

Y COMBINATOR

Startup School Paris

Notes on the talks

James Hawkins · Paul Copplestone · Stanislas Polu
Elliot Andre & Matt Rouif · Olivier Pomel
Alexandre LeBrun · YC Partners



The seven sessions, in the order they were held

One chapter per session, in the order the sessions took place. 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.

Recorded in Paris, 22 to 25 July 2026.

Source playlist: youtube.com/playlist?list=PLM9nPjgxyFBs

Contents

How to read these notes	3
1 James Hawkins: Why Ambitious Ideas Are Easier To Sell	4
1.1 From telling you the numbers to fixing them	4
1.1.1 What happens to the people doing that job	5
1.2 The gap he thinks nobody is working on	5
1.2.1 Where the pivot came from	5
1.3 Two CEOs who swapped jobs	6
1.4 The case for more ambition	6
1.4.1 On imposter syndrome, specifically the European kind	6
1.4.2 Why the maths favours ambition	7
1.5 Marketing as a product	7
1.6 Five pivots, and how they decided	8
2 Paul Copplestone: How Supabase Grew	9
2.1 Two startups before this one	9
2.2 Why Postgres, and why open source	10
2.2.1 Giving everything away	10
2.3 Time to value as the one metric	10
2.4 When agents became the users	11
2.4.1 Growth that broke the usual pattern	11
2.4.2 What that does to the product	11
2.5 Money, and what it does not fix	12
2.6 Fully remote, on purpose	12
3 Stanislas Polu: Building Alongside the Labs	14
3.1 Why he left a frontier lab	14
3.2 Where he was right and where he was wrong	15
3.3 Competing next to giants	15
3.3.1 The argument for not being tied to one model	15
3.3.2 What happens when a lab enters your vertical	16
3.4 Raising deliberately less	16
3.4.1 Building in France instead of San Francisco	16
3.5 Pricing, and where the margin sits	17
4 Elliot Andre & Matt Rouif: Training Yourself To Dream Bigger	18
4.1 The company, briefly	18
4.2 What changed their sense of scale	19
4.2.1 It happens continuously, not in one moment	19
4.2.2 Why they think European founders avoid it	19
4.3 A dissent on ambition	19
4.4 Tactics for aiming higher	20
4.4.1 Hiring for it, and screening for it	21

4.4.2	Keeping it up once you are large	21
5	Olivier Pomel: Monitoring The Entire Cloud	23
5.1	He opens by pointing out he was rejected	23
5.2	Twenty-six years of happy accidents	23
5.2.1	The bet, and the accidental tailwind	24
5.2.2	The naming mistake, and the fix	24
5.3	Fundraising while being in the wrong city	24
5.3.1	Not writing down the culture	25
5.4	Staying close to the ground	25
5.5	Twenty-five products, and how they choose	25
5.5.1	What being public actually changes	26
5.6	Telling the engineering team they will stop writing code	26
5.7	On the co-founder relationship	27
6	Alexandre LeBrun: What Big Tech Missed	28
6.1	Four companies, all of them early	28
6.1.1	The acquisition email he deleted	29
6.2	What a world model is, and why he is building one	29
6.2.1	What it is for	29
6.3	Why a big company could not do it	30
6.3.1	An ambition anecdote from inside	30
6.4	The largest seed round in Europe	30
6.4.1	Running researchers, and choosing cities	31
7	YC Partners: What Actually Makes A Startup Durable	32
7.1	A note on this chapter	32
7.2	The cost of intelligence	32
7.3	Where is the hard bit?	33
7.3.1	The same idea, applied to capital	33
7.4	Launch early, and why smart people find that hard	33
7.5	Co-founders, and what AI does not change	34
7.5.1	How to pick one	34
7.6	What not to delegate	35
7.7	On pivoting	35
7.8	Shorter answers worth keeping	36

How to read these notes

Six founders and a closing partner panel, recorded over four days. Unlike a conference program assembled around a theme, these were separate conversations, which makes the overlaps worth noticing. Four things recur:

1. **The ambition ceiling is cultural, and it is treatable.** This is the spine of the whole event. Photroom describe being trained into it, and give exercises for it. Hawkins argues ambition makes hiring, attention and fundraising *easier* rather than harder. LeBrun says Europe does not take enough risk. Polu criticizes his own caution as being a bit too French while defending the decision behind it. Nobody treats aiming high as a personality trait you either have or lack; all of them treat it as a habit you install by standing next to people who already have it.
2. **Ask where the hard bit is.** The partner panel states it directly: writing software used to be the hard part and no longer is, so something else has to be hard or you have no durability. The founders answer it in their own terms. Copplestone points at operating rather than building. Polu points at network effects, because scaffolding around a weak model stops being valuable once the model is strong. LeBrun points at problems a large company cannot take the risk on.
3. **Narrow the problem, not the vision.** Photroom focused tenfold twice and grew tenfold twice, and warn against confusing depth with a lack of ambition. LeBrun names trying to do everything for everyone as his own biggest mistake, and prescribes a narrow tunnel with a large vision. Even the panel's advice on becoming AI-native is to pick one deliberately narrow process rather than attempting everything.
4. **Stay in contact with the raw material.** Pomel reads the actual support tickets and survey comments rather than the summaries, because summaries arrive upward already beautiful. Hawkins took three trains and a bus to meet a customer worth a few hundred a month, and only learned what he needed because he had not phoned it in. The panel's answer to what they would automate last is talking to customers. Copplestone polls the community constantly.

A note on the final chapter: the closing session was a four-person partner panel, and the automatic captions carry no speaker labels. Rather than guess who said what, that chapter reports the substance without attributing it to individuals.

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.
- **Parting advice.** The closing thought each speaker left the room with. Where a talk trailed off on a prediction rather than advice, this box collects the closest thing the speaker did say, drawn from earlier in the same conversation.

CHAPTER 1

James Hawkins: Why Ambitious Ideas Are Easier To Sell



"Why Ambitious Startup Ideas Are Actually Easier To Sell"

James Hawkins Co-CEO, PostHog. Interviewed by a YC partner, 22 July 2026.

Video: <https://www.youtube.com/watch?v=ALJQHSgCI2E>

1.1 From telling you the numbers to fixing them

PostHog launched during its YC batch as an open-source product analytics tool, after pivoting several times. It now describes itself as self-driving software, and the reasoning behind the change is a clean piece of first-principles thinking.

They decided it was fundamentally better to solve someone's job for them than to give them a rough idea of what matters. An analytics company tells you your graphs are going up or down. They wanted to be the company that makes the graphs go up.

In practice that means pulling from every kind of data they hold: support tickets, logs and errors, session recordings, what people are saying internally, even video of a user struggling with the product. Then shipping pull requests that fix the simpler engineering problems it finds.

TRY THIS

His heuristic for finding startup ideas, which he applies to his own roadmap: look for jobs that are *already happening* inside a company. Because the work is already being done, you can be fairly confident there is demand if you can accelerate it and do it to a higher standard. That is a different search than looking for a piece of software to replace.

He applies it to product management specifically. A computer can do a superhuman job of absorbing everything happening across a customer base: every sales call, every internal meeting, every email, plus all the actual usage behavior. So he sees no reason you cannot build something better than a product manager for that particular part of the job.

1.1.1 What happens to the people doing that job

He is careful here, and says he does not mean it as a nicety. Their support product now has around 1,100 companies using it, and because they hold the data, they can resolve a lot of tickets well. The support team's role shifts to supporting the agents: writing the rulebook, iterating on it, improving it. His phrase for it is becoming a philosopher for support. Product management shifts the same way, toward configuring the software and setting the rules and policies.

1.2 The gap he thinks nobody is working on

Asked how you avoid ending up with a product that has a thousand tiny features that do not add up to anything, his answer is that this is exactly the weak point.

THE ONE THING

Today's tools do not capture *intent*. Ask a coding agent for a to-do list app and it will ask which database you want. Ask a human product engineer and they will ask why you want it, who it is for, whether it should be simpler or more powerful, what is special about it. As an industry, he says, almost nobody is working on capturing the product side of product engineering, and there is a lot of value sitting there.

They are building toward it by trying to capture intent as the engineer works, so the system iterates with you to understand why the product exists rather than starting from the technical requirements, which gives it something to aim at.

1.2.1 Where the pivot came from

Not from inside the company. He was on the longest holiday he had taken, reading about why some of OpenAI's co-founders joined in the first place. The reasoning that struck him was almost blunt: a brain is a physical object, so you can build one. Insanely difficult, but an engineering problem.

So he made a list of the differences between a language model and a human brain, and concluded there was a long way still to go. That thought became the company's direction.

THE ONE THING

His related observation, which doubles as advice: the models are already well past the point where they can deliver a lot of value. What constrains the value now is gluing the model to the use case. He says that is a good place to look for startup ideas, and much of their product thinking comes from there.

1.3 Two CEOs who swapped jobs

He runs PostHog with his co-founder Tim, and they call themselves co-CEOs. He recommends it at early stage.

They started conventionally, with him as CEO and Tim as CTO because Tim was the more technical of the two. After a while they noticed the assignment was wrong: he was better at pushing on product, and Tim was actually better at running sales and treated it more as a system. So they swapped.

