systems he knew of was consuming an amount that seemed ludicrous at the time, while the worldwide average per person was zero. Today the worldwide average is around what that heaviest user consumed, and the heaviest user is orders of magnitude beyond that. If that repeats, the average person's usage in another six and a half years is staggering, and it will just become the expectation.

3.7 On ideas that are too ambitious

His reaction to "I have an idea that feels too ambitious" was to ask to hear it.

He notes a pattern in the best ideas: the top-level vision is clear, and the first steps are deeply unclear. They wanted to build general intelligence, and it genuinely did not occur to them that they would become a product company. It took years to arrive at the products people know. If the vision is clear and the first steps are murky, that is normal for very ambitious ideas and not a reason to lean out.

WATCH OUT

The matching failure mode: having a brilliant big idea and never making any forward progress with it. At some point you have to take imperfect steps and get new data points. Theirs were certainly imperfect.

TRY THIS

His reframe on productivity, stated plainly: you can now do three months of work in seventeen minutes, so go do three months of work in three months of work, whatever the new bar for that is. The point of the speed-up is a bigger output, not a shorter day.

MESSAGE TO MY YOUNGER SELF

It is all going to work out. Doing a startup is genuinely stressful, his first one did not work out well, and it was a hard period of his life. Looking back, you can make a lot of mistakes and fail at a lot of things, and this industry is unusually forgiving of that. He wishes he could have kept all the drive and ambition and just been a bit happier along the way, trusting that it would eventually be fine.

CHAPTER 4

Boris Cherny: We Cut 80% of the Prompt

"We Cut 80% of Claude Code's Prompt"

Boris Cherny Head of Claude Code, Anthropic. Recorded the day after a major model release.
Video: https://www.youtube.com/watch?v=qyPCVqFUyDo

4.1 What changed with the newest model

He is candid about how model training works: you try to teach a lot of things, most of it does not work, some subset lands, and occasionally the model turns up with an ability nobody taught it. Two things he singles out:

- **It runs for a very long time without stopping.** Days or weeks on one task, without needing extra scaffolding to keep it on track, because it understands that the task needs finishing.
- **It resists being hijacked by instructions it reads.** A year ago, if the model read a malicious instruction in a web page or file, telling it to do one thing and also delete everything on the user's computer, it would have complied. Now it does not.

On that second point he describes three layers stacked together: a model trained over years to behave well, a classifier run over all traffic that watches for hijack attempts, and a third check on autonomous operation. Notably, the hijack detector works by watching which parts of the model's internals light up when it happens, so even if the model does not report it, they can see it.

4.2 Deleting the prompt

The headline: they removed more than 80% of the instructions that ship with the product.

His explanation is that the product is always changing, and every time a new model comes out they delete and rewrite large parts of the instructions and change the set of tools. Every model is genuinely different, so what you built for one model three months ago may not transfer at all. Much of what had accumulated was correcting for behaviors the model should have gotten right on its own. The new one does.

TRY THIS

Two things he suggests trying yourself:

- Override the instructions entirely with a command-line flag and experiment.
- There is an undocumented *simple mode* environment variable that strips all the built-in instructions, including from the tools. They use it internally as a test. His finding: with the instructions gone, the model is actually slightly *smarter*, but you still want some of them, because they make the product behave the way a person using it would want.

4.2.1 The method: delete, then add back one line at a time

The technique is borrowed from research, where you remove a component to measure its contribution. The loop:

1. Delete everything.
2. **Use it.** Do not guess what instruction the model needs, because you will guess wrong. Run the actual product or the actual codebase.
3. Watch where it fails and where it does well.
4. Only when it stumbles *repeatedly on the same thing* do you add that instruction back.

The reason for the discipline in step 4: the model reads every instruction every single time it runs, so each one needs to earn its place.

THE ONE THING

His advice for people who are not building on models but merely using them: every six months, delete your project instructions, your custom skills, and your hooks. See what the model does. It may surprise you, and it may not need any of the guidance that older models needed.

