# Uber: Leading engineering through an agentic shift - The Pragmatic Summit

37 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=I1TZN41VKCE
https://www.youtube.com/watch?v=i1tZN41VKcE

SUMMARY

*Anshu and Thai from Uber discuss the company's strategic shift towards agentic AI, emphasizing its integration into engineering processes to enhance productivity and reduce toil. They highlight the significant ROI achieved through AI, the development of various tools to support engineers, and the ongoing challenges related to technology adoption and cost management.*

- *Uber's AI strategy aims to augment human productivity rather than replace it, focusing on enabling engineers to engage in more creative tasks.*
- *The transition from pair programming to peer programming allows developers to delegate repetitive tasks to AI, resulting in increased satisfaction and productivity.*
- *Tools like Minion, Auto Cover, and Shepherd have been developed to automate code maintenance, testing, and migration, significantly improving engineering workflows.*
- *The company is experiencing a notable increase in developer satisfaction and productivity metrics since implementing agentic AI systems.*
- *Non-technical challenges include resistance to adopting new technologies and the need for effective communication of wins among engineers to encourage usage.*
- *Cost management is a critical concern, with AI expenses rising significantly, prompting a need for more efficient resource allocation and model selection.*
- *Uber is working on measuring the business impact of AI beyond activity metrics to demonstrate its value to stakeholders.*

Awesome. Good afternoon, folks. Thank you so much for joining. I really appreciate it. Yes, my name is Anshu. I lead the developer platform organization at Uber. And Thai is a principal engineer. He's been one of the the leading engineering voices that's led our agentic shift, but also our overall AI strategy over the past couple years. All right, let me try to get out of the way. We have a pretty packed agenda, but I want to try to get to the end as fast as possible because I'm curious about your folks' perspective on what I'm going to talk about.

We're going to walk through what has motivated our push into agentic AI and some of the key ROI that we we've we've manifested. And there is key ROI. I'm really excited about the impact that that we realized for Uber. And then I'm going to hand off to Thai and he's going to get into the specifics, the actual technologies that we built or integrated that has resulted in that AI that impact. And then I'm going to end the talk talking about some of the non-technical challenges that we've been dealing with.

Organizational and cultural basically people challenges, measurement, and of course cost. Okay, so AI is not new

to Uber. Our fares platform, our matching platform has been using AI methodology for years and years. It's one of the things that sets us apart from the competitors. But over the past few years using AI as part of the the engineering productivity and engineering life cycle, that's fairly new.

And it's AI's integration into not just engineering, but all aspects of what Uber employees do has become a bigger and bigger part of what we want to focus on. So much so that Dara has stated that AI is one of our six strategic shifts. We move from a early from a human {slash} early AI-powered company into a generative AI-powered company. Now, I will say that that term is really passe these days.

Nowadays, it's more fashionable to be an agentic-powered company, but the concept still holds where from the metrics, from the data that that we've gathered, Dara made this quote, which is, you know, AI AI is enabling people to become superhumans in terms of their productivity and the impact that we can realize for our end users. So, from that standpoint, we want to enable all tasks that people do at Uber to be supported by generative AI to augment human productivity.

And that last part is really important because what we're not pushing for is AI automate all humans in in the company, right? And especially on the on the engineering side, what we found is we want to focus on enabling our engineers to focus on creative work rather than toil. I'm going to get into some of the metrics and the impact that we realized from that, but the as we've unlocked AI, what we found is when we push some of the boring stuff to it, upgrades, migrations, bug fixes, not only does it result in much higher satisfaction for our engineers, they're able to push our product and create features for end users in ways that we didn't even thought was possible and at velocities that were just have just been incredible.

So, this has been the place that we've really been been doubling down on. Now, one of the reasons we've been able to push on this is the capabilities of of the technology and the industry. And part of it first part of it is we going from pair programming to peer programming. So, if you think about back in the good old days of 2022 and 2023, when GitHub Copilot first came out, it was pretty novel way of augmenting development. You had a system where you could do synchronous tab completion and IDE chat window that would help developers move faster.

