How Cursor Trained Composer on Fireworks: Distributed Infrastructure for High-Performance RL

45 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=UDTR9YUNLUI
https://www.youtube.com/watch?v=UDTr9yUnLUI

SUMMARY

Cursor's recent development of Composer 2, an agentic coding model, marks a significant shift from being an application-focused company to a foundation model company. The model is designed to handle long-horizon coding tasks more efficiently and cost-effectively by leveraging specialized training techniques and infrastructure.

- *Composer 2 utilizes a combination of continual pre-training and reinforcement learning (RL) to optimize its performance for specific coding tasks.*
- *The model's training environment closely mimics real user conditions to prevent it from recognizing it as a "fake" environment, which can lead to different behaviors.*
- *Cursor's approach emphasizes the importance of tailoring models to specific applications rather than relying solely on general-purpose models.*
- *The training process involves orchestrating large-scale GPU resources and managing asynchronous updates to maintain high efficiency.*
- *RL is used to refine the model's behavior, focusing on tool use and correct coding practices, while also allowing for real-time updates based on user interactions.*
- *The challenges of numerical stability and floating-point arithmetic in large, sparse models are addressed through careful design and optimization of training processes.*
- *The infrastructure developed for Composer 2 allows for rapid scaling and deployment of virtual environments, crucial for effective RL training.*
- *Cursor's strategy highlights the potential for companies to develop their own models tailored to their specific needs, rather than relying on off-the-shelf solutions.*

And you need all the infrastructure to run these environments that have to mimic as closely as possible what a user's computer would look like. And it's very important as closely as possible because sometimes the model can actually figure out when it's being run in like a fake environment and a real one and it has like different behaviors during RL than in production. >> Are you seeing it being conscious that it's being it's in a fake environment starts being behaving differently?

>> Yes. Yes. >> Interesting. >> Like it's like oh I'm in a fake environment. I've learned a few tricks to like get a better reward in this environment and let me try them out. Models love to cheat. Is really good at encouraging cheating. I'm delighted to welcome Federico from Cursor and Dimma from Fireworks to the podcast today. Feder Rico, you are the research lead on composer 2 at cursor, cursor's new agentic coding model and dimma,

you spent how many of the last few months moonlighting at cursor in order to support a lot of the infrastructure required to make this gargantuan training task happen. And so I'm excited to talk to both of you today about how the training of composer 2 came together, what hard problems you solved together, and what you think it means for the future of of AI and foundation model companies.

>> Exciting. Yeah, exciting. Thank you for having us. >> Thanks for joining. Okay, let's dive right in. For those who haven't been following us closely, uh Cursor recently announced Composer 2, which is an Agentic coding model uh meant for long horizon coding tasks. Fed Rico, uh up till now, um Cursor was mostly uh enabling uh other people's uh coding agents. uh what was the impetus for cursor to lean so heavily into composer 2 and how existential is it for you to become not just an application company but also a foundation model company yourselves >> the reason why we started the looking into training our own models is you can sort of think about the model as sort of like like a storage drive it has certain amount of bits that it can store in its weights and the idea is very simple you know like we care about only one task we don't even care about coding or programming necessarily. We care about software engineering inside cursor and inside cursor only. And so what if we were to allocate all of the bits uh of information that can be stored inside a model weights to that one particular task. Also, as people may have noticed, composer is order of magnitude less expensive uh than Opus and other like coding models because we can just simply specialize all of the model weights to that particular task and so we can serve like a smaller model or uh something of that sort. Yeah.

>> So, it's about let's make sure every single bit of weight or information we have is dedicated toward the specific problem that we have at hand. >> Exactly. >> Got it. um that seems like it's an almost generalizable problem. Uh DM, I'm curious your perspective. Do you think that every application company should be looking at cursor as a harbinger of what's to come? Like should they all be looking to do the same thing? >> Yeah, absolutely. I mean, we actually generally see it as a pattern of kind of evolution of applications. You maybe start prototyping. You might be using kind of off-the-shelf model to get something running. Maybe do some prompt engineering, figure out how your harness works. But the most kind of leveraged attribute of your application is actual usage of user data or particular specific aspects of how the application works. Maybe some aspects of your harness, which tools do you provide, how the application works, kind of really important bits which are important for your application. And the right way to capture that you can do a little bit of that through prompting but really the right way to do this is craft your model to act in your environment. Yeah, absolutely. Like there are certain tools the agent calls that it's very hard to succinctly describe exactly the behavior of that tool to the model and you know with just like post training we can bake in the optimal way to use those tools like composer we we do serve a prompt to composer but I I think the way we are training it it would work even without a prompt and it would know what to do just because like we are intrinsically pushing the model to like the right direction of how it should act throughout our training >> basically there's kind of like upper bound of like how far you can get is prompt engineering that if you want to craft really great AI products you have to go through kind of fine tuning and uh in influencing model behavior that's kind of one reason I mean reason number two is what the recommendation is kind of cost trade-off or like speed trade-off like the way we kind of view it at fireworks is that when you're trying to do optimization you had this like threedimensional trade-off between quality speed and cost and uh you can go quite far and we doing it with all the customers Initially you can go quite far with just optimizing infrastructure but when you start getting into model training you can really push this trade-off much further and you can get better model at fraction of the cost running much faster and