4.2.2 Almost none of the remaining code is prompting

Asked what is actually left in the product now, he says nearly all of it is safety, permissions, static analysis and interface code. A lot of the rest has been removed.

4.2.3 What survives a model change, and what does not

- **Instructions and harness code:** disposable. Rewrite on each model.
- **Tests that measure quality:** they last longer, but not much, maybe one to three model generations. He explicitly declines to call them permanent. What usually happens is the model maxes out the test, so they throw it away and build a harder one.

Either way the instruction is the same: be empirical. Use the thing, find where it struggles, and build your measurement from what you observed.

4.3 The central idea: get out of the model's way

Two terms he uses, which are two sides of one thing:

- **Untapped capability.** Today's model, not a future one, can already do things nobody has realized, and often no product exists that lets it express them.
- **Getting in the way.** The product actively preventing the model from doing what it could.

His origin story for Claude Code is exactly this. Roughly two years ago, the best coding model of the day was being used for single-line and multi-line autocomplete, and for read-only chat about a codebase. His read was that no product was letting the model do what it could actually do: write a whole function, a whole file, a whole small feature. So the product was to strip away the scaffolding and give the model the simplest possible setup with real access.

THE ONE THING

His claim to the room: with today's models there is a large amount of untapped capability that startups are not capturing. Anyone could build the next thing of this kind by finding a capability the products of today are suppressing.

4.4 How to find untapped capability

4.4.1 Ask for something slightly too hard

The common mistake he sees is over-specifying: telling the model to do the task, and then dictating step one, step two, step three, in exactly the order you would have done it. For modern models that is the wrong approach.

TRY THIS

Go a level higher. Describe the task, describe the boundaries it must respect, describe what *done* looks like, and then let it work and come back later. Give it tasks slightly harder than you think it can handle. He notes this would not have worked six months ago and does work now.

4.4.2 Throw the newest model at your oldest stuck problem

His worked example: their product runs on an alternative JavaScript runtime that was written in a low-level systems language with manual memory management, which made memory bugs easy to introduce. For a while, the useful thing the model could do was hunt those bugs one at a time.

At some point an engineer on that team decided to just ask for the whole thing to be rewritten in a different language. That request had been his recurring test for each new model generation, and previous ones could not do it. This one could. The setup mattered: the codebase had a large, thorough test suite, so it was easy to tell whether the result was correct. It ran for 11 days on one prompt, with occasional steering, and rewrote a codebase of over a hundred thousand lines. It is what ships in the product now.

Interviewer's framing: with the best engineers, that work would have taken over a year. His agreement was unhesitating.

TRY THIS

Keep a list of business, product or engineering problems you gave up on, and re-throw the newest model at them each time one ships. The previous model failing tells you nothing about the next one.

4.4.3 Play with no commercial purpose

The internal example he gives is someone discovering that if you hand the model an image library, it can draw portraits, animals and landscapes reasonably well. Nobody trained it to draw. It was found by messing around with no application in mind. His hypothesis is that there are dozens or hundreds of capabilities like that sitting undiscovered.

4.5 Verification is the thing people get wrong

Asked whether prompt engineering still matters, he says the labels keep rotating and the actual skill now is: give the model a task that is a bit too hard, and then make it possible for the model to check its own work along the way. Verification, he says, is probably the single most important thing people do not get right.

4.5.1 His two-week example

He wanted to know how their desktop application would feel if rewritten natively. So he started a session in a chat app and:

1. Asked whether it had access to a machine to run on. It said no, so he connected one.
2. Asked whether it could see an empty repository he had created. It said no, so he granted access.
3. Then gave one instruction: rewrite the app in the native language, run the old app in the virtual machine, take a screenshot, compare it pixel by pixel against the new version, and do not stop until it is done.

Asked how long it took to run: it is still running, over two weeks at the time of the talk. It also decided on its own to create a chat channel and post progress screenshots every few minutes.

THE ONE THING

That prompt is simple and anyone in the room could have written it. What makes it work is not cleverness in the wording. It is that the model was given a way to check its own output (run it, screenshot it, compare) so it never gets stuck and never has to guess whether it is finished.