We saw it ourselves in our in our metrics. We saw about a maybe a 10 to 15% bump in overall dev velocity. This is pretty phenomenal, but this by itself didn't push us in the direction that that we've seen over the past, let's say, year, where the paradigm has shifted to peer programming, where you could hand off workloads that are running asynchronously, and the models that that we use are so good, they're so accurate, that all you need to do with the AI agent is to redirect in certain ways, maybe give it some some course correction.

and that's that's all culminated in this model where we imagine developers acting as their own tech leads. Right? Developers are are directing AI agents using a variety of the different models and capabilities that are available to be able to execute asynchronously and come back for for direction. Now, this doesn't work for every single task, but again, when we think about some of the toil work that developers need to do, dead code cleanup, writing docs, library migrations, these all seem basic these are basic operations, but they're absolutely essential

for maintaining a healthy codebase.

But, they don't by themselves help to grow the business. So, pushing these workloads to AI agents in effect helps to grow the business. Another thing that's really helped is the growth of the capabilities of the system. So, on the right we see a logarithmic diagram that shows how long some of the models have been able to execute over the period of time. We go from, you know, less than 1 second to agents that can operate for hours and hours.

And that's helped with this paradigm shift that's happened where, again, when Copilot was first introduced, it was still augmenting traditional development. but as the capabilities have gotten better, we've seen that this concept of vibe coding, which was just a joke a couple years ago, becoming much more prominent and much more of a serious concept. and it's resulted in companies like I have an example for Ramp. I know they have a talk today. but there other companies that have had similar examples. In fact, even Anthropic talked about how they were able to release code work very quickly using a variety of agents.

this example is not unique, but it is representative of where the industry has been going so far. Okay. So, talking about toil, Ty is going to talk about the the agentic system that we've deployed out. As soon as we made our agentic workflows available to developers, what we saw is 70% of the workflows that developers are pushing into the system were toil tasks. There's a couple reasons for that. One is the accuracy of these tasks was much higher compared to the more ambiguous workloads.

And it makes sense, right? Like the start and end state of some of these tasks, if you think about a library upgrade or a migration, is much more straightforward versus say building a brand new feature or an experience out that requires an experiment. Because the accuracy was higher, developers were more likely to push more workloads into the agentic system that were toil oriented. And it became a virtuous cycle that that we saw. and based on that, based on the success, we we pushed for making this as one of my org's developer platform's top priorities in terms of AI augmentation.

Okay, I'm going to hand off the time now to get into specifics. I'll start by saying that we are not building this in isolation at Uber. Uber's had a long history of building AI solutions and having a lot of engineers across the organization building infrastructure. And so we really see ourselves as building on the shoulders of giants. Where you know we have our our historic Michelangelo platform which has had some public content in the past that provides things like a model gateway so that we can proxy and talk to the main frontier models or host internal models, traditional inference training platforms and all the other things you would expect in an ML platform that within the last couple years has really started to lean more into like the agentic side and the APIs that we're using to talk to open AI and Anthropic and those folks.

On top of that, we have a lot of the traditional infrastructure and context at Uber that we would want to take advantage of. Things like having access to our source code, our engineering documentation, Jira tickets, Slack information. Like these are all things that to have an effective agent have organizational memory it needs to start to get access to. One of the key ones that I'll dig a little more into in the later slides is our deployment of MCPs throughout Uber.

On top of that, we see a lot of industry agents. We really take a perspective of trying to enable the latest and greatest for our engineers allowing them to experiment, allowing them to have a a learning culture and use the best of class. So that means that there's a lot of clients that are coming in that folks are using and we use a lot of those to build specialized agents. This could be our background agent platform that we're going to be talking about, our test generation platform, or many other kinds of internal ones. And then at the at the top of that we have you know the engine engineering enablement phrase that's been going around the industry. It's you know measurements and cost control and education and everything else that you would expect.