you know composer is a great example of >> can I push on this a little bit I want to ask if this approach is bitter or less in pill and we were we were actually all talking about tab 9 on the walk-in I'm remembering before the LLM era there were these like small specialized coding models and one of the things that was I think surprising to to a lot of people was as you've scaled up, you know, you scaled up just training on the internet and a lot bunch of English text and other languages, actually the models themselves got inherently better at coding as well. And so at least the trend line I've seen so far is just like bigger models perform better on everything including on coding. Is what you guys are saying does that go against the the grain of the bitter lesson?

>> I think no. But one one sort of like thing to point out is that the big models trained by the labs train on a lot of code as well. like code is one of the main tasks the labs are interested in pushing and so they don't just generalize to it. They're a bit specialized as well. I think uh for our case actually you know if we believe about the bitter lesson we are just pushing very hard on the data dimension and we know that the models inherently have finite capacity and so if we want to saturate all that capacity we need to scale data and in order to ingest more data we we need to like free up the weights from distractions the model may have >> okay got it super interesting okay let's dig into the training of composer 2 you launched a couple weeks ago immediately grabbed attention. Strong benchmark numbers, much lower cost to to run imprints on. What's the short version of how composer 2 works and and what you guys did to make it so performant?

>> We started from a very strong base which is Kimmy 2.5. That's like a one trillion parameter that's 30B active. So very very sparse. Actually, we sort of like looked at the stack and realized there are like two axis. So mainly composer one was just pushing on one of these axis which is reinforcement learning but composer 2 pushes in two different axis. One is continual pre-training and the other is reinforcement learning. So the thing that made composer 2 very good is pushing in both of these directions. So we started off the training run by doing lots of mid-training on code tokens almost sort of pre-training scale actually and then coming out of that mid training run we took the checkpoints and we did very large scale RL on lots of lots of tasks.

>> Okay. And then the premise here would be because cursor sits in the middle of so many interesting coding tokens you actually pretty uniquely have access to data to be able to train at almost pre-training scale. >> Yeah. Why not pre-train your own model? Then >> we just think about our approach from top down instead of bottom up. So like how do we get a model that's useful to users in the least time possible if we were to start from the bottom sort of figure out how how we do pre-training and then scale it up to mid training and then okay now we figured out mid training or we do reinforcement learning. That would take a very long time to get a model out to our users. By doing it the other way around, we were able to give a useful model to our users in very little time. So hopefully you know like next composer versions are going to be our own model instead of basing it off an open source base >> and what is the model roughly learning in the kind of mid-training step and what is the model learning uh in the post-training step for you? Yeah. So in mid training it's sort of just kind of learning about libraries that of code and learning about specific code patterns that are very common like just world knowledge as well. There is like web data there as well. And this is sort of just creating a wider distribution that then reinforcement learning can sharpen on. And so during reinforcement learning you know the model gets to play directly with the cursor arness. And so it gets to learn about the world the model is going to live in for the rest of its life, right? In in some way. And and so then during reinforcement learning, that's where it learns how to call tools properly, how to navigate its environment, how to write correct code.

Because during mid training, it it learns how to write code. That doesn't necessarily mean it learns how to write correct code. We try to train on code that is largely correct, but the model doesn't actually know how to differentiate between the two. While in RL, one of the key things that we are doing is we're kind of tuning the feature of the model saying, "Hey, now you got to write correct code all the time." Exactly. >> Interesting. And is the is the model after mid-training is that similar to the model you guys have on tab autocomplete or is that a different core competency? Yeah, I mean it's uh yeah I think I would put it like that because like during mid training we are just doing next token prediction you know like how well you predict the next token and then the token after that. So yeah, >> so why not just post train on your tab autocomplete model then? Why mid-rains the different models?

>> Yeah, I mean tab is a very small model because it's like a super low latency model um as you want it to be very fast. So like the core two distinctions about the base models here is that tab is like small and and uh composer is quite large. >> I see. I see. Okay. So it seems like a lot of the focus of what you guys did for composer 2 was this large scale reinforcement learning run. Can you break that down for us? Like what goes into that and what are the various hard problems you solve along the way?