WATCH OUT

Do not look for one weird trick, because there is not one. And do not take your process from social-media advice. The approach is: give it a hard task, give it the means to verify the work the way you would verify it yourself, see where it struggles, and fix that specifically, with better instructions, with a reusable skill, or by connecting it to the context it is missing. That is the whole method.

4.5.2 The failure mode of experienced engineers

He is direct that this is hardest for people who have been building software for a long time, because over-specifying used to be how you had to do it. The common failure is trying to make the model do the task exactly the way you would have. He says a lot of people are in the middle of unlearning this, and it is a journey. The mental model he suggests: treat it like a coworker, because that is the level it operates at now.

4.6 Running many agents productively

He describes two different shapes, and stresses they are for different situations.

4.6.1 One task, split into pieces

The first is a feature where you simply say to use a workflow. Under the hood it starts a sandboxed environment and lets the model launch and coordinate other agents. What it does is not one agent, and not ten in parallel:

- Fan out a set of agents for a first pass.
- Then a second set to verify or summarize that work.
- Then fan out again for a third stage.

He explains the design came from his background in functional programming: essentially a small grammar for combining agents, with ways to run them in sequence and in parallel, so the model can compose them to do genuinely complex work while using tokens efficiently.

THE ONE THING

He frames this as a new way to spend more compute at the moment the task is being done. Historically, models got better through network size, training data, and training compute; then through how much thinking a model does when answering. This is a further step: orchestrating that thinking across many agents, and dialing it up a long way for a genuinely hard task.

4.6.2 One task, repeated forever

The second shape is a scheduled job, either on your own machine or in the cloud so you can close your laptop. The distinction he draws: a workflow is one task broken into chunks that share context; a scheduled job is one repetitive task that does not share context but may share memory, run hourly or daily.

They now have the product maintain its own codebases this way: for the command-line tool, the mobile apps, and the desktop app. Each routine is roughly one sentence. Examples he lists:

- Find dead code across all the codebases and open a pull request removing it. He notes they did not tell it how; it worked out that it should use both static and runtime analysis.
- Find experiments that have already been rolled out to everyone, delete the switching logic, and ship it.
- Write tests for areas that need coverage.
- Delete tests that should not be there, such as ones added by older models or by people long ago that serve no purpose.
- His favorite, nicknamed the abstraction police: in a big codebase the same concept often gets built several times in different places. This one goes looking for those near-duplicates every day and unifies them.

Around 20 or 30 of these run daily across their codebases: hundreds to thousands of agents a day, doing what used to take a large number of engineers. The point he lands on is what the engineers do instead: ship new product, talk to users, and do the parts that are actually interesting.

4.7 Is coding solved?

He has said so publicly and immediately qualifies it: solved for the kind of coding *he* does, not for everyone. Still hard: deep systems code, distributed systems, and fine-grained visual checking where something is off by a pixel. Vision and computer-use took a big step but are not perfect.

He polls the room. A fair number of hands go up for writing no code by hand at all; slightly fewer for more than half.

4.7.1 What separates people who are good at this

Not a technique, but a mindset. Be empirical. Forget what you learned about older models. Set aside the theory you learned in class. Watch the model attempt a task, see where it struggles, and adjust from that. He describes the shift as going from a theoretical science to an experimental one, and says the people who are best at this are good at letting go of their assumptions and willing to retry an idea that did not work before.

MESSAGE TO MY YOUNGER SELF

He learned to program practically, always to solve a specific problem he had. It started on a graphing calculator in middle school, writing an algebra solver so he could do better on math tests, then handing copies to classmates. When the math got to calculus, his solver was not good enough, so he learned a much lower-level language to write a better one.

So his advice for students is: learn the computer science, which is genuinely fascinating, but also learn to *apply* it. Build products, develop a design sense, develop business sense, learn to work with data, learn to talk to users. Combined with engineering, that combination is where it becomes valuable. Those are the skills he would still be practicing by hand.

