How Cursor is building the future of AI coding with Claude

30 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=BGGSOIGBT_Y
https://www.youtube.com/watch?v=BGgsoIgbT_Y

SUMMARY

Cursor has rapidly evolved over the past year, leveraging advancements in AI models to enhance coding capabilities and streamline software development. The team discusses how their internal use of AI tools fosters a recursive feedback loop, driving innovation and improving product features while addressing the challenges of working with large codebases.

- *Cursor has scaled to over $300 million in revenue in just over a year, with millions of developers using the platform.*
- *The introduction of advanced AI models, like Claude 3.5 Sonnet, has significantly improved coding capabilities, enabling multi-file edits and better reasoning.*
- *The team employs a self-improving feedback loop, using Cursor to develop Cursor, which allows for rapid iteration and feature enhancement.*
- *Background agents are a new feature that enables asynchronous coding tasks, allowing developers to manage multiple changes simultaneously.*
- *The future of coding may involve a higher percentage of AI-generated code, with developers focusing more on design and intent rather than low-level coding details.*
- *There is an emphasis on maintaining code quality and "taste," as AI tools can produce code quickly but may not adhere to best practices.*
- *The integration of AI in coding is expected to democratize software development, enabling non-developers to create tools tailored to their needs.*
- *Startups like Cursor and Anthropic attract top talent due to their innovative environments and the opportunity to work on impactful projects.*

- I think like every facet of producing software I think will be kind of changed to use AI in some way. - Very excited to have you guys out today. Looking forward to this
conversation for a while. As you know, I'm Alex. I lead our Claude Relations
here at Anthropic. - I'm Lukas, I work on
agentic systems at Cursor. - I'm Aman. I'm one of the founders and I work on ML and retrieval at Cursor. - My name's Jacob Jackson, I work on ML at Cursor.

- I'm very, very excited
for this conversation and to talk a little bit about Cursor, what you guys are building and also how you're using Claude. It's been a big year for Cursor, pretty obvious to anyone that's been following
along the AI industry. You guys have scaled now to

over $300 million revenue in just over a year. Pretty crazy millions of developers are now using Cursor. What's changed in your opinion and how is today in the version of Cursor today different
than it was a year ago?

- Yeah, I think there a few
big things that have changed. I mean there's always been
this massive overhang in, given the current level
of the language models, how much you can do with them and I think Cursor was probably one of the first companies
at least in coding to be able to close that gap a bit with a number of different features. And then in turn, you've also seen these models get much, much better at coding and I think 3.5 Sonnet was like the first clear example of this or this kind of step function
better in programming.

And so before then, Cursor is really useful at things like tab completion, right,
predicting your next edit. And that alone was, you know, growing fairly quickly and then editing within single files. But we did see that when you kind of mix the intelligence of a model like 3.5
Sonnet with a few other kind of custom models we use for retrieval and then applying the edits
made by this larger model, you now have the ability to do kind of multi file edits.

I think that was kind of the step function that resulted in mass adoption of Cursor and since then it's been a mix of the models getting
better than us trying to under the hood get better and better with like how far
we can push these models. - And was that a natural progression, something that kind of just arose or did you guys notice when 3.5 Sonnet that first one came out that, holy cow, now we can all of a sudden do all these different things that weren't possible before?

What did that kind of look like? - It did feel somewhat gradual. Like there are these
steps in model quality, but you saw hints of it with you know, the prior state of the art model. In fact, we've been notoriously bad at taste testing these models just because you know, the way we use them is very different than when you put it out into the world to
see how others use it. But there are just hints of over time each kind of new model that came out was better and better at being able to reason, do more agentic types of coding and then it's a lot of tinkering and trying lots of things, seeing what works, seeing what fails.

Yeah, I think Sonnet was
probably the first one where we were able to make the multi-file kinda interaction really work well. And since then there's been a number of step functions including
like tool use, right? And then you can actually have these models act like real agents within the editor. - Hmm, I see. So the progression of the new models, new capabilities over time kind of allows for further tinkering, exploring, which then rolls back

into your product in some degree and allows you
to build new features.

- Yeah. - That's interesting and kind of parlays into
this next question I want to hit at which is I've heard many stories of how your team is using
Cursor to build Cursor, it's in this like self-improving recursive feedback loop. First off, maybe you can dive into a little bit of how that looks and on a day-to-day, what does that look like within Cursors you guys are working on building new features? - Yeah, I think it very much depends on the individual like yeah use cases for each employee and I think it also very
much depends on what part of the product you might be working on and what kind of stage that part is in.

