What does the next training paradigm look like?

19 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=20P5-KQXF_Q
https://www.youtube.com/watch?v=20p5-kQXF_Q

SUMMARY

The discussion revolves around the potential of training AI through reinforcement learning (RL) in diverse environments to achieve artificial general intelligence (AGI). It highlights the challenges of sample efficiency and continual learning, emphasizing the need for innovative approaches to enable AIs to learn from real-world experiences effectively.

- *AI labs are betting that extensive RL training across diverse tasks will lead to AGI by creating problem-solving agents capable of long-term learning.*
- *Current models are significantly sample-inefficient compared to humans, but improvements in RL training are enhancing their problem-solving capabilities.*
- *Continual learning may not be necessary if in-context learning becomes sufficiently advanced, allowing AIs to handle complex tasks without needing to update their weights.*
- *Progress in AI for computer use is slower due to the lack of high-quality, grindable environments compared to coding tasks.*
- *The challenge of training AIs in non-stationary environments requires innovative solutions for effective learning from real-world interactions.*
- *Techniques like on-policy self-distillation (OPSD) could improve continual learning by allowing AIs to distill knowledge from experiences back into their weights.*
- *Speculative ideas, such as "dreaming," propose that AIs could simulate experiences to enhance learning efficiency.*
- *Future AI development may focus on leveraging real-world deployment experiences to continually improve capabilities beyond initial training.*

So here's a big research bet
that all the labs are making. They think that if we train AIs to
accomplish millions of verifiable tasks across thousands of diverse
RL environments, then we will have basically built AGI, because this kind
of training will have created a kind of problem-solving agent: the kind of thing
that can make progress on open-ended tasks for weeks on end in the face
of errors and mistakes and ambiguity. And the people who are optimistic
about this vision will say that all these things we talk about as the
fundamental deficits in the current training paradigm — for example, the
data inefficiency of these models, or the fact that they lack continual
learning — can just be steamrolled if we scale training more, in the same way that
all the fundamental research problems in natural language processing collapsed
when we threw enough compute into LLMs.

So in the previous essay, I talked
about how these models are one one-millionth as sample-efficient as
humans, and the people who are in favor of the current training paradigm will
say, "Look, that might be true, but this is only true during training." Training is this one-time cost that
is amortized across billions of sessions that a model will experience. What really matters is how smart
and general and sample-efficient the model is during a session, and
this has clearly been improving as we've been doing more RL training.

AI agents are able to solve more
and more ambitious problems over longer and longer time spans. Anybody who has used these
models for coding knows that. Similarly, people would say, look,
continual learning — this capability I keep harping about, where the
model's weights get updated based on what it's learning from deployment
— may simply not be necessary. Because if in-context learning gets
so good across longer and longer time horizons, then you don't need to
distill everything the model is learning on the job back into the weights.

People often say that their
employees are not net productive until six months or more on the job. So clearly, online learning
is necessary for competence. But what if you could just fit those
six months into the context window? There have been tons of architectural
innovations that dramatically increase the amount of information, or the amount
of context, that a transformer can store. And why not think that, with a
couple more years of progress, we might have what feels like
infinitely large context windows?

Okay, so before we discuss this research
bet a bit further, I want to step back and ask a completely tangential
question, which I find actually very interesting and confusing about
the nature of current AI progress. Why has progress on computer use been
so much slower than other domains? Computer use is so clearly verifiable. You could ask a question like: did the
desired Etsy item I ordered get delivered? Is the venue for an event I'm
trying to organize booked?

Have my taxes been submitted? So isn't it weird that computer
use has been making so much slower progress than coding and math and
these other verifiable domains? I'm sure there are many reasons for
this, and one of them, of course, is the fact that the models are
exposed to far less high-quality multimodal data during pretraining. But one reason that I think is actually
quite underrated by people, and which I think reveals the canyon walls against
which this river of AI progress will only slowly chip away, is that it is not

enough for a domain to be verifiable.

It also has to be very grindable, in the sense that you have to be able to run lots of parallel rollouts against a deterministic and replayable simulator, and you have to run those rollouts from the same starting point. If you're trying to make a model better at coding, you can define some container that has a software repo with some missing feature that you have tasked the AIs with creating. And then you have a thousand parallel agents go at the problem, each of which has an identical copy of the container.