TRY THIS

He found the swap itself generated energy. Whatever you are *not* doing is the thing you start worrying about, so he was anxious about not controlling product while Tim was anxious about not controlling sales. Swapping felt like a new job to both of them and they each went harder. His framing: do not be a lemming. People have a strong tendency to follow the path society expects. Pick the job that addresses the actual bottleneck to the company succeeding.

The swap is also how they executed the AI pivot without getting in each other's way. Tim took everything else, including finance and hiring during a year when they more than doubled headcount, and James spent his time with engineers building. He did no one-to-ones at all during that period.

1.4 The case for more ambition

This is the substance of the talk. PostHog started out almost apologetic, looking for a small place in the market, hiding from larger companies and frightened of competition. Over time they learned to embrace it, then realized they could build something better than the companies they were competing with. They went from one product, to wanting twenty, to building twenty, to now saying the individual products barely matter because an agent should be using all of them.

His argument is that ambition makes the mechanics of a startup *easier*, not harder:

- **Hiring.** Genuinely strong people are more likely to want to work on something inspiring. He treats this as obvious.
- **Getting attention.** If you are more ambitious, you are more remarkable, and remarkable things get remarked upon, which is word-of-mouth growth. As software gets easier to build, everything saturates and getting attention gets harder and harder for startups. A narrow point solution can still succeed, but you have an easier time with something more compelling.
- **Fundraising.** Easier to raise for very large ideas, and more polarized. A weak investor is less likely to work with you; a genuinely strong one is more likely.

WATCH OUT

You have to hold two things at once: the most ambitious version of the idea you can state with a straight face, and the next practical step you can take this week. He grants that holding both hurts your head, and says you need both anyway.

And the counterintuitive half: the more ambitious the idea, the *more* important it is to ship, because you are more likely to get it wrong. If you disappear into the wilderness for three years to return with something perfect, you will not. So the first internal question on anything innovative is when can we launch, and can we launch sooner.

1.4.1 On imposter syndrome, specifically the European kind

Before YC, they looked at people in YC as superstars and had strong imposter syndrome, which he thinks you are more likely to have the more European you are. Getting in was, in his words, underwhelming in

a really good way: everyone was far earlier and far more chaotic than expected. And from that starting point, they watched companies take off.

The same thing kept happening as they grew. Now, meeting people running enormously valuable companies, their approach is to walk in and treat them as peers and see through some of the veneer. His example is that a company like Anthropic looks undefeatable, and the reality is it is run by humans who are not necessarily smarter than anyone in the room.

AGAINST THE GRAIN

What he says YC actually gave him was not the thing he expected. He assumed the value was a network he could sell into to get revenue, then raise. What it actually did was normalize aiming very high and set a high bar. He calls it a fluffy and unsatisfying answer for an engineer, and says honestly it made them about a hundred times more confident than they were.

He frames the cultural difference as a question. In the US the question is: what if it all goes right? What would be possible if everything worked? In Europe the questions are: what if it fails, what will my friends and family think, and is it compliant yet.

1.4.2 Why the maths favours ambition

His sharpest point, and the one the talk opens on. You are default screwed as a startup, so it is all upside. You cannot win on downside.

The investors worth working with are focused on what happens if it all works out, because they know almost none of your upside has been realized yet. There is little point worrying about how it goes wrong. The partner interviewing him adds the numbers from the other side: YC has invested in around 7,000 companies, and he would guess about 90% of total returns come from five of them. It genuinely does not matter if the rest fail, provided a handful reach enormous scale.

THE ONE THING

So when you fundraise, pitch that way. Pitching that you will be careful with the money is pitching against the model the investor actually runs. You do still have to not die, but that is the shape of the bet they are making.

1.5 Marketing as a product

PostHog's developer brand, hedgehogs and buses and all, was intentional at the start.

His reasoning: if you go into a busy market, and he thinks entering busy markets is underrated, then standing out is your *only* marketing problem. You are not creating demand and you are not educating anyone about what the product does. You only need to get across that you stand for something more interesting.

When they launched, every competitor had a blue website, used long words, and had pricing pages with no pricing on them. Part of why they started the company was being annoyed at that as technical buyers: they wanted the products but could not easily get at them, and just wanted to sign up and enter a card. So they targeted engineers rather than the other roles their competitors sold to.

THE ONE THING

He is blunt that this mattered more than the product for a long time. Their product was terrible compared to the competition for a good while. But people wanted them to win on the internet, and that carried them until the product caught up.

TRY THIS

Treat go-to-market as a product. He points out that people think hard about product (who is this for, what should it do, how do I differentiate) and then copy and paste their go-to-market. Instead ask of your website: what are the use cases, who are we building it for, what profiles of person show up. Once they had done enough of that, the marketing team could be left to do what it wanted within a shared philosophy.

He extends the fit test to hiring: you are unlikely to be hired at PostHog if you wear a suit to the interview, and if you very much want to be a manager you should work somewhere else. They interview for people who will genuinely find the work entertaining.

On whether enjoying it matters: he thinks after product-market fit it is one of the most important things preventing founders burning out over the next ten or fifteen years. The example offered is Patrick Collison saying his mentality is that he wants to be running Stripe for another forty years, so his role has to reflect that.

1.6 Five pivots, and how they decided

Their goals during the pivots were not revenue targets. They were learning targets, at the resolution of *can we get one single user to run this in production*, and they iterated fast toward those.

THE ONE THING

His model of luck: your chance of hitting product-market fit is a function of how many shots you take and the quality of each shot. And there are diminishing returns on the quality of a single shot. He reckons you could spend ten years on their first idea and still fail. They gave it four to six weeks.

What they did not do was phone it in. They committed to each idea almost blindly and hammered it. For their fourth idea, they took two or three trains and a bus to meet their first paying customer in person, for a contract worth a few hundred a month, and probably spent about that getting there.

That mattered later. When they tried to raise the price, they could not move it at all, and that told them the customer did not really value it. Had they not committed properly, the customer could have been distracted or half-listening, and the signal would have been worthless. So they moved on when they kept hitting the same wall and either ran out of ideas for getting past it or decided it did not feel right.

He is honest about the bias: they were probably too quick to abandon things, and he thinks that was because they were bad product people at first and needed a few attempts to build that judgement.

PARTING ADVICE

Do not be a lemming, and do not let the downside occupy your thinking. You are default screwed anyway, so the only question worth asking is what happens if it all works. Aim at the version of the idea that makes strong people want to join and makes strangers want to talk about you, then break it into pieces small enough to ship this week.

CHAPTER 2

Paul Copplestone: How Supabase Grew



"How Supabase Became One Of The Fastest Growing DevTool Companies In The World"

Paul Copplestone CEO, Supabase. Interviewed by a YC partner, 23 July 2026.

Video: <https://www.youtube.com/watch?v=sG5aB79TE44>

2.1 Two startups before this one

Supabase is his third venture-backed company. His father was an entrepreneur, so it felt natural; he did one job at a corporate consultancy, hated it, and has done nothing but startups since.

The first, where he was CTO rather than CEO, never found product-market fit, and he left after a couple of years because he and the CEO disagreed about where to take it. For the second, his now co-founder asked him for two years as CTO to help launch it; he agreed, and incubated Supabase in the background. That company still exists and is profitable, but it was never going to be the thing he worked on for the rest of his life.

There is a small correction worth noting from the interview. He suggested running out of money is the top reason startups fail, and the partner interviewing him disagreed based on YC's own data: founders giving up, co-founder conflict, and not finding product-market fit come first, and most companies that shut down still have money in the bank.

2.2 Why Postgres, and why open source

The immediate trigger was practical. He was using both Postgres and Firebase at his previous company, hit scaling limits on the Firebase side, and wrote an open-source tool to migrate across. He put it on Hacker News and it got traction. Supabase launched three or four months later.

His reasoning for betting on a database that was already decades old:

- He read Hacker News several times a day, including the launches, which gives you a sense of what is trending. Postgres was trending without being the most popular database at the time.
- Building a database requires a good reputation, because databases are extremely hard to build. Postgres already had that reputation built in.
- **The structural argument.** Nobody owns Postgres, so everybody contributes to it. Because it is the most popular, every large cloud provider has to offer the best Postgres they can, which improves Postgres further. He liked that flywheel and leaned into it.

On open source, he is deliberately unstrategic: he likes open source and likes building communities, and both he and his co-founder wanted it, so there was not much discussion.

THE ONE THING

The positioning did the work, not the technology. He first wrote a real-time engine on top of Postgres and thought that was the thing. It did not take off. They changed the tagline to describe themselves as the open-source alternative to Firebase, and from that point it really took off.

2.2.1 Giving everything away

They release everything. The only thing not public is the code for hosting an enormous number of databases, which he says is tangled up with billing logic and would be a security problem to publish, not a deliberate paywall.

Asked what stops a large cloud provider from simply hosting it themselves, his answer has two parts. First, they counter-positioned: the big providers already sell hosted Postgres, so there is not much for them to take. Second, Supabase is a stack of tools working together, which is a much harder thing to operate than a single component.

On the community complaining that they are not open enough, or too expensive: there will always be that crowd and you cannot please everyone. His refinement is to listen anyway, and then still make your own judgement about whether what they are asking makes sense.