So I think for like initially laying out some code base, some new feature, it's very, very useful to just like use the Agent
feature to kind of get that started and then to maybe use the thinking models to like look at individual
box that you might be facing and then for making
like very precise edits, I think that's, it's a lot of tap also and then when initially getting
started with a code base that one might not be too
knowledgeable about that using kind of the QA
features a lot using a lot of search and I think
that's also something that Claude 3.7 and 3.5 also has been excelling at doing research in a code base and figuring out how certain things interact with each other.

- I see, so using these features to explore your code bases makes the process easier then you learn as you're using these features that oh there's a deficiency in this area, we should go work on that. - Yeah, easier I think Cursor's it's very much driven by kind of solving our own problems and kind of figuring out where we struggle solving problems and making Cursor better and then yeah, figuring
out what we can do there and then experimenting a lot.

We very much have this philosophy of like everybody can just try things and try adding new features to the product and then see internally how they are used and what kind of feedback they gather. - Do you think there on maybe of a more meta level there's an advantage to being your own best
customer internally? - I think 100%. I think that's how we're able to move really quickly
in building new features and then throwing away things
that clearly don't work because we can be really
honest to ourselves of whether we find it useful and then not have to ship it
out to users, kind of track how people use it before deciding to go ahead with a feature and I think it just speeds up the iteration loop for for building features.

Yeah, going back to overall
how we use AI to program, it feels like, I mean there's a lot of diversity within the company and how different people use it. I think it differs first in like the kind of work you're doing. So, you know, there are a

number of people that will for example, be working in pieces of the code base they're
really familiar with, right? And at that point when you
have it all in your head, it's often faster for you to kind of convey intent just by kind of typing code and then for that Tab is really useful 'cause kind of speeds you up there.

But then when you're in places where you're less familiar or you need to write out a lot of code, you can kind offload a lot of that and often some of the reasoning to these models and then, you know, as you got to places where you're really unfamiliar with Lukas is describing and you're kind of coming
into a new code base, it's just there's this
massive step function that you get from using these models and what we kind of see is over time as the models get better is and as Cursor gets better using these models you do a better and better job of when you're more in flowing when you have more knowledge of the code base.

- So there's a variation in when a feature is most applicable to like your use case and it kinda is like almost a spectrum to some degree. - Yeah like the spectrum on one end is Tab for when you're completely in control and you know what you're doing then it goes to Command K where you're editing
a single given region, maybe a whole file and then at the other end you have Agent which is quite good for, you know, editing multiple files and then at the very end you get kind of have this background agent which we've been working on and that can be useful for
basically doing entire prs.

- You guys just released a preview of background agent. What is background agent? - I think it's clear that the models are getting better and better at doing end-to-end tasks but they're not quite at 100% and I think it'll take a while to get to 100%. So the way you speed up developers, right, is you let them do these things in parallel but as opposed to kind of letting it just go in the background then spin up a PR that you look at in
GitHub if it's only 90% of the way there you want to go in and then take control and do the rest of it and
then you want to use you know, the features of Cursor
in order to do that.

So really being able to quickly move between the background and the foreground is really important and I think like, you know, we're in the early innings of this feature and I can imagine that there
are lots of interesting ways of being able to operate
for example on three or four changes at the same time and then quickly kind of
popping them to the background and then moving them into the foreground. It'll be interesting
to see how this changes how people use Cursor and just like develop the software in general.

- I mean we see background
agents basically as a new primitive that we can use in like so many different places
and the current way of exposing it is quite straightforward where you can just get a prompt and push it to the background and then it independently
iterates on that. But there can be like

many more integrations how these things can be spawned off and I think there's a lot of product that you want can make from that.

- So is this taking your code base and putting it in a virtual machine or what exactly is that transfer that's happening? - Exactly, yeah.
- Okay. - We have small enough independent environments that have all the developer environment utilities already installed and then the agent can use those and it has all the VS code extensions that are available and through that it can get et cetera. - I know we're kind of witnessing this trend of asynchronous tasks, background tasks across many different things from coding to like research, in your view, what does that look like as this progresses to where we might have thousands of these agents potentially going off and you could see like whole teams of agents attacking a problem all in the background.