CHAPTER 5

Jensen Huang: The Mindset That Built NVIDIA

"The Mindset That Built NVIDIA"

Jensen Huang Founder & CEO, NVIDIA. The opening session of Startup School 2026, interviewed by Gary Tan.

Video: https://www.youtube.com/watch?v=l4B37S1dyQQ

5.1 The founding technology was wrong

The thing he says people do not believe: the technology the company was founded on was flat-out wrong.

The company's actual thesis was sound. General-purpose processors are useful, but if you add a specialized helper alongside them you can solve problems that are otherwise too hard. The first problem they picked was 3D graphics, and in 1993 the personal computer was only rumoured to be coming. They had grown up on game consoles, so the idea was to turn every PC into one. They invented new methods, reasoned carefully about them, believed in them, and started the company to build it.

By 1995 they realized the approach was wrong, and by then 35 or 40 other companies were chasing the same market. Worse: nobody at the company knew how to do it the right way either.

THE ONE THING

His response was to go to an electronics store with a couple of hundred dollars, buy three textbooks on graphics and pipeline design, and hand them to the engineers. His summary: they raised money and bought textbooks. Everyone assumes NVIDIA started as the world leader in graphics, and they learned it from a book.

AGAINST THE GRAIN

The lesson he draws: technology changes constantly, so as long as you can confront reality and you can learn, the specific technology does not matter that much. Every new domain since has been approached the same way: if it is important, go learn it, and *how hard can it be?* It always turns out much harder than expected. But that is the attitude you go in with.

5.2 The real idea was never the chip

They realized early that the business is not building a great chip. It is accelerating a whole domain of computation. Molecular dynamics, image processing, fluid dynamics, inverse physics, and eventually deep learning are all instances of the same pattern.

THE ONE THING

What makes a great company, in his words: a high-level view of the future of something important, that is somehow unique, that you deeply believe in, and ideally that is hard to pursue. Technology, market and timing all matter and make life easier when they line up, but the unique conviction is the thing.

5.3 The Sega story

The project that revealed the flaw was a contract to build a game console. Because the technology was fundamentally broken, he flew to Japan and told the CEO that they could not deliver, explained why, and advised them to hire someone else, and then said he still needed the money. He recounts the reply: *so what you are telling me is that you cannot do what I contracted you to do, but you would like all the money.* He confirmed that was exactly right.

He got it. The money kept the company alive long enough for him to figure out what to do. His reading of why, addressed to the investors in the room: you do not invest in companies, you invest in people. What the other CEO recognized was that he was being honest.

For scale: NVIDIA went public in 1999 at a valuation of $300 million, which he notes was real money at the time.

5.4 Seeing deep learning for what it was

He saw AlexNet at the same time everyone else did. The difference was the lens. He was always looking for algorithms his hardware might accelerate. So the questions he asked were: what is this method, why does it work so well, what else can it do, and what could it solve if you scaled it far beyond this?

THE ONE THING

The breakthrough was realizing that AlexNet was not really about AlexNet. It was a general method for learning *any* function: give it the answers and it works out the function. And most genuinely interesting problems are ones where you cannot be precise anyway. Once you accept that, the question becomes what that does to the entire computing stack, to software itself, and to every industry it touches.

He describes reimagining the whole stack (processor, middleware, algorithms, applications) about fifteen years ago, and starting almost immediately on computer vision, robotics and self-driving cars. His description of the method is deliberately plain: ask questions, reason from first principles, and keep asking *if this, then what? If this gets better, so what?*

5.5 Being in the weeds as CEO

Asked how he can read papers and talk directly to researchers while also running a large organization:

- It starts with curiosity. He has his own questions and will take the shortest path to an answer, but the answers from people near him are often unsatisfying, or those people are busy with something else. So his first instinct is to go find out himself.
- If the area turns out to be important, his next instinct is to learn as much as possible so he can be *of service* to the company and share it. He describes a CEO as being in service of the people who work there, and wanting to empower them with insight.
- He is explicit that this is a personality trait rather than a management technique.