2.3 Time to value as the one metric

Nobody in their own batch trusted them with their database. Roughly one company used them. Today they say over 60% of each batch does.

What closed that gap was not marketing. People liked the idea immediately, because they already liked Firebase and wanted the same experience with Postgres and open source. What was missing was trust that Supabase could host it, and that accumulated as larger startups built on them.

TRY THIS

The metric they picked at the very start was *time to value*. He timed how long it took to launch a database on a major cloud provider and insert a single row: around eight and a half minutes. They decided theirs had to be under a minute, and tracked it from there. It is now a few seconds.

He notes they never advertised the number. It just becomes part of the experience.

His update to that idea is worth the space, because it changes what the metric even means. When an agent is doing the work, time stops being the right unit. If you come back and the job is not done and you have to prompt it again, that is the loss of value. So the goal becomes getting to the result with as little input as possible, ideally on the first try.

Revenue growth was, by his description, slow: about a million in the first year, then seven and a half the next. The partner pushes back that most people would take that, and he concedes it only looks slow by current standards. There is a useful exchange where he mentions companies reaching very large revenue in a month, and the partner cautions that the press only covers the extreme outliers.

2.4 When agents became the users

In December of one year they noticed they were being hammered by traffic from a couple of companies launching applications and databases on their behalf. Fast forward, and they can measure that around 60% of databases are now launched by an agent, with the real figure probably closer to 90%, in the millions per month.

WATCH OUT

He is candid that this was not obvious at the time, and that hindsight makes it look inevitable. When they looked at the tools driving that traffic, the applications being produced were not very good, and there was real internal debate about whether to support the use case at all. Some people argued they should build their own version of those products to own the distribution instead.

What they built instead was a product for platforms: the ability to launch enormous numbers of databases, which over fifty companies now build on. And it turned out to converge with what large enterprises want, especially innovation groups: launch many databases internally, see which are insecure, shut down ones growing too fast, manage the whole set. Two roadmaps they could merge.

2.4.1 Growth that broke the usual pattern

He describes the shape of it as the genuinely surprising part. They do not spend on marketing. His line is that if he can give away a free database, he would rather give that to developers than spend the money on marketing.

Normally at their size you would put money into marketing, which means the intent of incoming users drops, so conversion and activation rates fall even as user numbers rise. Instead every chart went up at once: more users, and rising conversion, and rising activation. He attributes the acceleration to coding agents, and says developers were building bigger things, more successfully, and getting to a product sooner.

2.4.2 What that does to the product

Three phases, in his telling. First a dashboard where you clicked around to add things, because discovery mattered. Then, once language models arrived, they kept the dashboard but stripped out most panels, leaving a code editor and a chat interface. Now agents do not care about either, and want command-line tools and a way to connect.

THE ONE THING

Where he thinks it goes: everything you could do in the dashboard should also exist in code, so your database is represented declaratively in your project. Once it is in code it is in version control, and once it is in version control you get branching, which means many environments. So the whole stack has to become branchable, and the agent has to be good at both spinning up a database from a stated intent and then developing on top of it as it gets more complicated.

Asked how a dev tools founder starting today should make sure agents find and use them, he is honest that he does not know. His reasoning is a warning: if agents only ever pick three options in a category, then there are three winners and everyone else is competing on something else. So work out what that something else is. It is probably the same as it has always been: make it very easy to build with you, earn word of mouth, and accept that becoming a default is rare and partly luck. He does not like betting on luck.

He allows that being open source helped here, because their community discussion and public code were absorbed into the models.

WATCH OUT

He voices the pessimistic version himself rather than dodging it: if defaults are all that matter, maybe only the largest cloud provider wins and everyone else may as well go home. His answer is that you do not actually know, so you stay in the arena. And it may turn out differently from the one-default hypothesis. He suspects people will still choose Supabase over the alternative partly on feel, and if a competitor matches the experience, that advantage narrows.

2.5 Money, and what it does not fix

They raised a very large round at a valuation over ten billion. Asked how he stays grounded, the answer is unglamorous: the money should not change how you operate. You have jobs to be done and the money is in service of them. If your reaction to raising is to go and do something else, you are probably going about it wrong.

AGAINST THE GRAIN

His clearest point, and he frames it as good news for early-stage companies. What keeps you grounded is knowing absolutely that money will not solve your problems.

The consequence is that Supabase is not limited by cash in the bank, which means a competitor can compete on many other vectors, exactly as Supabase did. By rights they should not exist if the largest cloud provider is so good, and that provider has far more money. The cash is only in service of the things that would let you win without it.

One structural oddity he flags about investors: a hedge fund invested early, which inverts the normal dynamic. Usually early investors lose interest as you succeed, because they have made their return. Your most recent investors are the ones intensely interested in what you will return for them.

2.6 Fully remote, on purpose

Around 360 people across more than 60 countries, no offices, and he would do it again.

Two conditions. You cannot half-do it, and the hybrid model where some are distributed and some are in an office does not work. And everything happens in writing, online.

TRY THIS

His reason for being more confident in this now than before is one most remote-company arguments miss. Because every decision at Supabase has been written down since the beginning, they can ask an agent what was decided six years ago and it can find it. Most companies have agents that know the last six months. Theirs know the whole history, which means the company's operating principles can be reconstructed from the record.

PARTING ADVICE

The bet he named for the next several years is a useful place to point your own attention. Almost all the current effort goes into the build stage: prompting for an application and getting one. Far less goes into the much harder problem of continuing to *operate* the thing once it exists, which for a database means it should look after itself rather than waking you up.

He generalizes it into advice. A lot of what is being launched right now is easy to replicate, because it is in the boom stage. Operating is one of the harder areas. Focus on hard problems.

CHAPTER 3

Stanislas Polu: Building Alongside the Labs



"The Model-Agnostic AI Platform Betting That No Single Lab Will Win"

Stanislas Polu Co-Founder, Dust. Interviewed by a YC partner, 23 July 2026.

Video: <https://www.youtube.com/watch?v=DbBnd9PYob4>

3.1 Why he left a frontier lab

Five years at Stripe, then three at OpenAI working on language models and maths, having joined at a point where you could do research as an engineer without being a trained researcher. He describes those three years as the PhD he never did.

He mentions, unprompted, that the stock options he gave up by leaving were worth more than the whole of Dust as of six months later, and says he does not regret it at all.

His reason for leaving is a good description of what research actually feels like. You scratch at a surface for weeks or months, find something interesting, and the high lasts about five seconds. Then you start scratching again. He enjoyed it and wanted to go back to building a product.

THE ONE THING

His framing of what changed about company building: for twenty years the technical ground under you was stable enough that you could see what you would be building years out. That is no longer true. The definition of the company is a moving target because the substrate moves quickly.

The founding thesis for Dust was simple: the models were ready to change how people work, and they wanted to focus on applying them to the workplace. He notes that in late 2022 and early 2023, when they were raising, this was almost considered a niche.

3.2 Where he was right and where he was wrong

He answers this directly, which makes it one of the more useful passages in the talk.

Right: that work would be enormously disrupted. But the timing was slower than the conviction implied. For roughly three years the shape of work stayed much the same. There were vertical products for particular tasks, and a period where everything was about asking questions and getting answers back, but the shape of the work itself did not change. Only in the last few months, he says, has the way he works as an engineer changed radically, and he expects other functions to follow.

Wrong: he believed in an earlier plateau. His argument at the time was that even if the technology stopped improving, there would still be decades of deployment and adoption work left. That remains true, but the plateau simply has not arrived.

TRY THIS

Asked what work looks like in two or three years, he gives an analogy rather than a prediction. If your great-grandparents saw you sitting in a chair looking at a screen, they would not recognize it as work at all, because theirs was physical. He is fairly convinced that in two or three years, today's version of you would look at that future and not recognize it as work either.

3.3 Competing next to giants

The interviewer's framing is building in the shadow of frontier labs; he pushes back gently that alongside is fairer than in the shadow. He is direct that Dust is tiny by comparison and that this is fine.

- **The benefit.** The labs educate the market very efficiently.
- **The difficulty.** They went horizontal early, at a time when the standard advice was to go vertical. Vertical go-to-market is far simpler, because you know exactly who to talk to and can repeat it. A horizontal platform is powerful but the go-to-market is finicky, because the response is often agreement followed by not knowing what to use it for.

His read on the current landscape is that everything is converging toward the same shape: a new kind of productivity suite for people and agents working together. Products used to have real depth in a narrow area; now there is less depth and much more coverage. So the differentiation comes from the sensibilities you bring to building that surface. Theirs is collaboration and multiple people working with AI together.

3.3.1 The argument for not being tied to one model

This is his sharpest analogy, and the video is titled after it. Buying your product and your tokens from the same place is like building a factory where you buy the machines from the energy provider, and the plug only fits that one provider. If your energy supply becomes a problem, you cannot switch. He thinks it makes obvious sense for the product layer to be free to source intelligence from anywhere.

THE ONE THING

The subtlety is that being model-agnostic does not mean spreading usage evenly. At any given moment most of their usage goes to one provider, because at that moment that provider is the best for the work. The point is that which provider that is keeps changing, so he sees no future in which they would give up the ability to switch.