So let's dig in a little bit to how we think about MCPs. This became a very popular piece of technology in the industry last year, and we moved very quickly to make sure that this was deployed and secure for our engineers so that we could make them as productive as possible. So, we ended up putting a a tiger team together from across the company. They came together and designed a strategy and built a central MCP gateway.

this allows us to both to proxy external and internal MCPs from our service infrastructure and expose those in a consistent way to engineers that handle things like authorization, telemetry, logging, everything else that you might expect. We also provide a registry and a a sandbox so that developers can come in, they can play with these MCPs, they can make sure that it's going to do what they're expecting, and that they can discover new ones. Continuing on with that, we also have Eduber through our ML our Michelangelo platform built agent the ability to build agents with both SDKs and with no-code solutions. Building agent builders, the ability to visualize, have telemetry, do tracing. that way as folks around the company are building some of these solutions that have access to the internal services, the data sets, we can reuse these in other systems.

This can be discoverable, and that we can provide a registry that's then can be found by other engineers or non-engineering alike to to deploy these. And they're deployed consistently in a lot of our environment. This can be from our dev pod infrastructure, which is our remote dev environment, local laptops, through our background agent, which is called minions. We're going to introduce that here in a minute. Or deploy these in production. So, we we talked about the agents, the kind of registry there, the MCPs, the registry there, all the different agent clients that our engineers might be using, be it Cloud Code or Code X or Cursor.

one thing that we recognized we needed to platformize pretty quickly was a central ability to provision and configure and update the clients, the agent clients themselves, the ability to install and discover MCPs from the registry, configure those inside of the agent clients, deploy standard configuration management so that people who are just new to the space are having more effective prompts and configurations right away. And management in connection into our background task infrastructure.

so we built this tool called AIFX CLI and it is the kind of the forefront of what developers are using to access our agentic infrastructure. So, let's let's take a minute before I jump into our our specific product and think about the traditional developer workflow. If you looked at how people were spending time a little bit in planning, probably a lot in code authorship historically, and then a small amount in review. And then typically they'd be in this edit, run, build, run loop of editing their code, building it, doing the verification, using some standard IDEs.

Now of course this has been changing significantly with the agentic world. And so if we look at what the the first agent workflow looked like, it might look something like this. You have a developer who's in the middle of using Cursor or Cloud Code, they're giving a prompt, it's asking for the ability to proceed and approve commands, and they're very interactive in the loop trying to drive it to an outcome that they want. but what we're seeing emerge now in the industry and at Uber is both background agents that are running fully autonomously, as well as the ability for multiple of these to be run at once.

Right? This gets into this place where as an engineer you're giving a prompt, you're waiting for something to you're waiting for some time while it's running. You're thinking, "Oh, what am I going to do? Am I going to go have a coffee or browse Reddit? Might as well kick off another background agent." And so you they get into this mode of the the new flow looks like running several agents at once, right? This this sounds great. We're I think us and a lot of the industry is trying to to push towards this. But a lot of challenges start to emerge with this different way of working. One of them for us was we wanted these background agents to be running autonomously and looking at the external vendors that were offering, you know, tools like like cursor and cloud code and codex. All of them are running their background agents in other people's infrastructure. And while we can get there, while that may make sense long term, having the ability to bootstrap on our own infrastructure was really important to us and allowed us to move really quickly. And so we built a product called Minion. Minion is our formal background agent platform.

It's built on top of state-of-the-art agents, CLIs, and SDKs. This leverages all of Uber's existing infrastructure. It runs on our CI platform. This has our mono repos checked out, ready to work in quickly, handles all of the network access into the rest of the infra, allows the connection to all of those MCP servers that we talked about earlier through AIFX. It's integrated for the developer in a bunch of different work workflows and panes of glass. There's the web interface we're looking at here, which is one of the main interaction paradigms, but it's also available through Slack, through GitHub PRs in the code review process, through the CLI that we saw earlier, and we have APIs exposed so that it can start to be connected to by other workflows and other services throughout the rest of Uber.