But this doesn't work with computer use, at least not trivially. You can't just have a thousand agents go try the same checkout flow on Amazon to get better at using websites, because Andy Jassy will find your bots and shut your ass down. You can solve this by making clones of Slack and Gmail and all the other common applications and websites. But at least currently, this is a very labor-intensive and unscalable way to build environments. Of course, once AIs get good enough at coding themselves to build these clones with extremely high fidelity, then I'm sure computer use will make quicker progress than it is right now.

And you're also killing two birds with one stone with this kind of procedure, because getting AIs to rebuild whole applications from scratch is also a great RL objective for coding. So while computer use itself may soon be solved, its current lethargy is telling us the following: that unless you can build a very replayable training target for a domain, the models will struggle to make much progress. And the reason this is true, of course, is that the models are incredibly sample-inefficient during training.

This is the point I was making in my last video essay. So for computer use, we might be able to make up for the sample-efficiency deficit by building these farmable deterministic simulators. But for so many other different kinds of skills that we need AIs to have, we simply can't do this. How do we train an AI to get really good at building a business from scratch? How about winning court cases, or having a profitable day of trading in the markets, or helping a candidate win an election?

The rollout here requires interacting with the real world, and you can't recreate it from just within a datacenter. The outer-loop verification here may take months or even years of real-world actions to elicit, and you can't re-observe it by perturbing the model's actions slightly in thousands of parallel rollouts to isolate exactly what the model did that actually worked. Now, dealing with such reset-free, non-stationary environments is a known open problem in RL. I'm not pointing out anything new.

But I really do want to emphasize
that because of the idiosyncratic and sparse nature of data in most
domains in the world, you need sample efficiency in order to get proficient. If AIs are to develop all the skills
that humans have, and even skills that humans don't have, then they need
to be able to learn from information revealed in unstructured, unverifiable,
and ambiguous ways from scarce amounts of real-world interaction. Because in many domains, the
relevant training information simply doesn't exist in any other way.

What is the RL environment to make
an AI that is as good at politics as Lyndon Johnson, or as good at building
a space-launch business as Elon Musk? The labs are betting that
RLVR will generalize. That is, that if you train on
enough containerized, reproducible environments, you will develop a
very general agent that can make and execute plans and learn rapidly from
new information, and even pick up new skills, all within a single session. If you drop this endlessly RLVR'd
AI into Texas politics in 1948, it could give you better advice than
LBJ about winning the Senate seat.

And if you gave it a hundred million
dollars in 2002 and let it cook, it would build SpaceX for you. Now, whether RLVR can generalize
this well is an empirical question. If the labs went from spending billions of
dollars on RL environments to a trillion dollars, would you get the kind of
thing that is a fully human-like general intelligence within the context window? Dario gave a telling quote during
our podcast together, which I think hints that RLVR generalization
is not infinitely strong.

When he was explaining why model
performance tends to degrade at long context, he said: "There's two things. There's the context length you
train at, and there's a context length that you serve at. If you train at a small context length
and then try to serve at a long context length, maybe you get these degradations." Now, maybe I'm reading too
much
into this, but it seems like he's saying that short-horizon RL training
doesn't necessarily generalize to long-horizon RL performance.

And if you can't generalize from short
horizon to long horizon, then how are agents supposed to generalize from getting
trained at a bunch of white-collar tasks to, say, having the ability to be dropped
in the real world and build a business from scratch as well as Sam Walton? And even if, after enough in-context
experience, the AIs could become like Henry Ford or Albert Einstein,
all that would be ephemeral and wasted if you couldn't get those

learnings back into the weights.

Around 30 to 50 percent of a lab's compute goes to inference, and that compute is currently not playing any productive role in helping improve the model. This seems like a huge waste. And it's even worse than it sounds, because it is only in deployment that the most valuable bits of information which your model could learn from are actually revealed. What's actually happening in the organizations where I'm being used? What are they using me for? And what kinds of mistakes do I tend to make in the real world?

We've got some genius grad student who's never been allowed to take a real internship, and we keep giving it more and more classroom case studies in the form of RL training on environments. It's so bizarre that we have AIs that are broadly deployed through the economy already, and are participating in so many different kinds of tasks, and are privy to so much domain- and organization-specific tacit knowledge, and they're not able to make use of it. But this kind of continual learning requires going back to the weights.