3.3.2 What happens when a lab enters your vertical

Asked what a founder should do if a large lab moves into their space, he separates the two things a vertical product was providing. The go-to-market advantage is real and remains. But the second source of value, the scaffolding you built around a model because the model was not good enough on its own, disappears as the models improve.

AGAINST THE GRAIN

His conclusion is that the advice loops back roughly fifteen years. If intelligence is close to commoditised, then what is defensible in a vertical is whatever network effect you can find in it: value that comes from multiple users interacting on the same platform. If you are simply delivering intelligence to one customer and then another, with no network effect, defensibility is changing underneath you.

3.4 Raising deliberately less

The partner introduces him as one of the more thoughtful founders on company building and on fundraising specifically, noting he could have raised at a very high valuation as a second-time founder out of Stripe and OpenAI, and chose not to.

WATCH OUT

His reason is a specific failure mode. If you raise too much and do not realize the potential of what you raised, you end up cornered, where the only remaining outcome is a down round. He has watched friends get burned this way. He has never done it himself and describes it as genuinely bad in many respects.

Their rule was almost a slogan: no expensive compute before product-market fit. They raised their seed at a time when it was fashionable to raise very large sums for training, and took a small fraction of that, because building at the product layer meant they would not have been able to use the money. Better to raise less at a lower valuation, be aggressive about building, and raise again when ready.

Then he argues against himself, which is the valuable half. The negative reading of that choice is that it is a bit too French. He is admiring of a European lab whose founder knew nothing about company building, grew into the role, and built something worth a great deal, and says there is something in that of kicking the ball very aggressively. He treats it as a grey zone where both instincts matter and you have to find your own point on it.

3.4.1 Building in France instead of San Francisco

He knew San Francisco well, having built and hired there, and came back to France anyway.

He is straightforward that everything would have been easier from a company-building standpoint in the US, and that easier does not mean right. It mattered to them personally, with an element of wanting sovereignty over what they were building, and he treats it as an additional constraint you should be clear-eyed about.

THE ONE THING

His observation about how much location actually matters: when things are going very well, nobody cares. Everyone says the founders need to be in San Francisco, and the same investors show no interest in where the founders of obviously successful European companies are based. It is friction you accept for something you value, and it is a trade-off rather than a mistake.

He also notes their funding experience directly: a very easy seed, a very easy Series A, and a Series B that was the first time they raised for real, which was complicated and hard as it should be. He attributes part of that to building from France at a moment when that was less fashionable from an American perspective, and part to the energy in the market being absorbed by the labs themselves.

3.5 Pricing, and where the margin sits

The most concrete operational passage in the talk.

They used a per-seat price for three years, deliberately, because they wanted to encourage usage and value creation rather than metering it. Over the following six months usage exploded: agent loops got longer and models used more tokens everywhere. Their margins compressed, and they moved to credit-based pricing.

TRY THIS

His flat recommendation: if you are building at the product layer, there is no way around usage-based pricing, because people want to use more and will use as much as you give them. With a flat price the margins compress and keep compressing while models improve and usage grows.

He offers two counterweights to the worry that the labs capture all the margin. There are precedents for selling a well-crafted product on top of infrastructure whose owner also offers a competing product for free, and a genuinely differentiated product can still capture a fair margin. And open-source models catching up should apply pressure.

He then does a back-of-envelope estimate worth understanding, because the method is reusable. Latency is a proxy for the cost of serving a model, since for a given infrastructure it tells you something about the model's size and energy use. If an open model and a frontier model run at roughly the same latency, you can compare what you pay for each. He puts the gap at close to an order of magnitude and infers frontier serving margins in the region of 70 to 80%. His hope is that open-source pressure recalibrates that, which would give product companies more room even if the labs still capture a lot.

PARTING ADVICE

Find the thing you actually want to fix. He calls this typical-sounding and insists it is true, because building a company is miserable on a daily basis: a few highs and a lot of lows. Without something that makes you able to wake up and say this is why I am doing it, it is not sustainable. If you do not have that yet, find it. It is fine to reinvent it while you build. But find the thing that keeps you pointed at an objective through all the glass eating the job involves.

CHAPTER 4

Elliot Andre & Matt Rouif: Training Yourself To Dream Bigger



“How Potoroom Trained Themselves To Dream Bigger”

Elliot Andre & Matt Rouif Co-founders, Potoroom. Interviewed by a YC partner, 24 July 2026.

Video: <https://www.youtube.com/watch?v=RgYCO87ghRY>

4.1 The company, briefly

Potoroom gives e-commerce businesses AI visuals to grow and sell. By their own numbers: around 300 million downloads, 20 million active users, and the largest e-commerce companies in the world using it to edit the images they sell with. The figure one of them says he is proudest of is being described as the biggest YC company headquartered in Europe.

Asked whether they always thought it could get that big, the answer is honest: you are always somewhat surprised by how big it becomes. Coming out of YC they had far more ambition and did have today’s scale in mind, and it was still a large hypothesis. They knew the path and were surprised when it happened.

4.2 What changed their sense of scale

This is the substance of the session, and both founders keep returning to it: ambition is something they had to be trained into, not something they arrived with.

Before YC, the goal was a vague dream of building a very large company, with no path and no playbook. It was something to look at from a distance. What changed it was proximity. The people you talk to are already at the scale you are aiming for, so a casual conversation with someone who has built an enormous company recalibrates what the scale even means. It looks closer.

THE ONE THING

Their description of the before and after is precise. Before, you tell your friends you want a billion-dollar company and it sounds like a joke. After, the number you say out loud has an extra digit on it, because the scale at which you can *realistically* dream has changed.

The interviewer adds a personal version: knowing the question “how big could it get?” was coming, they had to rehearse many times to be able to say “obviously a billion-dollar company” in a way that sounded natural. It was hard. American founders in the same room said it without effort.

4.2.1 It happens continuously, not in one moment

Asked whether there was a single shift, both say it accumulates. One concrete moment they name: their demo day was the first remote one, and they were the fastest-growing company in the batch, having gone from nothing to a million during it. The arithmetic did the work. If some small percentage of a batch becomes enormously valuable, and you are at the top of the batch, then something ought to happen. The maths gave them no reason it could not be them.

They stress that it did not stop after the program. Beyond the three months you continue to meet partners and other founders, and simply talking to founders in a meeting room resets your sense of pace, because they are growing faster than you and you start asking how to get there.

Their sharpest anecdote is from a retreat. At dinner, two founders from two different companies were comparing how many satellites they each had in orbit. Six, versus eight. Both companies were twenty or thirty people. That becomes the benchmark. The interviewer adds the detail they left out: one of those founders was nineteen.

4.2.2 Why they think European founders avoid it

They put it down to culture. Where they grew up, describing a big idea invites someone to laugh or to look for what could go wrong. Do the same thing in the US and the default response is encouragement, which pushes you toward trying.

The second half is the treatment of failure. In some countries failing is simply bad; in others you are assumed to have learned a great deal and are more employable for it. Their conclusion is that this is precisely why you should surround yourself with people from different cultures.

4.3 A dissent on ambition

One of them offers what he calls a contrarian view, and it is the most useful part of the session because it puts a condition on everything else.

He asks the room to choose. Option A: your company works decently, money is a bit of a struggle, growth is slower than you want, but you are working on the thing you started it for and it matches what you believe. Option B: it grows very fast, it is not quite what you had in mind, and you have all the pains that come with growth.

His point is that you have to answer that honestly first, because if you are living your passion, you may not need ambition at all.

WATCH OUT

But if you take venture funding you have signed up for growth. Your investors and your employees will all be asking for it. So if you do not have the ambition to make it as large as possible, you will be miserable in the model you have chosen.

The interviewer frames it as a door that only opens one way. Very few decisions in a company's life are irreversible, and taking venture money is one of them, because from that moment you have committed to more growth every year, and that dictates a great deal about how you operate.

There is a striking example of the cost of the ambition gap, told as a true story. They made an offer to an engineer in Paris and an equally good offer to an engineer in the US. The French engineer declined, saying the team was too strong and he expected to fail. The American engineer accepted, and also asked for two levels more seniority than was offered. The result is that the engineer with the skills did not join, and the one who did was set up to fail and eventually left. Everyone lost. Their read is that the right answer is somewhere in the middle.

4.4 Tactics for aiming higher

The most directly usable part of the talk. Several of these are exercises rather than attitudes.

TRY THIS

Ladder the goal deliberately. Work out what you could decently aim for, then imagine you have no constraints at all and ask what the level just above that is. Then ten times that. Then the version where you have total control. The point is to force the number upward, because aiming lower is what comes naturally.

TRY THIS

Say the number out loud and work backwards. At around a few million in revenue they held an offsite with ten people and set a target of twenty million. Everyone in the room said it was not possible. They hit it a year later. His read is that if he had said eight million, they would have reached eight million.

The interviewer offers the supporting observation: look at the distribution of marathon finishing times and there are spikes just under the round numbers, because people set those targets and then arrange themselves around them. Worst case you land below the goal and still above where you would have been.