>> When you do a it's quite different from like from like pre-training or m training because you're not just trying to predict next token. You're actually running the entire harness like the entire experiment. You're letting the model act in the environment. See how how it performs for a given rollout. That's the terminology which is called rollout. and kind of assign it reward whether it did something correctly or not which might be some using LM as a judge or maybe something verifiable like does this code compile or something like this which actually means that compared this regular training you need a bunch of other components like you still need large scale training you still need to orchestrate tens of thousands of GPUs to do forward backward propagation do all the stuff you do in mid- training and pre-training but now you also need to orchestrate a bunch of environments you need to run model inference because when you do this this roll out you're effectively running like real cursor session in some sense right so you have >> notes is like a forward pass >> uh no roll out is basically your entire like agent session from cursor right so basically means it might take something like 50 turns model will take take initial prompt then decide to call some tools you want to execute those tools then model generates a bunch of other code kind of entire session which you when you interact with agent and cursor right you you kind of simulate this entire session as a part of your training run you get to final reward and use the you get use that signal to now go back to trainer and kind incorporate it in the model weight. So you have this kind of very big loop update loop which is uh very heterogeneous right because you have all these like different components working together and now you're trying to orchestrate all of this to work efficiently and work with high throughputs because GPUs are expensive and you want to get your model trained quickly in in economic fashion. So that's by by itself is like very interesting kind of problem and intersection of algorithms and infrastructure because there are a lot of trade-offs how you can kind of co-optimize and co-design the system.

One aspect is kind of people call about like a synl of pipeline. The idea is basically okay you're trying to update this model in steps right so you have your current model version and you're trying to do a bunch of rollouts with it. What does your trainer do while you're doing this rollouts? Right? Like n approach would say that okay now I'm going to stop my trainer. I'm going to do a bunch of sessions and those sessions might run for like 5 10

minutes or even longer if it's like longer horizon tasks. I'm going to get those uh outcomes and now I'm going to pause my inference. I'm going to go back to training trying to do updates. That's like very theoretically algorithmically robust because you are not precisely simulating everything but it's very system inefficient because half of your capacity is sitting idle all the time.

So you can uh do all the clever like algorithmic tricks allowing you instead. Yeah. >> Yeah. You can you can like kind of pipeline all of this. So imagine this as a gigantic like factory, right? You have this like trainer building and you have rollouts building. They're always turning, right? So rollouts always take like latest model version and try to do new sessions and kind of simulate new agent sessions and trainer always takes new outcomes as they come and try to compute updates. So everything is moving along all the time. The trade-off is that why I'm saying that algorithmically it's different because now by the time you finished some test roll out in your kind of simulated environment maybe model weights already updated on some other data. So you have this kind of staleness like delay between how quickly model can learn uh updates because by the time you kind of process or some uh interaction session with a simulated environment your model was changed and that introduces interesting training dynamics and there are clever ways how you can address this but the flip side of that is that your all your GPUs all your computers kind of load it and chiming all the time which actually you're using more more flops and to your bitter uh less as an example. Yeah, you you have like higher compute efficiency.

You can get to a better model in smaller amount of time. Yeah. Maybe you're losing a few% from being asynchronous and not doing like perfect mathematical updates, >> but you way compensate for that by effectively not leaving half your capacity on the table and there are a lot of kind of depths and interesting interaction in that part. And we're very serious about performance at cursor because unlike the big labs, you know, we have tens of thousands of GPUs, not millions. And so yeah, we do all sorts of tricks to make get the most out of GPU like we train in production with FP4 even we work with fireworks to like push on inference as well cuz the thing about infrastructure is just like it's just inherently more complex than pre-training cuz you need all the pre-training infrastructure. That's just like one of the requirements. Then you need all the infrastructure to run these environments that have to mimic as closely as possible what a user's computer would look like. And it's very important as closely as possible because sometimes the model can actually figure out when it's being run in like a fake environment and a real one and it has like different behaviors during RL than in production.

>> Are you seeing it being conscious that it's being it's in a fake environment is starts being behaving differently? >> Yes. Yes. >> Interesting. Like it's like oh I'm in a thick environment. I've learned a few tricks to like get a better reward in this environment and let me try them out. >> Models love to cheat. Is really good at encouraging cheating. >> Yeah. >> Yeah. And then we need a really efficient inference. So this is really important. So there is like actually this kind of myth that during you spend more way more inference flops than training flops. This is sort of like just because the open source inference engines are very unoptimized instead of actually being a property overall.