What does that future look like? - I think the next bottleneck you'll run into is verification of software, verification of code, models getting really, really good at generating writing lots of code but let's say developers spend, I'll throwout some random-ish numbers, but 30% of their time writing code or 30% of their time reviewing code, 70% of their time writing code. If you completely solve writing code you still haven't really sped up software engineering by more than a factor of three.

Yeah, so I think we're going to need to figure out how to make it easier for people to review code and how to be confident that the agent's making the changes that are not just correct, 'cause correct can be vague, right? It may just be in the thing you specified, it was under specified enough that it actually did like the best that was possible for even you know, the best human programmers to do but what it actually what you had in your mind's eye and so making the process for you much, much better I think will be really, really important.

And it's something that we're really interested in as well. - Any early ideas there on what that looks like? - I think there are a few floating around from various people at the company. One that Michael, our CEO who really, really likes is the idea of operating in a different representation of the code base. So maybe it looks like pseudo code and if you can represent changes in this really concise way and you have guarantees that it maps cleanly onto the actual changes made in the real software, that should shorten the time of verification a ton.

But that's one possible route. I think, so like the reason why quote unquote vibe coding works often is because the process of verification is like really easy since all it is just kind of playing with the software, right? You make a change and you actually play with whatever software you've built. I think it's just gonna be

really hard to do for real production code bases and cracking that problem
is really important. - That's a good question
around the difference between like a standalone
thing they might be vibe coding versus a production code
base that has millions and millions of lines of files.

How do you guys see the difference between those two in your mind and where are we at in terms of like working within
them with current models? - I think that's something
we've thought about a lot with background agent because something that's really simple and obviously should be very easy with these models is
I have this test here, the test is currently failing, can you fix the code so that it passes and it's like okay how
do we make that happen?

Well the model needs to
be able to run the test and if you have a very simple repository, that's very simple, but when you start getting to these larger enterprise code bases, it can be complex to get the dependencies set up properly so that the model can run the tests. But this is something we've thought about with background agent a lot is how do you make this
process straightforward for the developer to
create this environment where the agent can run the test and then make it repeatable
so you can snapshot it and you can quickly update it when your code state changes and this unlocks the ability to, you know, spin off a VM in the background, have the model make experiments, you know, and some of them will make it pass and some of them won't.

And then eventually you
as the developer only have to worry about the case where it succeeded and there's just a lot
of infrastructure there and a lot of user experience that is important to get right. - Mm hmm, mm hmm. - Yeah. And then I think there are other fundamental problems. So one way is you get the model to try to pass the test, right? That's how you can kind
of guarantee maybe, some sort of correctness.

But with these large code bases, you're often dealing with things that almost
look like their own language where they have these kind of DSLs within some languages and everything is done in this particular way and it's really sprawled out
across millions of files, which is hundreds of millions of tokens potentially maybe more. We've done a number of things to make this much better, which include training retrieval models and then integrating other
sources of context as well.

For example, you can imagine there's a lot of richness in the recent changes that you've made, when editing your code

it kind of indicates what you're working towards. There could be richness in the changes that other people on your team have made in your code base, especially recently and
using those as hints. But I do think it's still
this really hard fundamental problem of you know, just giving the model access to really good retrieval
feels insufficient for having the model really
understand the code base.

I think it's a problem we're really interested in solving. - Mm hmm. Probably through some combination of like memory plus long context and. - Yeah.
- Other things. - I think memory is one interesting approach people have taken to get the model to kind of learn from your usage of it but it also feels like, you know, it's a small boost in performance and it feels fairly primitive relative to like where we need to
be in order to get things that are excellent at large code bases.

- Yeah and large code basis, it's not only just about getting the test to pass but it also is about
doing it the right way. Like looking at the existing code and making that match the new code and bringing it into the correct structure and kind of using all
the guidelines correctly and like we've been trying
very hard to kind of make that happen through Cursor rules, through integrating different
types of context, et cetera. - Hmm.
- Yeah, like I could write a deep bounce function from scratch and just use that and
that would make the test pass but that's not the right way to do it.

You should use one of the DeBounces and maybe there's three or four DeBounce functions
used across the code base. How do you know what
the right one is to use? Maybe the only reason
like someone knows is because they message someone on Slack that this is how you do it. And so I think yeah it gets really, really hard to solve these problems with extremely large code bases. - That's interesting. So there's also kind of an element to the org knowledge that lives outside of the code base itself and that like plays a major factor sometimes in some of these decisions, especially as you're operating on- - Yeah.
- Large code bases.