AIs can't just keep building up a bigger and bigger KV cache as they learn from more and more users. That's just not scalable, and that's also not how humans do it. There's no clean separation in our brain between parameters and activations, and it's not like some part of your skull keeps expanding as you learn more things throughout your lifetime. When we learn stuff, there's clearly some kind of compression, and this aids our generalization and grokking. There are, in fact, some humans who have this autistic-savant-type ability to recall random tables of numbers or nonsense syllables years later — basically the kind of fidelity of information that models have in context.

And such sheer volume cripples these humans' ability to understand abstractions and metaphors. Human continual learning is less about having all your observations at the tip of your tongue and more about chiseling the right intuitions and big-picture knowledge back into the weights. But the moment you move into the weights, you have to give up on in-context learning's sample efficiency. Because gradient updates are super sample-inefficient, all of the successfully shipped online-learning models have had to learn the exact same thing across millions of users.

For example, the Cursor Tab model online-learns by predicting the same exact objective for over 400 million requests a day. The objective here is which edits actually got accepted by the user. At least so far, we haven't seen models online-learn different kinds of things for different users, because while a single

session may generate more than enough data for a human to learn from, it's not enough to train a more capable AI. Current online learning can work for a very limited number of use cases.

But the whole point of continual learning is that the world is very complicated, and each job and company and problem is different, and you need your intelligence to be able to learn the specific information related to a particular deployment, which simply can't be stuffed into some shared training run. These are all the things we're talking about when we talk about on-the-job learning: things like how everything in your organization works and fits together, how to cooperate with all the infrastructure and the other people around you to make progress on some larger project, what the common failure modes are, and many other things like this.

As the podcast has grown, I've had to deal with more and more operational overhead. Take paying bills. In the past, contractors would just email me their invoices. Every few weeks, I'd dig through my inbox, create a folder with all the bills, and manually pay each one. At this point, though, I just give everybody an email address that goes straight to Mercury, which is my banking platform. Whenever anybody sends an invoice to that address, Mercury automatically downloads it, scans it, and extracts all the relevant information — things like the contractor name, address, payment amount, invoice number, and due date — and then uses all of this to create a draft payment.

Mercury then stores a list of these drafts for me to review. I just go through the list and double-check that they've been built correctly. I don't have to track anything or enter any information myself. Mercury does all the fundamental things for your business extremely well, and it puts them all in one place. If you want to learn more, go to mercury.com. Mercury is a fintech company, not an FDIC-insured bank. Banking services provided through Choice Financial Group and Column N.A., Members FDIC.

In this way, sample efficiency and continual learning are actually deeply connected problems. Relatively little data is available to the model on the job. Now, to learn from this data requires sample efficiency, and models can do that in context, but using the fast weights that are built on the fly by attention, which allow for this sample efficiency, scales very poorly in terms of memory. So we need architectural innovations that allow for some kind of intermediate representation. I talked before about how we already

have many different working ideas for this kind of thing, from sparse attention to KV cache compaction.

And every week, somebody releases a new paper suggesting some kind of other architectural optimization. It doesn't seem to me that architecture is fundamentally what is bottlenecking continual learning. So perhaps the bottleneck is the loss function. How do we update the weights, AKA how do we improve the model itself, based on information that was learned from one particular session? Even here, naively, it seems like there are many ideas that ought to work. A lot of people are talking about this technique called on-policy self-distillation recently.

If you want to learn more about it, I recorded a little impromptu blackboard lecture on my iPhone with Sasha Rush a couple weeks ago, and it's in the link in the description. But to summarize the explanation, the idea is that we encourage the base model to make the same predictions when trying to solve some real-world problem as the model with all the context accumulated after a long session would have made. The whole point of this procedure is to distill what the model learned in a session back into the weights themselves.

This is better than RLVR for two reasons. One, OPSD doesn't require us to have some outer-loop verifiable reward. We just need a model that can learn the right things within the context window. And as long as we have that, we can train the base model to match our veteran teacher model, which has built up all this experience during the session. And two, OPSD provides a much denser supervision signal than naive RL. Instead of projecting a single reward through the whole trajectory, you can train on the per-token probability discrepancy between the teacher and student.

For continual learning, OPSD is also superior to supervised fine-tuning. The most naive version of SFT for this application that you can imagine is just to train the base model to predict all the tokens that are observed during the session. But this makes no sense if you think about it as a learning target. The way you get better at your job is not by recalling the transcript of every single thing that happened every day with perfect fidelity.