TRY THIS

Add a zero. Attributed in the session to Sam Altman at a recent YC talk: take whatever success metric you have in mind and add a zero, then do the thought exercise of how you would get there. The exercise alone is useful, even if you never commit to the number.

TRY THIS

Explain how you get to a hundred million. A YC group exercise, done a few weeks into the batch: explain to the group how your business leads to a hundred million in revenue. Doing it when you are extremely early forces you to think about how big the market can be and what the most ambitious version of what you are building actually is.

TRY THIS

Halve the resources. Ask the team what they would do to still achieve a goal with half the people or budget. The constraint produces creative thinking and often a better outcome.

4.4.1 Hiring for it, and screening for it

Their filter is unusual and worth reporting exactly. Photoroom is global, across a large number of countries, and their view is that if you do not want to be global you will not be ambitious enough. So they speak English deliberately in the Paris office, including between the two co-founders in their own messages. It turned out to be a strong filter for people who want to build a global company. Their reasoning: you will not lead the world if you only speak French in the Paris office.

THE ONE THING

The other founder adds the more general version, which applies to any pair of people starting something. Ask each other the growth question early. He hears about people who started working together and only later discovered a disconnect, where one is there for the mission and would be content if it stayed small and the other wants to change the world. That mismatch alone is enough to slow a company and generate a lot of frustration.

He is explicit that wanting a stable job you do not worry about, with your weekends intact, is a perfectly fine thing to want. You just have to understand that the other path requires trade-offs and compromises, and is costly as well as rewarding.

4.4.2 Keeping it up once you are large

Two practices they named:

- **Rank projects by the honest upside.** Once you get bigger it is easy to forget the end goal, so ask of each project: if this works perfectly, how much revenue does it bring?
- **Look backwards at your own estimates.** If the company is not growing as expected, go back to a project that took three months and ask what the actual impact was versus what you expected, and which link in the decision chain went wrong.

TRY THIS

Always ask what the smallest version is. Roughly half of projects fail to have the impact you expected. Since trying more projects is how you get impact, the lever is reducing the time each one takes. So ask what the minimum version is that teaches you whether this will work. Otherwise someone spends three months on something that could now be learned in three days.

They close the section by referencing the previous speaker, who runs a very large company, and his lesson that you should always be learning how to go faster. Their point is that this applies to small startups and to much larger companies alike.

PARTING ADVICE

Two answers, one from each.

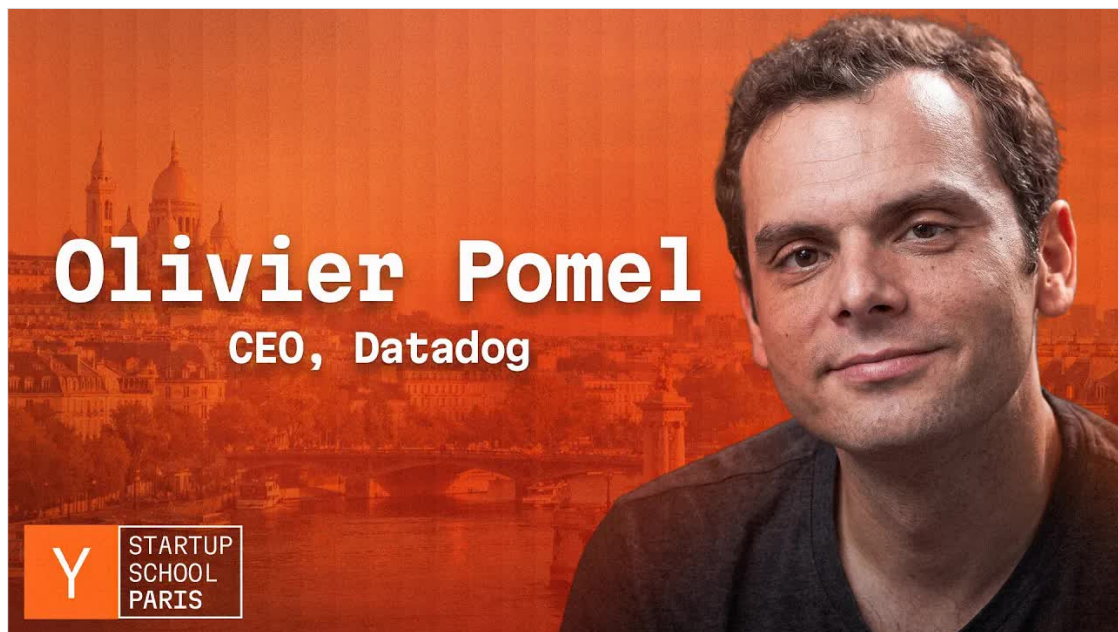
On focus. What helped most at the start was narrowing. They were doing video and photo, focused ten times harder on photo, and grew tenfold. Then focused on e-commerce photo and grew tenfold again. So do not confuse depth with a lack of ambition. Being deep is a very good way to achieve ambition.

On doubt. If you grew up somewhere that did not push you toward ambition, you will have to

push yourself, and you will have doubts regardless. The useful move is not to erase them but to set them aside: acknowledge they are there and then talk about something else. And anyone can build a large company. One of them was sitting in the audience at Startup School twelve years ago.

CHAPTER 5

Olivier Pomel: Monitoring The Entire Cloud



“How Two French Engineers In New York Built The Company That Monitors The Entire Cloud”

Olivier Pomel CEO, Datadog. Interviewed by a YC partner, 24 July 2026.

Video: <https://www.youtube.com/watch?v=vb1Gcn10enw>

5.1 He opens by pointing out he was rejected

Datadog did not get into Y Combinator. They applied, got an interview, and he still has the rejection email. He then got rejected a second time, in a sense: having agreed to speak at this event, he received an invitation to a YC dinner in Paris and had to be told he was not eligible because his company was never in the batch.

He tells it as a joke, and the partner draws the obvious moral for the audience: here is what being rejected can lead to.

5.2 Twenty-six years of happy accidents

He graduated in 1999 and did an end-of-study internship at a corporate research lab in upstate New York, working on internet protocols. His co-founder Alexis was a couple of years ahead of him at the same school and was already there. They met properly at that lab.

He thought he would stay in New York six months. He is still there, twenty-six or so years later, and describes that part as a happy accident rather than a plan.

What followed was formative in an unusual way. He caught a little of the dot-com boom and the entirety of the bust, working at startups rather than running one. One of those companies should have been remarkable and made a lot of mistakes he still remembers: spending too much on the wrong things, and not shipping product when it should have.

Afterwards, when nobody thought technology would be worth anything again, he and Alexis joined an educational software company that grew to around 800 people. He built the development team; Alexis built the operations team. That is where the idea came from, because they lived the problem: developers and operations people disliked each other and pointed fingers, which was strange given that the two of them were close friends and each ran one of the teams.

5.2.1 The bet, and the accidental tailwind

They started in 2010. At the time this was not called observability; it was called monitoring, and it was job-specific and reactive. Network monitoring was one product. Operations had tools developers would never touch, which left developers essentially blind to what happened in production.

Their premise was to bring those groups onto one platform. What they did not fully realize was that this premise was central to the growth of cloud computing, and they badly underestimated how large cloud would become.

THE ONE THING

His account of the market research is instructive. When they talked to established companies about cloud, the answer was that it was a toy they would never use. It turned out they all did. The reason Datadog ended up cloud-first is more mundane than a prediction: they talked to smaller and more modern companies, and those companies were already built on the cloud.

5.2.2 The naming mistake, and the fix

A genuinely useful lesson buried in the middle of the talk. They deliberately avoided calling the product monitoring, because they thought monitoring was a boring thing from the 1990s. They called it a data platform for development and operations teams to collaborate on.

WATCH OUT

People love the idea of a new thing, and have no idea why they should pay for it. Neither does their boss. Say instead that it is infrastructure monitoring, and everybody and their boss immediately understands: it is a category, it is real, we need it, let us buy it.

They renamed it, even though the product did not resemble the monitoring tools that came before.

5.3 Fundraising while being in the wrong city

It was not obvious at all that it would work. Their angel round was extremely painful and none of the smart investors would engage.

The two failure modes were geographic. Investors in New York were interested until they saw the competitive landscape, realized they did not understand any of those companies, and concluded the market was not for them. Investors on the west coast at that time only funded companies in their own region, and treated starting an infrastructure company anywhere else as a form of impairment. He says he could hear them start to speak more slowly and loudly at him.

He turned it into motivation, and says several of the things they did afterwards were deliberately different from what they saw west-coast companies doing.

5.3.1 Not writing down the culture

Their example of doing it differently. At the time every company had a list of values or principles on the wall. Datadog never wrote theirs down.

AGAINST THE GRAIN

His reasoning: if you need “don’t be evil” written on the wall, you probably should not work there. Culture flows from the top, so they exemplify it and then hire, promote and fire based on it. That still works at around 8,000 people.

He is pragmatic rather than dogmatic about it. Every year they ask whether this is the year they need to write it down, because thousands of new people have to be trained and told something. So far the answer has been no.

5.4 Staying close to the ground

The partner mentions that what impresses people who work with him is how involved he remains in small product decisions rather than spending his time on strategy documents. He is dismissive of the word strategy, and says when people use it in a sentence he mentally replaces it with something less polite, while allowing that sometimes it is meaningful.