And one other powerful thing is this offer is good default. So when people are coming here and kicking off these background jobs, you know, they're giving a prompt, they're expecting a PR out of this. They may not be giving the the ideal prompt or it have the ideal setup for it. And we can provide great defaults for each of our mono repos, make sure that this is more likely to have a successful task that the engineers authoring than if they were, you know, if they just did this locally and they didn't have a lot of the code empty setup or the other context that may want to provide. So let's walk through a demo real quick of what using Minions is like with an example.

So we have this web interface and in this this is one of that I actually ran. We had a user report an error. They said, "Hey, this is crashing on my machine when I run this command. Here's what the error is." And I threw that into into Minion. I said, "Hey, you know, we're having this issue. The user's on a Mac. Here's the error they're seeing. Here's Here's the command that was run." And so you can see a few things here that are cool. One, we have these existing templates that users can choose from that are well-written prompts that have placeholders they can fill in. We have the ability to choose and run in our different mono repos. I can run in both on a branch

or we can switch it to a follow-up task of existing PRs or diffs.

We have all the task history here. Then we have some cool things here. We have I can select the agent. So in this case I'm going to run it in cloud code. Put it out as a GitHub PR. We've been in a long multi-year migration from Fabricator to GitHub. So having this dual mode is important for our internal engineers. And one interesting thing you'll see is this red icon here. What this indicates is that this wasn't a great quality prompt and it would have less chance of successful success. So one tool that we built into this was a prompt improver.

The ability to analyze the prompt and make suggestions that the user can accept on how to have a more a higher chance of success. Now, once that kicks off, this is running, you know, background agents can take a little bit of time. We ping the users on Slack, give them links so that they can go ahead and track this. And a few minutes later, in this case, it was 7 minutes later, the Slack notification pings them again. It says, "Hey, the Minion task is done. You can go look at the PR here. You can go look at the artifacts."

So, let's say let's not go to the PR quite yet. Let's jump back into the task completion. I have a view here now where I can see what ran. I can investigate the agent logs if I need to. If this failed, I can retry or have follow-up tasks. Let's say it failed, then I can search through the logs here and start to try to understand what the agent was doing and maybe give a follow-up. But, in this case, it it was successful. We got a PR out of that immediately.

It was a very straightforward one. Our Minion bot co-authors this with the person that kicked it off. Here we have a link Jira. We have the test plan of how it verified. You can see it was authored here by which agent Claude was Minion was running. And it was a very straightforward fix that we got. So, this was a very simple workflow, but it was very much easier for the developer to just say, "Dump in a prompt. Hey, here's a problem the user is having." and get a PR of that as opposed to all of the context switching that they would need to traditionally do.

Right now, the work the workflow for the developers has changed and is changing further. They're spending more and more time in planning and code review because there's so much more code being generated that they're being forced to do it. This probably isn't the favorite type of work the developers love doing, code review. And there's a lot of challenges with that. If people are doing code review and it's taking more time, they're maybe slowing down. They maybe let more bugs in because they're missing it in review because there's much more.

So, let's jump into a few of the investments we made to try to improve that. So, one of the big problems is context switching amongst all the background agents. This could be on PRs that are coming out or the agent itself needing attention. So, we built a tool called code inbox, which was designed to try to help with this situation. It's a unified inbox for PRs that a developer needs to review. And what's interesting about this is it's designed to try to remove noise. So, only bringing out the actionable ones that are directly relevant for a user then when it needs attention, not when it's, you know, sitting there waiting for someone else.

And we put a lot of work into the to smart assignments with code inbox so that we try to find the most relevant

person to review the code both from a ownership and compliance perspective, but also the history of how that person was working, their time zone availability, their calendar availability. and we we try to find the right person and assign and then have strict SLOs that we track so they can see how long it's been assigned, help reassign, do automatic reassignment or escalation if necessary.

And then this does a a smart job of the Slack notifications to devs. So, it's doing thing like batching notifications so they don't see a bunch of noise or accounting for their focus time so it's not bothering them in the middle of it or, you know, their holiday time if they're out. it'll also handle integrating this into teams' existing processes. So, if teams have existing Slack usage with code review queues, we can plug directly into that and inject the reviews at at that level as well.