Rather, it's by consolidating the handful of insights and pieces of knowledge that are actually relevant to you getting better at your job. RL training doesn't suffer from this failure mode. RL is great at concentrating the

update to only what is relevant to getting the outcome right. That's why the updates from RL are incredibly sparse. And this is a very important property
for continual learning, because as you're learning on the job, you don't
want to overwrite and forget all the other things that the base model knows.

I wrote a post a few months
earlier arguing that RL learns much less information per
sample than supervised learning. But this may be a good thing
rather than a bad thing. You only change the model as much
as is absolutely necessary to achieve the outcome, and no more. OPSD preserves this property of RL,
where instead of slingshotting towards the teacher distribution as supervised
learning would have you do, you only extract the knowledge that is necessary
to achieve the same results as the teacher on actual real-world tasks.

OPSD is one way to attack the
sample-efficiency problem. You take this scarce real-world
experience, and you squeeze all the signal into a tiny, well-targeted update. But there's also another
much more speculative idea. Let's call it dreaming. If the AI can build a good simulation
of reality against which to rehearse new skills, or try alternative
strategies and reinforce what actually works, then AIs could experience
orders of magnitude more simulated samples in the same wall-clock time. Let's go back into history a bit.

A couple years after DeepMind released
AlphaZero, a group of researchers trained a model called EfficientZero,
and the whole point of this model is to be very efficient with data. So if this model and a human both got
two hours to play against a simulator of an Atari game that they hadn't
seen before, this model would actually probably beat the novice human. Does this mean that the model was
more sample-efficient than the humans? Well, that was the goal of the
training, but it depends on how you measure sample efficiency.

Because for each step in the real
game, EfficientZero is playing dozens of simulated games in its head. In a similar way, future LLMs
might be able to consume far less real-world data while practicing
endlessly against environments that they build for themselves. The big difference, of course, is
that it will be much harder to build a simulation of the whole world than
it is to emulate the game of Go. That's why I said this is a
much more speculative idea.

If it works, it would become a fourth
axis of scaling alongside pretraining, RL, and inference-time compute. You could call it test-time
training or dreaming. The model spends compute writing up

RL environments and then training against them, and it's rehearsing all
the skills that will actually be used in production for a specific user. So instead of hitting /compact in Codex
or Cursor or Claude, which kindles a small amount of compute to write up a summary,
and which gives you the simulacrum of continual learning, you hit /dream.

And this incinerates huge amounts of
compute to build and train against a video-game version of what the model
is witnessing in the real world. So what might continual learning
look like by 2027 or 2028? And how do we get there? Here's one scenario. All of this RLVR training is producing
an agent that can get its bearings when it's thrown at an unfamiliar problem,
and it can try different strategies, and it can iterate when it hits a roadblock.

This is the crucial thing that
RLVR has given you: an AI that is at least competent enough to
start getting some real-world experience, if it could learn from it. And once you have that, you send
it out into the world to do real work, even on projects that are
off the training distribution. Now let's say at this point, the
effective context lengths have expanded such that AIs can jam and co-work with
you for a full week of wall-clock time.

At the end of a week, you give
it a thumbs up or a thumbs down, you give it a work review. And if you give it a thumbs up, the
base model distills everything that the AI learned during the session, and
it may use OPSD, it may use dreaming, it may use some other technique that
we aren't even aware of, or it'll use a combination of all of the above. And AI can get better at domains that
are adjacent to what it was explicitly trained for beforehand with RLVR.

And in the next round it gets better
at the thing adjacent to what it was previously online learned. In this way, the gamut of AI skills
and knowledge and capabilities can expand far beyond the verifiable
domains that the model was originally trained against before it was deployed. Just as pretraining created a base intelligence that was smart enough to become a competent agent with enough
RLVR on top, so RLVR has created an agent that is competent enough to actually
be broadly deployed in the world, and from this broad deployment to learn on
the job once the training recipe for continual learning actually arrives.

By this point, the main way that
AIs get better is not from the training they have received before
they are released to the public. Rather, it's from all this experience that
they'll be accumulating from being broadly deployed in the economy and engaging

in so many different kinds of tasks. Every time that you interact with an AI, it'll be smarter, not only because it's been learning from your previous sessions, but also because it's been learning from all its interactions with all the other users in the world.

And that's very scary and exciting and different from the way that AI improves right now. This was a narration of a blog post that I also released on my website at dwarkesh.com. Go there if you want to read all the footnotes, or if you want to sign up so you can find out when I release the next blog post. Otherwise, I'll see you on the next episode.