5.5.1 Why proximity matters when things move fast

When technology is changing quickly, without a hands-on feel for what is happening it just looks like it is moving too fast to follow. But if you understand the underlying principles, it starts to make sense. His analogy is surfing: to an outsider the ocean is chaos, but a surfer reads the waves and stays on top. You cannot read the waves or the wind or get the timing right without actually going out and doing it.

TRY THIS

His framing on organization design: you are building a car that *you* are going to race. Build it so you can drive it, adapting the company to yourself rather than to convention.

Asked what happens to the company when he is gone, his answer was that they will simply have to reshape it for the next person. The reason he calls that wisdom: you are the driver, the world is competitive, and you have to win. Whatever it takes to fit the car to you is what you do. The next driver can figure out their own fit. He says he is constantly tweaking the company and its processes so he can be more effective in it.

5.6 Systems thinking is the durable skill

His answer to what will matter most: systems awareness, systems design, systems thinking. The reasoning is that the low-level work is going to be automated anyway. In his generation, synthesizing the individual pieces of a chip was automated, and now most of their designers are systems designers. He expects the same for software.

So the useful questions become abstract ones: what problem are you solving, what are the constraints, where does the input come from and where does the output go, how fast does information flow through, and what is the actual limit: processing, memory, or networking?

THE ONE THING

He does not think that kind of knowledge ever becomes useless. He thinks it becomes more useful.

5.6.1 What agents need next: fine-grained control

His observation is that agents already improve themselves at a coarse level. Every use updates their notes and long-term memory, which gets compacted or reorganized in the background, so they get a bit better each time.

The problem he flags as worth solving is that our influence over them is too coarse. What he wants is the ability to change one word in a plan and have that make a specific difference rather than a completely different result: one pixel, one triangle, one component in a design file, one connection, with everything else regenerated around it.

THE ONE THING

His point about why this is enough: agents do not need to be perfect for us to use them. They could be 80% and we help with the rest, or 99% and we help with the rest. Controllability, he says, is probably the single biggest breakthrough needed for agents at every level.

5.6.2 Why NVIDIA cares about agent software at all

Three separate reasons he gives:

1. **Agents are the new software**, and how that software gets processed matters enormously to computer architecture. The more intimately they understand the shape of the work, the better the systems they can design. He notes they have to live five to ten years in the future, because it takes about three years to build a system, a couple more to ramp it up, and customers want to use it for ten years after that. So the questions are: what is the workload, how will it evolve, where are the bottlenecks, what happens as more of it runs at once, how do sandboxes and memory work.
2. **To make NVIDIA itself faster.** They have agents running autonomously throughout the company, and deliberately let people choose their own tools, letting a thousand flowers bloom, and learn from all of it.
3. **To find what comes next.** His example: seeing early work on step-by-step reasoning years ago and asking how effective and how scalable it would be. Thinking that through led somewhere unexpected: if a system can reason from knowledge it already has, maybe a self-driving car needs far less driving data. That produced a car that reasons, which drives well on a couple of million miles. His explanation is that this is how people work: we do not need many miles either, because we have prior knowledge from language and can break down a situation we have never seen using things we understand.

5.6.3 On open source

He is emphatic. Without open source, he argues, the mobile and cloud industry would never have happened, and modern AI could not exist, naming the succession of open frameworks that everything was built on.

His view is that everyone should use the big hosted services, *and* that the world needs the ability for anyone to build their own AI, particularly companies that need domain-specific systems. He notes this is relatively easy now precisely because the software is intelligent enough to adapt itself, and says he has no idea what innovation will come out of it, which is the point.

There is a nice aside on words: asked whether text files are "just text," he answers that words are thoughts, and invites you to try thinking without them.

5.7 Jobs

This is his most direct disagreement with the prevailing worry. He grants that the effects are uneven and that many tasks will be automated away. But:

THE ONE THING

The narrative that AI destroys jobs is, in his words, exactly backwards. AI eliminates *tasks*, not jobs. A job has a purpose, and that purpose contains many tasks, some automatable, many not.