>> >> Some of the other cool stuff that we built into this one was we tried to understand the risk of the change and we're going to continue to invest in that. there's a much different risk profile to a small change in test versus a change that's in one of our key services. And so, we we try to highlight that here by analyzing the surface area, the the blast, how much that's going to affect, what type of service that's hitting, and then make those estimates so that we can raise that to the developer, so they might put more scrutiny on the review or or bring in another person or, you know, whatever decision they might want to make for a riskier change.

So, in the code review space, I want to move on to a second product. this one we talked about the notifications and bringing context awareness, but this is our product it's called U Review, and this one is designed more at the review help itself. There's a bunch of external products right now. We've all seen them in the market, everything from Code Rabbit to to Graphite. All of those are are trying to solve this problem, and we've played with a bunch and we'll continue to use external ones as well, but what we found is we had a lot of internal context. We had a lot of complexity like the migration between Fabricator and GitHub that made it make sense for us to have a platform that we were controlling the surface area for the comments coming through.

and so what this work How this works is we have a preprocessor for the code, and at that point we have a set of plugins that are going to run. There can be general defect bots that are analyzing it. it can be pulling from best practices or MCPs or other types of information around the organization. and we also have an API so that we can plug in external bots. So, if we were using, you know, one of those external code review tools, we can just plug it into the API here and have it surfaced with the rest of the comments that are coming in to the developer to help minimize duplicate comments or or extra noise.

That then runs through a review grader. this has been one of the common problems that we've seen a lot of low value comments surface from those because they'd rather give something to the developer to do even if it isn't maybe necessary. and we really only want to put the high confidence changes that the developer really needs to focus on, not little nits. And so, this continues through the flow. It looks for duplicates from these different systems and finally categorizes these. Now, each one of these layers we've done evaluations and have different models running based on the the performance that each model has on the type of behavior.

This has been something we've been working on for most of the last year. So, we saw some growth and some

progress in this system as as it matured. One we saw that we were able to get higher quality comments at a higher rate. As as we invested and integrated additional best practices and other rules, we also saw the rate of the comments and the best practices increase while maintaining a high rate of comments being addressed. This is the the specific piece of feedback that we're looking at to make sure that this isn't noise, that developers are actually fixing these and it's not just annoying them. And then here's a screenshot in Fabricator, not in GitHub, where I mentioned we have to have the dual kind of UI because of the two systems at the moment. And so this was a custom one we built to try to have a feedback loop for the developer.

In the code review space, it would it would wouldn't be complete if I didn't talk about the verification, the validation, CI, test. I think that's the other big part that we are really concerned about to make sure that the those mistakes aren't slipping through code reviews more code is coming in. So, we built a system called Auto Cover that we've talked a little bit about in the past. Actually, the author for it is saw him around here somewhere.

This this was a system that we designed to generate unit tests. Now, you might say, "Well, you can just do that with cloud code or or many other products." And you can, but what we found is by really focusing on this project, building a custom agent on top of our internal LangFX SDK built on LangChain, we were able to get a much higher quality type of unit test output. and so at this point we're seeing about 5,000 tests generated and merged per month around the company from this.

And almost a 3x rate of quality versus something that would be generated from your typical generic agent. Now, as we were doing this, we were quite concerned with, you know, bad quality tests, change detector tests, things like that coming in. And so we we built into this a critic engine. So it has both the generation and the critic engine. And we separated that out into an independent test validator that now developers can use independently, whether it's a human generated test or an AI generated test. which is great to help up level the test quality in general and ignore any false confidence that we might get from having the higher coverage.