Roughly the same ratio is kind of the same. In theory, if you push the GPUs to the maximum, you should have onethird of your training GPUs allocated to inference, right? Because training is effectively three forward

passes. You have the forward pass, you have the data gradient, the weight gradient. While if you really hit the critical batch size on inference, you should only have a single forward password worth of flops. >> So that's why you guys use fireworks instead of using an open inference engine.

>> Yeah, I mean the other alternative is we would build one inhouse, but you know we have finite engineers like everybody else. We would like prefer to have engineers make training more efficient and more precise rather than like spin up like a inference effort. Yeah. >> Okay. That's super hardcore. What about I think you mentioned in your technical paper paper paper that you were doing this in a kind of globally distributed way. Why globally distributed and then what makes that hard?

>> Yeah. Yeah. Well, there are various reasons. One, you know, like this very large contiguous clusters are hard to find in the market. And so what we can do instead is we have one cluster that's going to run all of training. You know, we can't do global training cluster. But then the inference component of reinforcement learning we can globally distribute that across small clusters all over the world. So I think for the composer to run we used four clusters in total that were all over the world very far away from each other and we even used some of our production traffic when it was least used. So like we had composer 1.5 the previous model served and when it was least used by people we just grabbed some inference GPUs and we put them to speed up training and so we can do these sort of things uh and sort of easily scale up our training ground without having one large continuous cluster and the thing that enables it maybe Dimma can talk more about >> kind of like to to reimulate what Fed said is basically our training is like very heterogeneous right and by leveraging heterogeneity how different components like what infrastructures they need. You can actually drive efficiency and you see this pattern kind of across the board everywhere.

Specifically for for training you have all this like highly interconnected clusters you need high speeded network kind of need to work in lock step. So those clusters are expensive right and actually it's really hard to find big ones right basically at the scale with which composer was trained finding like 2x larger cluster is like significantly harder than finding the current size one. And that's why if you can disagregate these components and put them on different places, one you don't need to find such a big cluster. Two, you can actually find like different trade-offs of hardware because for inference you don't need that kind of wide interconnect. You can have smaller groups of GPUs interconnected together.

You can have heterogeneous types of GPUs. You can have different generations of GPUs. You can kind of play all these games games optimization. And finally like inference. It's much easier to scale up and down uh as you go. And yeah, it's very convention like when you have off peak covers, you can view all your kind of inference pool as one set of GPUs >> serving production traffic for real users or serving simulated environments for RL purposes and kind of bal balance between this. Of course, it's a very interesting systems problem. The recommen one one terabyte training step takes somewhere between like 5 to 15 minutes.

So it basically means like every like every 5 to 10 minutes you are producing like one terabyte new snapshot of weights. So the question is like how are you going to ship it to a different cluster on the other side of the world very efficiently right and you want to like do it quickly because remember you don't want to get this staleness to

get out of hand. So I think that was probably one yeah the the kind of the the most fun part which we figured out together is that despite know full full model being like one terabyte not all the weights change every step right because RL does a lot of very like precise adjustments especially the training going along. So actually there are very kind of regular patterns in like which subset of weights gets changed maybe not all of them change every time. So if you were to look at like how my model changes within one training step like after 10 minutes there is relatively small delta between those. You can write write a compression algorithm which basically leverages this property and now you end up with kind of like database systems problem which is okay I have my delta and I just want to like ship it across across the world. My delta maybe is like 20 times smaller than was shipping the full model with and that and this makes it practical but of course now you need to build all this kind of machinery from storage system.

So full snapshots and deltas and recovery and reconciliation etc. We were able to build it kind of in lossless uh fashion basically means that like you always end up with bit equivalent model in the other side. So you don't need to worry about any mess aspects of this and you can do it really fast. You can you can do it under you know under a few minutes even in the worst conditions. Usually it's under a minute and most importantly you like pause only for like maybe 30 seconds to swap the weights in your actual inference. also like fully like saturated the band the egress of the cluster by like sharding the upload and the download as well.

>> So you can do all this like system tricks to bring the stand down. It is it is quite a few complexity but you can kind of abstract it out and just make it work great like it doesn't interfere with your training algorithm and on the flip side you have this kind of power to disagregate to leverage other clusters to do that and that kind of goes against kind of conventional wisdom of how you should do RL infrastructure because conventional wisdom is like you okay you're going to have this really huge one cluster connected with RDMA and it's going to be very expensive and you're going to probably spend you know maybe you're going to allocate one S to training and two SS to inference and sure if you have very expensive network it's much easier to copy this one terabyte quickly but now we have like three times larger cluster now if your inference engine is more optimized then maybe you going to save oneird of that cluster in terms of GPUs anyway because you're just more efficient and you can take you know half of this cluster somewhere else in a maybe cheaper hardware in a different region so your cost comes down quite a bit >> I love that you guys are just grinning as you describe this because it's like it's so hard and this is like a systems engineers dream, right? And so it's just like a it's an amazing amazing system you guys have.