- I don't think that's
the bottleneck today but I think if you solve, like if you made models like perfect and kind of knowing the code base, - Yeah. - I think you'll immediately,
like you'll maybe get like a 5x maybe 10x improvement but you can't get farther than that because now it's completely bottlenecked by, how much does it know these things that are never ever explicitly mentioned or shown in like the PRs and the actual state of the code.

- Mm hmm. - And then there also
just outside concerns from the business side

from sales, et cetera. And those kind of have to be brought into
Cursor to make that work. - Right. So some future version of Cursor then has to plug into many more systems- - Yeah.
- And things. - To be clear I think like, you know, that's like still some ways a way for that to be like really, really critical relative to the other things.

I think we have a long ways
to go still on just using the interactions users have like details of their code base and commits made in order to make Cursor much better. - One interesting thing I've started to notice at least with
like webpages and content, is people trying to now think about how to optimize the page for an LLM reading and browsing it. Do you think we're gonna
see something similar maybe with code and in that code could transform how it usually is written and what it looks like if you're writing for primarily human reviewers and humans working within
a code base to models?

- I think that's totally the case already. I mean API design is
already adjusting such that LMS are more comfortable with that. For example, changing not only the version number internal but making it like very
visible to the model that this is a new version
of some software just to make sure that the
API is used correctly. And I think that the same also holds, for like normal code basis and internal libraries as well where like structuring the code in a way where one doesn't have to
go through like end level of interactions but maybe
just through two levels of interaction makes, yeah, LLM models better at working with that code base.

- Right. - But I think ultimately the principles of clean software are not that different when you want it to be read by people and by models. You know, when you are trying to write clean code you want to, you know, not repeat yourself, not make things more complicated
than they need to be. And that is just important for models as it is for people. And I think taste in code and what's a clean solution that's not more complicated than it needs to be is actually gonna
become even more important as these models get better because it will be easier
to write more and more code and so it'll be more and more important to
structure it in a tasteful way.

- That's a really good point on taste. Taste is kind of this thing that I feel like maybe
some people are born with more taste than others, but generally you kind of
develop taste through experience and learning what works
and seeing failures and seeing successes. In a world where we're
having AI write more and more of our code, there's been real pushback against some that say, oh you're gonna make programmers lazy or you're not gonna give
juniors a chance to learn what it actually looks like to work within a large code

base and do all these things.

How do you think about balancing this sort of automation or assistance in this case with also preserving the core engineering skills that maybe a software
engineer has to go through, those like trials and tribulations? - I think these tools are
very good educationally as well and they can help you
become a great programmer. You know, if you have a question about how something works, if you want some concept explained to you, now you can just, you know, press command L and ask Claude, you know, what is this?

How does it work? Can
you explain it to me? And I think that's very valuable. It does make it easier to write more code and do more stuff and
that can result in higher and lower quality code being out there. That is true, but I think
in general it's a very, very powerful tool that
will raise the bar. - I think quality comes
very much from iterating quickly, making mistakes, figuring out why certain things fail. And I think models vastly accelerate this iteration process and can actually through
that make you learn more quickly what works and what doesn't.

So I think in the long term, it's a super helpful tool for developers just getting
started and working on bigger and bigger projects and figuring out what
works and what doesn't. - Yeah, I think it'll be
really interesting to see how programming evolves. I think you'll still for
a very long time need to have the engineers that
know the details right, can go into the weeds. I wonder how much you'll
start to see people that are now learning
programming who don't know many of the details but can
still be fairly effective.

I think today you still do need to know a lot of the details. I think over time you might have a class of software engineers
that need to know very few of like the low level details and it still operate at a higher level and maybe it looks a lot
more like kind of thinking through like the taste is like more in kind of UX taste, right? Like let's say you're trying to build something like a notion, right?

At the end of the day, I don't think you can offload that entire thing to the language model. You need to kind of describe like, okay when I do this type of interaction then I expect it to pop up in this particular way, right? Maybe you don't have to get to the details of writing pure software that does that, but still describing those interactions, describing the way this thing roughly works. That is a form of programming.