So I'm going to talk about one more category before I hand it back to Anshu. we've talked about you know, authoring code initially in the code review process, but code maintenance is a big area. And you know, at the beginning Anshu was talking about the toil work. This is where a lot of folks kind of consider toil the heaviest. So as we were looking at how to how to build this out, we we looked at the space, we looked at the messages coming out from other companies. You know, where their CEOs are going up in front of the news or to their their boards or their investors and saying X percent of our code is generated by AI now. And we were looking at that and saying, well, how is how is some of this done? And some of the companies were very mature companies, like like Google or Meta. And we'd had a lot of discussions and seen that they had fundamentals that Uber hadn't invested before. Which was the ability to kind of scale out large-scale changes so that the AI can then build on top of that. And and so we got together last year and we decided we needed to run a big program to create a scalable version of how we handle large-scale change. We call this auto migrate.

we broke this program up into kind of four key areas. we have the problem identification area where someone would looking at a migration or an upgrade and deciding what the risk of the migration is or like what the

surface area of the the changes or how to cut up the PRs so that they they make sense and reduce risk. You have the code transformer piece which could be an agent, you know, we could be using you know, cloud code or any others, but it could be something deterministic like open rewrite which we've made a lot of investments with.

Then it gets into the validation phase where we need to understand how we get confidence that that automated change is going to be successful and that it's not just relying on human review. So, this might be CI or unit test or sometimes even you know, staging or production signal, but this is a key area as we think about it. And then finally, the the area of campaign management was something specifically that we needed to build from scratch. You know, the ability if you have 100 PRs that need to go out to developers for a migration, how do you get those all into the right spot? How do you track those?

How do you make sure those folks are notified? How do you refresh this? And so, this was this became the key of the platform that we called Shepherd. here's introducing the experience with Shepherd. At the surface, it's a web UI where developers can go, migration authors specifically, and they can track all of the PRs that are associated with a migration. It will it allows them to define those simply through a YAML file where they can either give a prompt if it's an agent or they can point it to the script that's going to be handling it. And then Shepherd is going to take care of generating those PRs, refreshing them on whatever cadence you defined, keeping those fresh for the developers, notifying the the people that need to review it, getting it in the right queues, integrating with code inbox, the last product I showed.

So, let's walk through two quick demos of this. One, here's a a PR using one of the deterministic transformers. This used open rewrite. we had Shepherd generate all of the PRs to move our Java services to Java 21. Here we had this PR generated that correctly found the owners, created a limited PR in just the space of the code owners that upgrades it to Java 21, and we can see that generated here in a small change needed for that upgrade.

Here's one more where it's similar, but in this case it's using the Minions platform. It's integrated with it as an agent. So, we have separate tools in our programming systems group to do analysis, find performance issues. A lot of these are generating really good data. one of these we called Dr. Fix. Actually, it's a different one, sorry. It's not about Dr. Fix. but it it identified these performance issues, and it was able to generate a lot of PRs to and or diffs in this case to account for those, run those through Shepherd, have a a standard thing that accounts for how how it was tested, how it was verified, what the developer needs to know to review it safely.

And with that, we've walked through kind of the major deep dive. I want to hand it back to Anshu to talk about the couple last topics. >> All right. so Ty talked through a lot of the engineering investments that we made to make you know, the the Sagantec shift. I'm going to talk through a few non-technical challenges that we're still dealing with. First up is on the people side and the business side. So, on the business side I have a diagram here that it's very topical since the Olympics are on right now. the the leaders when it comes to AI tech is are changing pretty frequently. you know, the the models that are the most powerful for certain tasks, whether you should build something in-house versus use a SaaS provider, these these decisions need to be revisited on a pretty regular basis.

Unfortunately, in a large organization like ours, some of the investments that Ty talked about, whether it's auto cover or auto migrate, these are not trivial decisions to make. We need to commit dozens of people on projects that might be running for months. So, we can't just change our mind after a quarter. >> >> there's there's two things that that we've done to mitigate this. One is seemingly pretty basic, making sure that we have the right abstraction layers in place.

We talked about the Ty showed them the Minions infrastructure. Under the covers, if we need to swap out the model or we need to swap out the technology that we're using, we're we're now able to do so if if a better technology comes around that can solve some of the underlying pieces more effectively, we can do so. but the the second part is just having this this this belief that the tech we're building will likely be replaced with something better in the industry. And so it's really important for us to not be married to the tech that we're building and being okay if something comes along like if the the co-founder cursor talked about the the auto the test coverage system that that might be coming a couple weeks. I'm really excited about that. It might make our auto auto cover infrastructure obsolete and that's okay because at the end of the day we need to deliver impact for Uber.