>> We spend a bunch of nights working on this. >> Yeah, you look like you spent a long time a lot of time together. What about I mean you mentioned at the beginning uh that Kimmy is a very large sparse model. Does that make the RL run tricky in any way? >> Mhm. Yeah. >> How so? >> Well, when you do inference, you're essentially doing like a forward pass is just kind of like auto reggressive. And in this forward pass it produces like log probabilities of like the tokens it it has sampled. When we ship back the like uh generations of the model to the trainer we have to rerun that forward pass because as we mentioned we are doing asynchronous training. So the model that has produced the pass may have been like actually a few steps behind what the trainer is at and so we have to rerun that forward pass and reproduce log probabilities. Now the problem is in theory this log probability should be exactly the same if it's the same model version but even with the same model version you get slightly or sometimes very different log probability values for the same tokens.

So this is often called what like a numerical mismatch for inference. You hear this about all the time these days for mix. >> And why is that? Why does that happen? I mean primarily because like fundamentally floating point arithmetic which is doing is is nondeterministic. So if you >> sorry floatingoint arithmetic is nondeterministic. >> So you know we learned this code that like if you take a plus b plus c right uh and like c plus b plus a it's going to be the same result.

>> Uh if you're doing this with integers with whole numbers on the computer that's going to be always true. >> If you're going to do it with floatingoint numbers which are actually like approxim approximation numbers you have this like mantis and exponent etc. a plus b plus c and c plus b plus a is going to give you like different results or even like a plus b and b. So basically like fundamentally it's accumulation order of like all the operations which models do is basically like multiplications and additions and like addition order matters to your final result. It's all like small differences but they get a amplified through like millions and billions of operations. So when you do inference of models usually it doesn't matter that much because you pre-train your model you're actually pretty robust. If you like flip some bits, it's still going to produce you like good results. Your benchmark is not going to change. But ARL in particular, because you're using this very very like weak signal to teach the model, the noise from this numerical differences can make or break your training. And that's like particularly important. And it again, it's an interesting intersection between like algorithmic and systems part because you know you can write a beautiful mess and it just doesn't work in practice. There are ways how you can drive this difference to pretty much zero. There are always like batch invariant ways you basically you can be very very careful and write all your GPU kernels so they always add numbers in the same order. So you always do like a plus b plus c and not a different order. Uh it's possible but it always has like trade-offs right.

Basically your like your system becomes maybe like 2x or 3x slower. Again it becomes an interesting trade-off like okay what is the 10% of slowdown which we can take or in practice actually few% of slowdown we can take to address 90% of this difference. That's you know the right trade-off which kind of we we find together through iteration and you mentioned that particularly for and sparity is hard the reason for that is that like the way work is that you take your activations at every layer and you would run it through gating layer but you basically decides okay for this token I'm going to run out of 384 experts I'm going to run this eight right so it's going to do like some mess and like top eight scores those eight experts going to be activated other ones will not be activated for this token this operation ition amplifies your small numerical differences quite a bit because maybe your hidden states were like difference by like fifth digit after dot doesn't really matter but this difference made it so you picked expert number seven versus expert number nine as kind of as a cutff and suddenly you went and like activated totally different part of the model and your difference got amplified quite a bit and my models by definition are like very more sensitive to this mismatch again when you do inference So when you do kind of regular lot it usually doesn't matter in average out but now if you're trying to basis model learn this difference is huge because your inference activated expert number seven now in your training you're trying to like update expert number nine which didn't even contribute to that during inference.

>> So were you guys handwriting GPU kernels then to help get around this problem? >> Yes. So you can again you can address a lot of this throughput and there's always trade-off specifically for me you can do this interesting trick which people call router replay but basically you can have your inference just pass extra information to training and say that hey I activated expert 7 for this token this very small bit piece of

information is just one integer saying that like okay this is the expert that you activated so trainer can be aligned with that and a lot of this numerical alignment is basically you know doing tricks like that matching quantization levels matching kernels etc.