- Switching gears a little
bit on the topic of models, so we just recently, by the time this video comes out, Claude Opus 4 and Claude Sonnet 4 will
be out into the world. Love to hear your guys'
thoughts on the new models and how you're starting to think about integrating them within Cursor. - I mean we've really
enjoyed the new models. I think we were pretty shocked
trying out the new Sonnet because I think 3.7 is a fantastic model.

It was better at agentic coating but everyone knew it kind
of had these deficits right, where it would maybe be a little bit too overeager-
- Like to do a lot. - Yeah it did. Would like to change the test sometimes, that they passed.
- Yeah, yeah. - We found that Sonnet
4 has effectively fixed all those, it is much better and then the intelligence has also been a big step up where you know, you've seen other models that are kind of steps up in intelligence, maybe not as like strong as agentic coding but like you know, O3 is an example and we found it goes toe-to-toe with that despite being you know, a much cheaper model.

And so we're extremely excited for Opus because we think it'll be a fantastic agent to use in the background. - Yeah, that's awesome
to hear the test writing and overeagerness things are things that we were trying to tackle pretty intensely
with these models and concept of like
reward hacking in which the models will find some way to basically take a shortcut to get to the final reward in rl. So we've done a lot of
work to cut that down.

I think we cut it down to like 80% in these new models. - I'm really curious to hear how did 3.5 Sonnet come about, 'cause that felt like
the first kind of punch of like this is like a really good coding model for Anthropic. - How did it come about? We trained it. Just, it was good. Yeah, I think we have always
known for a while that, I mean probably since the
genesis of the company that we've wanted to make
models really good at coding, it just seems important for everything else that you do, especially as you make more models.

3.5 Sonnet was, I wouldn't, I mean I think 3-Opus was a really good coding model as well, especially for its time. But 3.5 Sonnet was the first time that we really put a
strong dedicated effort to, hey, let's get these
models good at coding, but not just specifically coding, this sort of longer horizon
coding where it's having to do these things like
you're mentioning earlier in the conversation around making edits on different files, going off and like taking a

command here, calling a tool and then going and making a change somewhere else.

That was the first model in which we could kind of put all
these things together and I think it just turned out really well and kind of set the stage for what our future models would be. - And how do you guys think about code versus other areas where you want Sonnet to excel? - Yeah - And Opus to excel. - I mean code is one of the primary areas, but I think it's not the only area.

I think there is a good amount of transfer that you see from models
getting really good at code to them just getting better at reasoning over taking many actions and working in this sort of agentic way. And that carryover is pretty nice as you're dealing with applications that might mix in code but also have to go retrieve
knowledge from other places or do research. Generally we're about just
pushing the frontier as much as we can with our models.

Of course there is like
considerations that we make around safety and making sure that the models are in line with what you as a user want and also what we believe
the model should be doing. But generally we want to keep pushing the limits of what these models can do and kind of show developers
in the world this is what models are capable of. So things like computer use, when we unveiled that back in October, that was like another
direction in which we're really pushing forward in terms of how can a model be good at actually navigating something that is a primarily a
human interface, right?

So it's not in the world of like APIs or tool calls or anything like that. It's literally just looking at an image as a human would and then having to direct an action onto that screen. There's also a strong part to how we think about
these models character, as it's known now, Amanda Askell is one of our lead
researchers on this effort, kind of crafting Claude's character and we put a lot of thought and consideration into what Claude should feel like and sound like and what does it mean for an AI to play a really prominent
role in somebody's life.

Not as just a coding agent, but as kind of like their
confidant in a sense and an entity that you're gonna be spending a lot of time talking to. So that's also really factored into all the decisions we make around these models
and how we train 'em. - How does Anthropic
as a whole think about where things are going both in terms of software engineering and then in terms of like research, like in terms of how much like these models will augment, replace, do a lot of this work?

- Yeah, it can speak personally here. So personally I think that we're not gonna be replacing, as we've been

talked about earlier, there's just like so
much more you can do now that you have like models that can do all this, you know, nuts and bolts like typing of the code basically for you. I see this with myself too. Like I studied computer science in college and did software engineering and now I feel like I'm at the point where the models are like better at producing code than I am, like if I were to just like think about doing like a lead code problem or anything like that, where it's like a contained environment and the model has to write code, it's gonna like beat me and yet I feel like I
can do more than ever.