His argument for staying in the product is about a specific function only he can perform. Plenty of people handle the product and talk to customers. But as you get further up the chain, ending with him, you are in a position to *edit*: to say this is too much, this is not enough, stop doing that, or I do not understand this part. And if he does not understand it, he asks how the customers are going to. He is careful to say he is not a required approval step, and that part of the job is knowing where to direct attention and where not to.

TRY THIS

The habit he recommends to anyone whose company is starting to scale: stay in contact with the raw material rather than the summaries. Actual support requests, actual sales call transcripts, actual customer feedback, actual employee survey comments. He still reads all of the employee survey comments; for everything else he samples.

The reason is that summaries arrive upward already beautiful. Every new product is going great. Then you read two customer complaints saying it does not work or is too expensive, and you have a question worth asking.

His method for asking it is deliberately light. He picks up one of those messages and simply replies with a question, saying he does not understand what is going on. Two things then happen. The person receiving it wonders why he is looking at that. And the management chain above them stops looking upward to work out what to manage upward, and starts looking downward to find out what is actually happening, because they might get asked.

THE ONE THING

He states the resulting principle plainly: your job is not to understand what your boss wants. Your job is to understand what is happening within your span of control, very precisely, and what is working and not working.

5.5 Twenty-five products, and how they choose

Because the product is platform-like, customers build around it and develop their own extensions and workflows. That shows them where customers think Datadog belongs, and much of what they build is

what they see customers doing.

The rest are deliberate bets, where nobody is asking for it but the world looks like it is heading that way. Those can have bigger impact because they can be genuinely new, and you are wrong more often because you made it up.

THE ONE THING

What has changed with AI is the timing. Things now move fast enough that you cannot wait for demand to materialise from your customer base. You have to get ahead of it, which means he and others will be wrong more often, and they need to be at ease with that.

5.5.1 What being public actually changes

Less than people expect, in his telling. As a private company you spend a lot of time with investors, because your job as CEO is not to run out of money, and every investor needs to believe you are the future. Once public, it is choreographed: a couple of days each quarter and a week preparing.

The real risk is not company survival. If you do something the market dislikes you may lose a large fraction of the share price, which does not affect the company in the short term but does create a compensation problem, because people are paid in stock. So the thing to manage is retaining key people whose equity is suddenly worth less, complicated by the fact that people joined at different times and are therefore at different levels.

He is unsentimental about volatility, and cites the day their lockup expired coinciding with a major market crash, when one of their largest investors sold everything and the company's value fell by roughly half. The crash lasted two weeks and the market then rose sharply again. They have watched the stock go up and down ever since and have got used to it.

THE ONE THING

His comparison of the two kinds of investor is worth keeping. You may complain that your venture investors do not understand you, but they spend far more time on your company and are far more concentrated than public investors, who follow twenty or thirty names and move in and out of them a few times a year. Public investors find it harder to reason about you. So the job is to tell the right story and then produce numbers that agree with it, consistently, over years.

5.6 Telling the engineering team they will stop writing code

The passage the video opens on. His co-founder stood in front of the whole engineering team a couple of months earlier and said that in two quarters they would not be writing any code.

He immediately qualifies it: they will write code, and rewrite code. The point is an inversion. You used to mostly write and sometimes automate. Now you mostly automate and sometimes write.

What convinced them was specific and dated. Around December, partly because the models crossed a threshold and partly because people had more time over the holidays to try a hack project, they watched genuinely excellent developers, not beginners, rebuild entire systems alone in a couple of days. Work that was always possible, but that would have taken a team of six around six months.

WATCH OUT

He is honest that the change is unresolved and uncomfortable. It changes everything, and the tricky part is that nobody knows what the destination looks like. They know smaller teams can do much more, so you can explore something with two or three people instead of eight, which is excellent. But nobody knows what the process looks like a year out, or how many product people, designers, security people or engineers you need to build a thing. He says the reorganization is painful, and that they are doing it anyway.

Their own bets in response: investing so the product can be used by agents, because they see customers doing that; and building the intelligence directly into the product. His analogy for the alternative is pointed. Using a general coding tool to work through your operational issues is a bit like buying a business-class ticket to send a postcard. The work should be fused into the data rather than sitting outside it.

He also notes an advantage of their position: they have the top AI companies as customers alongside a good fraction of AI startups and the large traditional enterprises, so they see a wide panel of behavior. With one caveat he raises himself, which is a nice piece of self-correction. The labs use the product in strange ways because they have effectively unlimited compute and models nobody else has, so they are a glimpse of what is coming but not representative. Most companies will never be happy to burn that much compute on internal work.

5.7 On the co-founder relationship

He and Alexis have worked together for around twenty years, and he jokes they are an old couple now.

Two things he credits. They worked together for almost ten years *without* starting a company, so they had already tested the right way of working together and the right boundaries. And when they did start, they gave themselves a lot of room to talk.

TRY THIS

The mechanism is a standing lunch, originally weekly and now roughly every two weeks, with no agenda. You just talk. His point is that when you run a company there are very few people you can talk to, and a co-founder is a good one.

On whether European founders should start where they are: he thinks it is fine now, and says it was not true when he started, because getting funded anywhere in Europe was hard. What he considers critical is going after the US market as soon as you have some product-market fit. But not the whole company: typically one of two founders moves, because you cannot start in the US without a founder there. Datadog did the reverse, opening in Paris later, and mostly for talent, including people on visas who could not renew or wanted to return for family reasons.

PARTING ADVICE

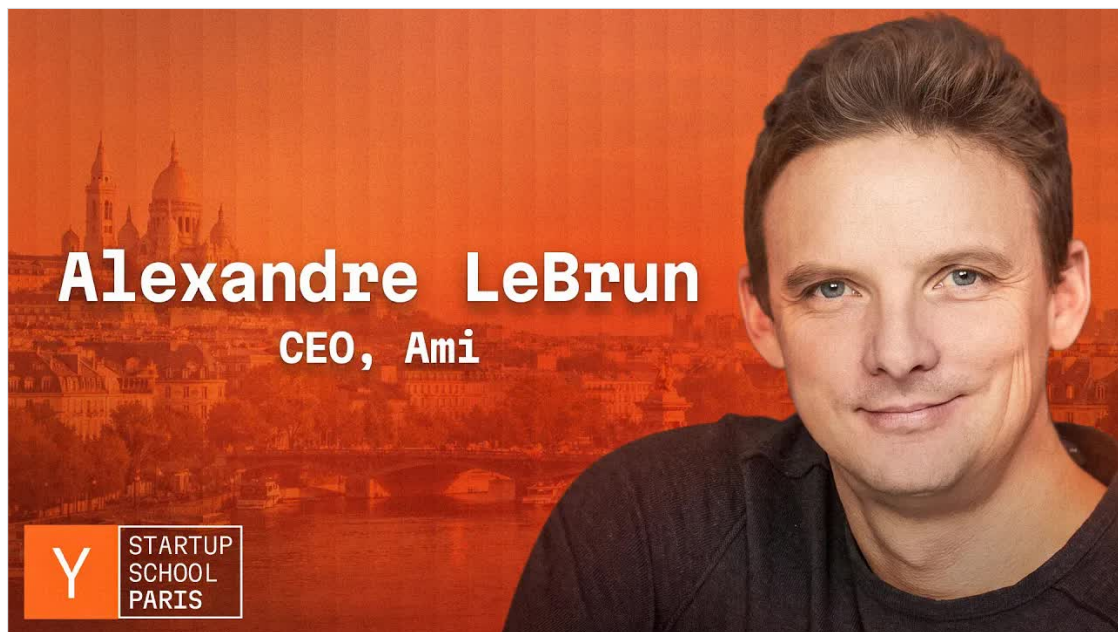
Asked for one piece of advice to his younger self, he first resists the premise, pointing out that the mistakes and the hardship are part of what made them successful, including how hard it was to raise money.

The one lesson he says he keeps relearning is to move faster, especially on hiring and firing. He notes people always cite firing and rarely hiring, and argues the two are the same lesson. If you need a head of sales and are unsure you have found the right profile, find someone who looks good and hire them now; if it does not work, you part ways. You will run more cycles and get more chances, and it is too hard to know in advance anyway. He was paralysed early on trying to make the perfect hire.

And on the other half: the first time you fire someone you feel horrible and assume everyone will hate you for it. What you mostly hear from the rest of the company is a question about why it took you so long.

CHAPTER 6

Alexandre LeBrun: What Big Tech Missed



"What Big Tech Missed And How Startups Can Still Win"

Alexandre LeBrun CEO, Ami. Interviewed by a YC partner, 25 July 2026.

Video: <https://www.youtube.com/watch?v=FVsgX0AdDT0>

6.1 Four companies, all of them early

He and the interviewer were in the same YC batch years ago, when he was building a natural language company. He has since founded four; the interviewer notes he has done one.