His evidence is a set of cases where the task was automated and the job grew anyway:

- Coding has been substantially automated, and the number of software engineering jobs has been rising year over year.
- Reading medical scans has been automated, and the number of those specialist jobs has risen, because the backlog of patients is enormous. To admit more patients you need more nurses and more specialists.
- Legal research tools were supposed to eliminate paralegals; those roles are growing fast, because the backlog of cases is high and firms can now process more of them.

The common thread is backlog. The backlog of ideas and ambition is so large that automating programming means you hire more engineers and attempt more things. He calls it a classic case of productivity increasing growth, and growth increasing employment.

5.8 Physical AI and robots

The moment that convinced him was seeing AI generate video. His reasoning was immediate: if you can generate video of a finger moving, or a hand picking up a glass, why can you not make a robot do the same? So robot movement was around the corner.

The remaining questions were about physics. How does a system come to understand causality, friction, tension, and generate motion that obeys physical law? That started the work on models that understand how the world works.

His claim is that the equivalent of the chatbot moment for robots already happened a couple of years ago. His justification is a good reframe: when chatbots first appeared they did nothing useful either. What they did was open our imagination about what was possible.

What remains is the same work being done for agents: build environments for them to learn and be evaluated in, generate simulations grounded in physics, and then bridge from simulation to reality.

5.8.1 Where it shows up economically

They deliberately chose self-driving cars as the first application, because it was the one with a large enough market, standardized enough technology to scale, and real economic value. They are now in cars and data centres across several manufacturers.

He explains why they open-sourced their self-driving stack: you need autonomous navigation for agriculture, mail delivery, and warehouse robots, and none of those individual markets is large enough on its own, but they were diverse enough to justify building the whole stack and giving it away.

He puts the current robotics and physical-AI business at roughly $10 billion, expects it to be one of the largest industries in the world, and says it will take longer than two or three years and less than ten.

5.9 On joining a public conversation late

He notes he made his first post on X in 2026, and jokes that it shows how introverted he is, probably the last person on earth to do it. His reason for finally doing it was that what he wanted to say was too important to him, to the industry, and to the world.

5.10 What a young person should learn

The simple stuff gets automated. By simple, he explicitly includes the act of sitting at a computer writing code, comparing it to long division, which also got automated away.

What does not go away:

- The hard sciences: physics, chemistry, biology.

- Computer science and computer engineering.
- Systems thinking, so you can orchestrate very large numbers of agents solving problems on their own.
- Especially the places where domains intersect, and where technology meets social questions, market gaps and opportunities.

His argument is that AI is a tool that lets you be more ambitious and more impatient about enormous, genuinely hard problems. His illustration from his own career: when he graduated, a chip with a thousand components was a very large chip. Today, being told the next chip has a trillion would not even register as remarkable, because the scale of the task has stopped being the constraint. You no longer have to think about how much coding or how many engineers. You only have to think about what problem you are solving.

His parting instruction on the subject was blunt: stay in school.

MESSAGE TO MY YOUNGER SELF

What he actually felt when NVIDIA was founded was that there was an enormous amount he needed to know and did not. There was no YouTube and no YC teaching anyone how to start a company, so he went to a bookstore and bought a book called *How to Start a Company*. It was 500 pages, and he figured he would be out of business and out of money by the time he finished it, so there was no sense reading it.

What he remembers vividly is how frightened he was of raising money, because he was about to talk to people whose questions he could not answer. And that was true. He says he barely knows how to answer their questions even today, and what he learned is that none of that matters. You will always have things you do not know, and the world changes every day.

So: this is important, I have to go learn it, I have to do something about it, and I should get to it as fast as I can. *How hard can it be?* In truth it is far harder than you think, but that is not where you want your mind. Do not imagine how hard it will be and let it turn into anxiety and inaction. Let the suffering arrive a little at a time.

Believe in your ability to learn, because learning is the single greatest superpower. And resilience is the single most important thing: you do not have to overcome life in one day, you only have to get through today, then work toward tomorrow. Stick with it long enough and NVIDIA happens.