I can make prototypes of anything. I can like spin up demos super, super fast if I wanna like
show off a new concept. It's felt very empowering
in that sense than like taking away or dismissive
of what I've been doing. And it is similar to
where I feel like just because I have that knowledge of software engineering from the past, I can actually exploit it much better and I can use the model, I can push it farther than if I just still didn't have any idea about what code is.

Maybe on that getting more into like the sort of fun future speculation I want to ask like maybe a practical question in a few years we can come back to this one and
see how we turned out. January 1st, 2027, so what is that a little
less than two years from now? What percentage of code do you think will be AI generated and following that, what
does the day in the life of somebody that's
considering themselves a developer now look like?

- I think it's similar to going back to, let's say before I was born, but you know, 1995 and asking a lawyer in the future what percentage of legal documents will be
word processor generated and the answer is 100% or you know, close to 100% in that AI will be involved in almost all of the code that gets written. But still your role as a lawyer or as a developer in understanding what the code needs to do and having taste and guiding what is done
with the software is going to be more important than ever.

- I mean already at Cursor it's probably 90% plus, but that's because a large fraction of it is using more higher level features like Agent. - Yeah.
- And Command K and whatnot. But then a lot of it is you're typing and then Tab will as you type do 70% of that.
- Right. - So in the cases where
you're actually going in and doing it manually yourself, Tab is still doing most of those changes, - Right, so the actual
letters typed is like a very, very low percent.
- Yeah.

But I think like every facet of producing software I
think will be kind of changed to use AI in some way.
- Do you think we ever get to a world in which you basically have software on demand? What does that look like? - I think you're going to see people building software, people in organizational
functions, building software who are not previously building software. You know, like people in sales who would not have built their own tools before will now be building, for example, dashboards to track
what's important to them.

And going back to how taste becomes more important
than ever, you know, now you can build the dashboard, but you still need to decide what metrics the dashboard is gonna show. It doesn't prevent you
from having to decide that. I think you're going
to see many more people building their own software, but it will be bottlenecked
on having a unique thing that you want to do with the software that isn't properly
served by existing needs. - One example I like to
tell people is we've a guy, our comms team who's actually been like shipping bug fixes to Claude.ai, which is just like absolutely insane.

Like he's in a completely
different part of the org, he's not touching product at all and yet he pops in with like a PR and he's like asking for a stamp and you're like, what are you doing? And it's like, yeah, he's
using, you know, Claude code or some coding tool with Claude has the base model there to like fix bugs in a production code base. I think that's amazing as well. And it ties back into
this like general, hey, if you have taste, if
you have good intuitions, like you're just gonna
be able to do a lot.

That's kind of how I see the
world keeping progressing. I think things will change and like roles will look
much different in five years, 10 years, but generally like I'm very much in favor of like, if you can do more with these things, like that's generally always
gonna be a good thing. - Yeah, I feel like there are a lot of interesting paths that this could take. One is just completely on the fly on demand software where I am using my own version some app and just like as I use it, you know, this interaction I don't really like and it just changes for me that's one kind of crazy
future you could imagine where it's not even you
kind of actively doing it, but just based on your
interactions with it, the software, whatever you're using, changes to kind of fit you.

That's like a cool potential path forward where I don't know if everyone in the world is going to want to like, I don't know if the
total size of like people who want to kind of

build their own software is like that large.
- Right. - But I think the people who could benefit from software that kind of fits their needs is potentially the entire world. - All right, maybe one last thing to just kind of close this off here.

For all the people watching this, if you're a talented engineer out there, and you're thinking about making your next move or you wanna get more involved in the industry and you're trying to decide between maybe going to a larger company or joining more of a faster pace startup like a Cursor or Anthropic, what would you tell someone in those shoes? - Yeah, I think startups have an advantage these days like with Anthropic and with Cursor and getting like really excellent talent in a way that like when you're at a bigger company, a lot of people, you know, a lot of the best people in the world find that less exciting, right?

And some people do, and certainly like large companies have great people, but the density of that talent tends to be much lower. And I get a startup, you can get this really high talent density and that makes it really enjoyable to work with a bunch of other excellent colleagues. You can work on really impactful things in this incredibly small team, right? Building a product that kind of, and building models that change the way that the world writes software.

And you can be a one of like, you know, tens, hundreds, or thousands of people working on that and that's really cool. - Yeah. That's great. Well thank you guys. This has been awesome conversation. - Thank you.
- Thank you.