>> to drive the divergence between training inference implementation down and that makes huge difference in between you know your run maybe divergent completely or being you know multiplex less compute efficient because you'll need much more data to address to this mismatch >> I'd love to maybe chat a little bit more about the RL kind of recipe can you say a word about the reward signal yall are using is like or you can't okay can't say got it top secret stuff top secret stuff okay that makes sense like it seems like there's a almost like the equivalent of learning in sim is simulated rollouts versus like you have so much actual user data that you could be learning on why not just do RL on your your actual user data and your actual user harness versus doing this in sim >> yeah we are also doing that so that's uh what we call real time RL >> okay >> and uh we use the same technology to do like the inference weight sync with like fireworks to do this we find like user signals where the user was happy or sad about a particular model generation and we are able to update that model live and so then ship a new version of the model continuously every few hours.

We're working on decreasing that time. Actually at some point we'll have to increase that time because as the horizon of the model gets longer and longer. We'll have to reextend that time. It's like an interesting play like right now we are trying to decrease the time for stability because we were figuring out the right hyperparameters and then after we have figured it out we have to reextend it again just because we want to lengthen the horizon of these models. Yeah.

>> Do you need to do any of the kind of like pre-training simulated RL you have so much actual user data I imagine that's just like much more valuable to to train and tune on. Like why not just go straight to the online RL step? Why why do you have to do the the offline RL? >> The online RL currently is pretty inefficient. we suffer from this problem that the GPUs are offline for a long time essentially >> and beside that there's also like different trade-offs both in terms of efficiency and user experience. Yeah, >> if you do simulation, you actually do multiple rollouts from the same prompt, right? You effectively take a task and you ask a model to do 16 tries at a task or like 128 tries on task like different rollouts from the same prompt. Some of them are going to go go well, some of them are not going go well. And by doing it multiple rollouts in parallel, you are able to get much more precise signal. Maybe like you know maybe model is very good and it's does it well 90% of the time, maybe it's not very good.

losses like GRPO like group group policy gradient like kind of work by doing multiple rollouts at the same time. If you're doing online, you have only one rollout coming back and so so trade-offs of like how you do it algorithmically different and most importantly if you simulated rollout goes wrong it's not it's not wet right I mean you just you know maybe spend some time on GPU uh if it's actual user you you have much higher like minimum bar on that because effectively you're doing AB test right so if the model produces something weird like that's a bad user experience >> yeah okay so you can go off policy more often when it's not a real user because you can like you experiment with like crazy things and without affecting the user experience >> you can do a lot more rollouts you can do gpo >> um and then you can basically like bootstrap some level of performance

that's good enough to even put in front of users okay >> yeah like we teach reasoning through like the offline which is actually like called online offline is more like dpo kind of technique sort of reinforce kind of is online and then we there we like teach the reasoning to the model we give it some kind of input of the behavior should have. Uh we try to give it new information about the world and we teach it tool calling and then we put it live to users because you could imagine like if the model is bad, users don't want to use it, they're not going to give us any feedback, right? So the model has to meet some kind of bar to even like be put into online rail. Like we want to be really happy with the model and this is the model we ship. That's kind of the paradox of online rail or how we like to call it real time is that you know we can't use this to really like create the model from scratch because users need to be using the model and so it has to be good already and we can only make it better. Yeah.

>> Yeah. It's kind of like cherry on top to really get this super delightful experience for sessions. Hopefully one day it will be like big big cherry you know. >> Yeah. >> Yeah. that Dan Roberts presented at our conference last year. I think you were there. It's like traditionally was the big cake and the little cherry. >> Cherry. Yeah. >> Little cake. Big cherry. >> Yep. >> I'm curious uh the the Andre Karpathy line of like right now RL is, you know, still super inefficient. You you do a big big long roll out and then you kind of get like, you know, a little bit of information at the end and it's still like I think slurping bits from a straw.

What do you think? And have you have you been able to figure out how to get more bits out of that path? Uh, I can't talk about that. >> Okay. Okay. Got it. We're back on We're back on the secret stuff. Good. That's how I know I'm asking the right questions. You mentioned the roll outs are a few minutes at a time. It seems like the whole field is pushing towards making like long horizon agents, agents that can work for for a long period of time, uninterrupted, and generally not failing. I love that meter scaling chart. What goes into into the RL process to try to get the agent to run for longer?