The pattern across all of them is being early:

- **2002:** a chatbot company for customer service, started almost straight out of school. He calls this one genuinely far too early. Trying to sell it, people asked what a chatbot was. He would explain it was like live chat but automatic. They asked what live chat was. They survived roughly ten years, long enough for a market to begin existing around 2010, which was still too early.
- **2013:** a natural language company, which he says was not *that* early. It had the first domain of its kind, which he bought because he thought it would be cool, and nobody understood it was a domain at the time.

- Then a health company, and now a company building world models.

THE ONE THING

His explanation for why being early suits him is really a piece of advice about matching the timing of an idea to your own strengths. If your strength as founders is go-to-market, sales and marketing, you should *not* be too early: you need the product and the market ready to receive what you do well, and being early will kill you. His own strength is taking very early technology and working out what product could carry it, so early ideas suit him. It depends entirely on what you want to focus on.

6.1.1 The acquisition email he deleted

In 2015 he received an email from Mark Zuckerberg and deleted it without opening it, assuming it was spam. About half an hour later he thought it might not be, went to his spam folder, and it was not.

He notes they had followed the standard YC advice up to that point: do not talk to anyone unless you are selling, because most approaches come from corporate development teams who need to talk to everyone to understand the market and to find out whether you are desperate enough to sell cheaply, which wastes your time and attention. The refinement he adds is that if the person contacting you genuinely has the power to do something big, it may be worth the conversation.

6.2 What a world model is, and why he is building one

He explains it without jargon, and the explanation is the clearest part of the talk.

A language model learns to handle language, trained by reading effectively everything humans have written. So it learns what people have *written about* the world rather than learning from the world itself. Text is a proxy.

THE ONE THING

His image for it: a language model is like someone who has never left the room they were born in, but has read every book, every day, for centuries. That person would seem expert in a great many things while having no direct experience of anything. They would have limited common sense, and limited ability to work out what to do when facing a situation for the first time.

A world model, by contrast, does not take the shortcut. It learns directly from the world the way people and animals do: video, audio, sensory data, handling objects. Starting from scratch like a baby, without language.

They train mostly on video because it is easiest to obtain, though he notes it would work with audio alone, in the same way a person born without sight or hearing can still live and learn through touch. Touch data comes via robotics.

6.2.1 What it is for

Robotics, in the near term. Current robots are narrow: they do one thing in one position, and it takes a long time to train them to do that one thing. Nobody is close to being able to put them in an open environment like a house or a street. His examples are the videos of robots knocking things over or performing near a child, and his summary is blunt: the hardware is remarkably advanced and the brain has not changed, so they are not safe or useful in the open.

He is careful not to overclaim against language models. They are already very good and he does not expect anyone to beat them at what they are for: anything that is language, mathematics, or programming, which he characterises as low-dimensional sequences of symbols. His argument is that there is a separate class of problems, high-dimensional and noisy with a long horizon, where he thinks world models will be much better.

He is also clear about maturity: some of the underlying ideas have been around for five to ten years, and some flavour of them is already in production in autonomous driving.

WATCH OUT

Asked whether this is a contrarian bet, he declines the framing. It was contrarian a year ago. A year ago, saying that language models were powerful but would not by themselves lead to general intelligence got disagreement; now he thinks a good number of people agree, and the remaining argument is about what *would* get there.

He adds a dry observation as evidence: the people who promised general intelligence have changed the definition and stopped using the term, which he takes as a sign something is not working.

6.3 Why a big company could not do it

This is the answer to the video's title, and he is credible on it because he was inside one.

His example is a chatbot built in a lab at a large company, trained statistically rather than from rules. It worked, they had fun, and the first time they asked it how it was doing it produced a reply that was completely unpublishable. He had wanted to release it. At that point he knew the legal team would never have that conversation with him, so it could not ship.

AGAINST THE GRAIN

The general lesson he draws: there are things you cannot do inside a large company that you can do as a startup. You can take a lot of risk, and if you take no risk you are dead, because the big companies have more resources, a brand, and many things you do not have. The one thing you have that they do not is the ability to do crazy things and take risk.

His worked example is the transformer idea: the paper existed in a large company and nobody did anything with it, and nobody realized its power. What another organization did was take it, scale it, and solve the engineering problems, which required both the money to do the training *and* the willingness to take risks nobody else would.

6.3.1 An ambition anecdote from inside

A month after the acquisition, he wanted to hire 100 people to do data annotation: concierges doing real bookings for real customers, recording everything they did in order to train a model. This was 2015, the company had about 6,000 people, and it was not in the budget.

You have to send a memo before those meetings. He arrived ready to argue every scientific point for why 100 was justified, and was told that the idea was great, that it would let them capture user intent, and that they should hire 10,000 rather than 100.

He then had to argue the other way, telling them not to, because there were too many unknowns and it was dangerous. To this day he does not know whether the offer was genuine or whether it was the fastest way to get him to volunteer everything that could go wrong with the project. They hired 200.

6.4 The largest seed round in Europe

Around 1.2 billion, which he confirms is a new kind of territory for everyone. His explanation for why it is necessary is unromantic: if you want to build a foundation model you need thousands of processors, which costs billions, and that is the only abnormal thing about the project.

He is direct that the bottleneck is not money. You need three things: people who genuinely understand this work, of whom there are not many and most are working on language models; a lot of data; and compute, which he says is very hard to secure even when you have the money for it.

WATCH OUT

His warning about raising a great deal early is a cultural one. Having a culture of not overspending, and treating the money as if it were yours, is very important and very hard to install later if you do not start that way. And it is harder here, because once you have spent a billion on compute, everything else looks cheap by comparison.

THE ONE THING

The line the video opens on, and the most quotable thing in the talk. He told his team that the real cost of raising 1.2 billion was not dilution. The real cost is expectations. Not from the investors, who understood and agreed with the long-term vision, but the expectations you create in the outside world. If you raise that much and people see nothing come out of it for two years, it is very hard to survive.

On the mechanics of raising it, he offers a genuinely useful reframe: difficulty does not depend on the amount. It depends on whether it is your first time. The first raise for a first startup is hard and you have to go through it. The second is easier. By the third, people ask to give you money for whatever you want to do. The interviewer adds the obvious caveat, which he accepts, that his first two companies having succeeded is part of why.

6.4.1 Running researchers, and choosing cities

He splits the company with a co-founder who holds the scientific vision and leads the research direction. His own job, as he describes it, is making sure there are people, electricity, compute, data and offices.

TRY THIS

His description of managing research is the most transferable thing here. You cannot tell researchers what to do; they hate it and will leave. But if you leave them entirely alone they will do somewhat random things and run in every direction, and you will never get anything out of it. The job is finding the very tight balance: enough direction, with enough freedom, to eventually produce a product.

They are in Paris, New York, Montreal and Singapore, because those are where teams already existed, and pointedly not San Francisco. When they hire people from San Francisco, those people have to move to New York, which he treats as a commitment test. At this stage he wants people who join with real commitment rather than because they want to try it. He does not think they miss much, since they travel there often and stay connected to the ecosystem.

PARTING ADVICE

His advice on ambition contains a specific shape rather than just an instruction to aim higher.

He thinks Europe is not ambitious enough and does not take enough risk. But the mistake he made himself was wanting to build a chatbot that did everything for everyone, which is far too broad for a small team to deliver value with.

So: choose a very narrow problem, one thing for one industry or one set of customers, and be extremely ambitious *within* that narrow tunnel. Do not be afraid to share the long-term vision: say this is what I am doing today and this is what we want to do later. If you are very ambitious across a very wide field, you are not credible.

CHAPTER 7

YC Partners: What Actually Makes A Startup Durable



“What Actually Makes A Startup Durable”

YC Partners A four-person panel taking audience questions, 25 July 2026. The captions carry no speaker labels, so the points below are reported without attribution to individuals.

Video: <https://www.youtube.com/watch?v=99sPd15j3Zc>

7.1 A note on this chapter

This was the closing session: four partners answering questions from the room for nearly fifty minutes. Because the automatic captions do not identify who is speaking and the participants are not named on screen, nothing here is attributed to a particular person. Where partners disagreed or built on each other, that is noted as coming from the panel.

7.2 The cost of intelligence

The first question was about economics: if you can do an engineer’s work with a model, when does that actually become cheaper, given tokens can cost more than the engineer who gets it right first time?

Two answers. The flippant one about credits, and then the substantive one: the cost of intelligence is falling fast enough that even where it is not true today it will be within six to twelve months, at roughly

a tenfold cost reduction per year for the same capability. It is something they tell founders not to worry much about, because time solves it.

A stronger claim followed. A year ago it might have been reasonable to argue that for a particular task a given engineer was more cost-effective than a model. The panel's view now is that there is no realistic chance that holds if you are using the best available intelligence, and that there is not a single programming task where a human is more effective than a model.

7.3 Where is the hard bit?

The question that produces the best answer in the session, and the one the video is named for. A founder asks: if any internal tool can now be built instantly, what is the point of business software at all?

The panel agrees with the premise but reframes it from a yes-or-no into a spectrum.

THE ONE THING

Your startup has to have a hard bit. Ten years ago, writing a lot of software was itself the hard bit. That is no longer hard, so you need something else to be hard in order to have any durability.

Things that still qualify as hard, from their examples: brutally difficult enterprise sales into certain industries; regulatory barriers, where one partner notes that getting a banking license almost killed him; the physics of launching a spacecraft; and hardware generally, where the atoms are the hard part.