The second part is another is a people problem. So, a lot of the challenges that Ty alluded to deals with like, you know, historic infrastructure that's been built out over the last 10 to 15 years at Uber. We have some really sophisticated code that we built out and then we have some really I would say archaic code that you know, very few people know about. Getting that technology integrated into to places where AI can reach it is challenging.

E- just getting MCP endpoints set up to reach to different parts of our ecosystem has been a challenge. Similarly, the tech that Ty talked about, like I've seen in action, it's magic. I I ran a demo session with some of my VPs, and in 24 minutes I had four VPs land code for the first time in years. it was it was a pretty amazing experience. They were pretty satisfied by it, too. but our adoption for this technology has been relatively slow. It's been slower than I've expected. And it's Part of it is because we're trying to have developers do something that they're so not used to.

They're used to looking at code and generating from scratch, operating in their IDE, and we're telling them to take a risk by operating in a very different way. In both of these cases, you know, we've tried different tactics to get around this people issue. We've tried a top-down approach, you know, directives from leaders to say, "You must do X, Y, and Z. You must adopt." It's had some impact, as you track, as you folks know. You track a metric, it's going to go up, or it's going to improve.

The more successful technique that that we've applied is actually just sharing wins. So, as we share examples between different engineers, cool things that they've tried that have resulted in in wins, adoption of that technology has has erupted. so, that's been the the tactic that we're we're pushing on now is key promoters pushing techniques to their peers, because those promoters are typically engineers, and engineers trust other engineers as opposed to directors like me.

Okay. I'm going to touch on measurements now. So, we have tons and tons and tons of metrics. I can say with

confidence that objectively AI is having positive impact. our net promoter score, our overall developer experience at Uber, has never been higher. the the self-reported net satisfaction developers have and their productivity has never been higher. The amount of code that we're landing through AI is is amazing. The overall engineering velocity is fantastic. And you can see the graph over here. We see the inflection point where when we were introduced agentic the Minions agentic system along with when the models became really, really good like Sonnet and Opus being introduced, the the delta between developers that are using it very casually versus the ones that are the the the power users that are using at least 20 days a week, it's only exploded. it's only the deviation has only gone up.

So I'm really pleased about this. Now, the the issue is that these are activity metrics, right? These are not necessarily business outcomes. And when we start talking about the costs of this technology, you know, our CFO has has asked me what is the impact of this? Right? I You know, I can't put him to diffs. I need to show him what's the impact on on revenue. I'm sure you folks are dealing with the same problem. We This is not necessarily a solved problem for us. One of the tactics that we're taking this year is to instrument our overall feature infrastructure so that we can time from when a, you know, a design is first created to when an experiment is is launched in production.

And then seeing how we're able to speed that pipeline up. And then speaking of costs, the cost of AI is too damn high. you know, since 2024, our our costs have gone up at least 6x. Now, we'll say that this technology is amazing. It like there's again, no question that it's had positive impact. But it's gone from something that I can self-fund using my own budget to something that I need to ask permission from, you know, the CFO.

What that's necessitated, especially where we went from a model where, you know, it's it's not necessarily cursor's or Entropic's fault that it's going up, the GPU costs are high and memory costs are really high. So, we've had to be more responsible about how we use tokens, how we think about what's the right model for the job, and then helping developers select those models. So again, going back to the the example with Minions, we helped developers think about the right model to form the plan for the for the project. And then, lower cost but still pretty effective models to do the execution.

we don't necessarily want developers to think about it, but we want to be able to help them help have the infrastructure decide for them so that we reduce the friction for them, but then we also optimize our costs. but this is this is something that we continuously have to keep on evaluating and adjusting. especially as new technologies introduced. So, like this year we introduced JetBrains AI and and Warp, which we hadn't introduced in the past, all have their own costing model and all have their own complexities with regards to how developers are using them.