>> Several things. So one problem about uh sort of like reinforcement learning is that the longer the trajectory is the harder it it is to do credit assignment. So you can imagine like we are giving thumbs up thumbs down at the bundle right at the end of its work and sort of like to simplify the problem is like the model asks itself okay where did I do right and where did I do wrong that's basically the the problem called cray assignment. It gets harder as this gets longer. So you have to do a bunch of tricks there. The other problem is just like you run out of space, right? Like these models have a finite context window and at some point they're going to reach that. So actually the way we solve this at cursor is uh we put compaction in inside the loop. So we call this self summarization. So during reinforcement learning the agent actually learns how to continue and go on forever. So in practice our model is like a 200,000 context window model but in reality it can go on for millions of tokens and just because of this ability that it can summarize its work and then take that summary to restart its context window while still trying to accomplish the task and through because pushes the model to do uh things correctly towards the goal at the same time jointly we are training the model to produce a good summary And then we're training the model to listen to that summary very well at the same time. Um, and so this is kind of like a continuation to reasoning almost. I feel like I find it fascinating because uh I mean usually context management consider like part of the hardness, right? In this case, you're effectively co-optimizing like how part of the hardness and like model itself work together and throwing all of that in the optimization loop. And we've seen this again and again in AI that like the more you throw computers a problem, the more you can solve the problem end to end. The magic of computing bitter lesson works and you get much better system which can

work together.

>> Totally. Totally. Do you think every company is going to be rlinging their own harnesses? Like do you think that every company has the same shape of problem as cursor? >> If they are using AI and they're like producing lots of tokens and they have a product to optimize against, I think it's it's like the right move and the the right direction to train models. >> Yeah. Yeah. Interesting. Interesting. Um, and so, so it seems like most of the reinforcement learning you guys did then was on the kind of like the harness tool use part rather than on the get good at, you know, complete the next token for code. Is that roughly the pattern that other founders should have in mind when they're trying to think about where should I use reinforcement learning? So like if you're trying to get an agent to perform tasks with tools over long horizon, you need RL. If you're trying to create a model that's good at summarization or a next token or whatever, you probably don't need RL. Is that a good framework for when you need RL?

>> I think RL fits everywhere. So even for tab, we use the personally this is just my theory and it's not backed up by anything. When you pre-train a model, they're just the models are just in ingesting the totality of human knowledge. Let's say you're training a model for math. The model sort of like learns all the math on stock exchange. the model when it's presented with a math problem and this is a model that hasn't gone through a real the model is needs to wonder what kind of person it is. Is it the expert or is it the student that's trying to learn? And so one of the things that I think happens during is that we are tuning this knob letting the model know hey you are the expert you need to do things correctly.

So that's like one thing that happens is we are sharpening this distribution sort of like a has a few phases. So like there is the very first phase where the model learns and becomes very good very quickly and then there is like a second phase where like it takes a lot of compute to continuously improve the model and like you see the model starts reasoning and have this pattern. So in the very first phase of the curve I think that's where we're just tuning the knob telling the model hey you should do things correctly here and so in the small compute case is also very useful just to let the model know that it has to do things correctly that's sort of like my case to this >> yeah I mean second that I mean you we see this pattern across many use cases you know we helped RL fine tuning generally for many customers and we see this usually you kind of continuous pre-training basically m training like regular supervised fine tuning is simplifying you can say it's transfer of new knowledge kind of in abstract way and RL is kind of sharpening the behavior or like particular qualities you would you would want from from the model and usually you end up needing both and even to your example of summarization it's actually like RL may very useful for this because sometimes it's if you want particular style out of summarization right it's really hard to like come up with examples of like good and bad summarization it's actually really describing this precisely but if you use for example LM as a judge right you can actually say very precise rubrics you can kind of prompt saying like okay this is a criteria how I'm going to evaluate whether summarization good or not throw it into RL loop and let the model kind of experiment with different summarization styles figure out what you actually want from it while maybe another LLM kind of evaluated whether it's matching particular rubric or not and that's kind of type of pattern which you see a lot not just in coding like >> I Okay, I'm going to ask this question to Dimma because Fed Rico is going to plead the fifth. Um, you've mentioned LLM as judge a couple times. Do you think that ultimately companies will be more successful having like experts handexamining RL rollouts and you know hand coaching the model behavior in some way or do you think LLM

is judge other automated rubrics are likely to get us there?

>> You don't really like put experts directly in judg judging rollouts. I mean that would be some kind of like I mean real time RL if it's actually users or like some form of I don't know like RLF for DPR I mean generally the more verifiable your reward is uh the better because it allows you to like scale the compute and just get better outcome in some case and by verifiable basically means like okay can you automatically produce it without the human uh of course if it's like mass or coding and you can craft something like very deterministic that's the best the reason why LMZ a judge works is that it's actually it's kind of generator discriminator distinction like it's much easier to judge. I mean the same for humans, right? It's easier to judge than to create >> a VC.

>> Yeah. No, no implication there. But yeah, it's much easier to judge and you can craft precisely like different criterias you want to rank some answer and you see this pattern where you might have like very complicated eval from multiple aspects, right? Because if you dump multiple aspects to a single LM it might be get confused how to judge, right? like you you might break it down. Okay, you're going to judge rubric based based on style based on like some different aspects like based on factuality kind of really craft this rewards. Some of them will be the genius, some of the BLM based and that's what guides your model behavior then you just turn on turn on more computes and see the graph go up.