The practical instruction is to place your idea on that spectrum and move away from the easy end, deliberately, because it makes the company more durable. They name specific categories of pure software product they would not start today, on the grounds that pure software is now too easy to replicate.

7.3.1 The same idea, applied to capital

Later in the session this connects to fundraising. If you are producing the same software people produced five years ago, you can absolutely do it for far less money now, but the result is much less durable and defensible.

THE ONE THING

Their conclusion is that AI lets founders attack harder problems, and as those problems get more ambitious the capital requirement goes back up. The examples given are a founder building small nuclear reactors who will need to raise several hundred million, and the difficulty of launching a regulated bank without a great deal of money. One partner argues this is good for the world, because we end up with solutions to harder problems.

They also address whether the traditional funding path still matters. Founders genuinely do not need to raise as much today to achieve the same thing. But it is the same world for everyone, so the bar moves with it.

7.4 Launch early, and why smart people find that hard

Asked what the top-performing companies in a batch have in common, the answer was immediate and unambiguous: they launch early and often, by a wide margin. The most common regret after a batch is not having launched sooner.

The diagnosis of why founders resist is the valuable part, and it is aimed directly at the room.

WATCH OUT

Many of you were selected because of impressive things in your background, and many of you come from environments where sitting in a library thinking very hard about a problem is how you get the top grades. That translates extremely poorly to startups.

Humans are bad at predicting the future. A startup is an empirical exercise: form a hypothesis quickly, run a test against the real world, see whether the data matches, and change approach if it does not. Running that cycle is the core activity of an early-stage company. It is not thinking very hard in a library.

The panel adds that founders convince themselves a few more weeks of research is the smart move, and it is not. The example offered is the founder of a now enormously valuable coding company posting launch after launch to no reaction at all, over and over, before it worked.

TRY THIS

Inspect your own biases here. Most founders are far more comfortable building than talking to customers, and going out to find whether anyone wants the thing is frightening precisely because the answer might send you back to the start. There is real loss aversion in that. So if you feel very comfortable in the research or building part, treat that comfort as the signal, and deliberately dial up how much you talk to customers.

The second habit of the strongest founders: set very ambitious goals, reassess them every couple of weeks, and hold yourself to them. The pattern they see is that the best founders arrive at each meeting with a *new* problem, because they overwhelmed the previous bottleneck. Work out the single biggest thing, throw everything at it, and come back with the next one, rather than being distracted by the hundred smaller fires available.

7.5 Co-founders, and what AI does not change

On whether a one-person very large company is possible: in theory, probably yes. The panel's objection is economic rather than romantic.

THE ONE THING

Adding a second person has a low marginal cost relative to the benefit, so it is illogical not to, unless you are specifically trying to set a record. But that ratio changes sharply as you add people, because coordination cost grows with the number of connections between them. At a hundred people it is hard to keep everyone aligned; at two thousand it is very hard.

So their expectation is that AI compresses companies downward, from thousands to something nearer a hundred or fifty, where people can actually maintain relationships. Not that it takes them to one.

The evidence offered is a specific case from a recent batch: a very strong solo founder building autonomous agents that would communicate with each other, who did not want a co-founder. About six weeks in he became sad and had a crisis of confidence about whether the direction was right. The partner's reflection is that this is exactly what a co-founder is for: to pull you out of the gutter when you are down, and to pull you back to earth when you are manic. They are explicit that YC does fund solo founders and has nothing against them, but the bar is higher.

7.5.1 How to pick one

Their criteria, and a warning:

- Someone who can lift your mood when you are low is a good start.
- The smartest, most determined, hardest-working person you have ever worked or studied with.
- An integrity check: do you trust them, are your values aligned?
- A useful test: in a competition, would you be worried to be on the other team?

AGAINST THE GRAIN

They think complementary skills do not matter at all, and name the biggest mistake they see: technical founders believing they need a business co-founder. If you are technical, find someone as deeply technical as you who you work well with. The business skills are, in their view, very easy to learn.

A second partner adds the reason this has become more true: it is now possible to go from an average engineer to a good one, or from not knowing how to sell to being decent at it. So complementarity matters less, and what matters is whether you are genuinely glad to talk to this person when you are miserable.

7.6 What not to delegate

Asked what they would automate last, or never, if starting again, the answers were consistent.

- **Talking to customers.** One partner says this is what kept him focused on what mattered and is the last thing he would trust to anything else, because it made him the person closing the loop with all the context on what to build. Even in a self-serve business you need that direct exposure, if not with everyone.
- **Talking to your co-founder.** Their stated reason: co-founder problems are the main reason companies fail during a batch.

There is a related answer earlier in the session about writing. One partner deliberately writes most of what he writes himself, on the grounds that there is a direct relationship between how much thinking you do and how you write, and he does not want to delegate too much of the thinking.

THE ONE THING

The organizational version of the same thought: alongside asking what AI can do for you, ask what it cannot, and make sure you are excellent at that. The examples they give for YC itself are founder wellbeing, community, and the fact that a room full of people obsessing over the same new problem produces knowledge that does not exist as content anywhere. They also point to the value of being witnessed: having someone present who acknowledges that what you are going through is normal, in a job that is extremely lonely.

7.7 On pivoting

Asked whether the standard advice changes now that pivoting is cheaper, the panel pushes back on the premise. Writing software is cheaper. Pivoting means discarding a good chunk of what you have built, which may well be the hard part, and that has not obviously got cheaper.

Their framework is about evidence. You have a top-level hypothesis that this is the right business because the world will look fundamentally different in five years, and a set of smaller hypotheses about the specific product. Every rejection is data testing those. You should only pivot when you have real evidence the fundamental hypothesis was wrong and have genuinely exhausted the options.

WATCH OUT

What normally happens instead is that you run out of enthusiasm. The kernel of a good idea is there, it is a long way from product-market fit, you are bad at sales, you keep getting rejected, and you get sad. So you think a completely different idea would be easier. It will not be. Pivoting from something you are deep in to something you are shallow in means discovering a whole set of problems you were not aware of.

The version of pivoting they endorse is when working on a hard thing exposes you to a better idea, and specifically when customers pull the pivot out of you: they tell you they want the underlying capability for something other than what you built it for, and that is a larger idea. Those decisions are much easier to make. Their summary of the usual conversation is that they are talking to a founder who is getting sad, and the question is whether that founder needs a pep talk or has genuinely exhausted the options.

7.8 Shorter answers worth keeping

- **On building a community from scratch:** do not. Join an existing one. If you cannot go to the largest hub, go to a city with at least a critical mass of startups, because without peers to push you the people who do not understand your life will hold you back. One partner traces a family tree from a handful of early companies to dozens of later ones in London.
- **What early startups feel like from inside:** a cult. You believe something nobody else believes, and if you said it outside the room people would nearly laugh. Look for that.
- **On the term “AI wrapper”:** they consider it close to meaningless and derogatory. Almost everything being funded has AI at its core, because it is the first time anyone could attack that problem. Whether you add value on top matters; whether someone snickers at you does not. Their historical note is that some of the biggest companies today were called wrappers of whatever infrastructure came before.
- **On what they actually bet on:** the founders, which they consider the biggest advantage by a wide margin, and more true now than before. Roughly a third of companies in a recent batch changed their idea anyway. The illustration is a pair of founders who applied at nineteen, having already built and sold a payments network as teenagers, with an idea for cardboard virtual reality glasses. The point being that with a team that strong the idea was almost irrelevant.
- **On deep technical founders:** they were biased toward teams with at least one strong software engineer and are now testing whether that is still required, having funded people who taught themselves recently. They say openly that they do not know the answer yet, and that you still need to be very smart, determined, and a systems thinker.
- **On model choice:** they expect routing to become automatic and invisible within roughly twelve months, with bills simply going down, done both by large labs and by open routing models.
- **On export restrictions:** they say plainly they know no more than the audience does. The interesting observation is that it was a wake-up call for how dependent the rest of the world was on decisions made elsewhere, and that this itself creates demand for alternatives and for sovereignty over inference.
- **On standing out when everyone is advertising:** the point of the story about a company buying every billboard in San Francisco is that it worked *because* nobody was doing it, and it would not work now. The job of a founder each time is to work out how to differentiate, and if the answer were obvious everyone would do it.

TRY THIS

The most concrete answer in the session, given to a founder who understood the concept of an AI-native company but not how to start. Do not try to do all of it. First make your information

legible and queryable: email, documents, support tickets. Then pick one deliberately narrow process and implement it. Their examples run from the mundane, sending a follow-up after every sales call with the key points, to the ambitious, transcribing each sales call and having something build a prototype of any product idea mentioned so you can demo it by the end of the call. Pick the narrowest implementation where you can see value, see whether it is any good, and improve it.

PARTING ADVICE

Remember which game you are playing. Looking at how companies with product-market fit acquire customers is looking at a completely different game from the one you are in. At the early stage you are trying to find first customers and learn from them what to build.

The way you beat a company with a billboard on the highway, for an individual customer, is the unscalable thing: turning up in person, spending an unreasonable amount of time for what is a small contract, and building something that fits their needs exactly.