>> Do you think that we're going to see RL be more effective in the harder to verify domains? Like do you think LLM is judges sufficient? That's one of the techniques you would you would you would start right ideally you want to figure out what is the actual outcome what is the actual metric you want to get right so kind of trying to approximate this is one way trying to get bigger simulated environments is another right like if you can simulate more of your product if you can simulate more of your environment usually you have like final metric which you care about it's just harder to capture if you can figure out how to capture this that's great and to your point about you know experts I mean experts are still still needed right because crafting this task can actually encoding the product experience you want that's that's what matters right we went through software 1.0 2.0 3.0 Right?

Because of crafting software directly. We went to crafting training data. Right? Now you're effectively crafting the evaluation rules. But that's still very important. You need to look at examples. You need to look at the data. You need to look at like where your product fails and how to nudge the model in the right in the right behavior. >> I want to ask about RL environments which is maybe related to what you were talking about. It seems like there's been a huge explosion and just the revenue scale that some of these RL environments companies are reaching.

What do they provide that's actually useful? Because I think cursor for example, you have so much data on like how your customers are actually using your environments. What do the RL environment vendors offer you um on top of what you already have? >> Yeah, we don't actually use any of the environment vendors. I think so it's very difficult to construct working environments. It's a valuable product for people that do not like have access to this. However, uh for coding particularly, there is like a very large amount of working coding environments available to everybody. That's GitHub, right? You can go in and maybe like you can have a model like just install all of the dependencies for a repository and that's like a working environment. I think a lot of the difficulty

comes from the infrastructure as well. So you can imagine that uh a environment that's that works well for a particular task may need like services up. you're like making a change that um let's say like a database migration to test that is actually working you need the database app right and so those kind of things are very tricky I think like these environment companies are like quite helpful for that that kind of stuff there are kind of two aspects on to this right f first like if you look like frontier labs right they're trying to build generic model which is good at everything right so they need to cover all these different tasks underneath package up in one model and kind of encourage it to generalize Right. Uh so that's that's kind of one part and that's that's very helpful right in cases like composer right you have you have your actual product right and I think that's what also kind of believe at fireworks like yeah if you have your actual product you should you should do well against it right >> the most powerful environment is your own product >> exactly because like that's where your model actually will be used and and of course uh if you have frontier lab you're not going to do it across all the products right but if you're if you're trying to build the best model for your product specialize and tailor it we should just use your production environment of course you want to isolate it properly, right? You don't want to model wreck havoc on your production database. You want to clone it, etc. And there are some, you know, tools from environment companies just like from general infrastructure which makes it easier. But generally, you want your RL environment to be as close to real production as possible.

>> And that's what you know, as an example, we see it is if you look at kind of toy RL examples, toy frameworks, they always start like, oh, there's this like toy environment. and I'm going to spin up a Docker container and run everything in it, which is great for like toy examples if you're trying to teach model how to play Atario or whatever, right? But if you're actually transition to like production cases, you can't just put your real real production application in the Docker container. And we found it pretty early yourself like working with Manifox like in case of Corser trainer on their side. Some other customers we run trainer on our training platform but for environments we actually default to running them on the customer side because that's where the actual implementation is and you you effectively have the same setup of trainer even if it's part of fireworks platform or on the customer side calling the actual production environment not trying to kind of wrap it and componentize it.

>> Yeah. >> On the on the hosted platform because that's really hard and that's introduces differences. >> Yeah. Like I mean what we call a environments is really three components. one is the harness. So the harness is like where the model can submit tools and the tools get executed and the second thing is let's call it the like a kind of operating system right. So like what is the actual like world the the state where the model is like interacting with and then there is like the reward component we need which needs to check at the end that the work is is done correctly and generally the harness is pretty portable. You can take the harness and put in in many different environments. The thing that's key is the operating system and to replicate this just normal containers don't really work very well. So at cursor we actually built like a whole virtual machine stack and so we can spin up like virtual machines really quickly and it has to be super bursty because you can imagine like we are asking this system please give me 100,000 virtual machines now and it has to come all come up and um um yeah >> awesome I really enjoyed this conversation today I think cursor is such an inspiration in what you all are doing as a company towards going from application company to really a frontier model lab And I think the work you did with Composer 2 really leads that charge. So really special to hear about it. And then Dimma, really cool to hear about the hardcore infrastructure problems actually that the two of you solved together in the trenches

over many, many late nights to make it all possible. So thank you. Thank you guys for joining today.

>> Thank you so much for having us. >> Thank you.