(no title)

75 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=DIFAI87WS4E
https://www.youtube.com/watch?v=dIFAi87Ws4E

SUMMARY

The lecture discusses advancements in Reinforcement Learning from Verifiable Rewards (RLVR), highlighting the significance of recent developments and algorithms like GRPO and DPO. It contrasts traditional RL methods with newer approaches, emphasizing the importance of reward models and the challenges of implementation, particularly in language models.

- *Recent breakthroughs in RLVR are timely and relevant, particularly in solving complex mathematical problems.*
- *Traditional RLHF methods face issues like overfitting and annotation bottlenecks, necessitating new approaches.*
- *GRPO simplifies the RL process by removing the value function, using z-score normalization for advantage calculation.*
- *Implementation of RL algorithms can be complex and sensitive, with many nuances affecting performance.*
- *The lecture covers various models, including DeepSeek R1 and Kimmy K1.5, which utilize distinct approaches to RLVR and data handling.*
- *Effective data curation and filtering based on problem difficulty are crucial for successful RL training.*
- *The importance of robust reward models is emphasized, as they directly impact the effectiveness of RL training.*
- *The Quen 3 model showcases a structured approach to combining different training phases and expert models for improved performance.*

Okay, maybe we shall get started. Um, this is the second uh of the post- training lectures. We're going to talk about the exciting developments in RLVR or reinforcement learning from verifiable rewards. Um, and I think this is uh in some ways very timely. Um I didn't get to update the slide but there was just a kind of announcement you know by open AI folks that they had solved uh one of the kind of major herdish problems. Um so that's a you know open math problem using one of their like thinking models. How does that thing work? That's exactly this you know thinking model stuff or RLVR stuff. And so thus far you know where we're at is we are kind of at the left side here right after we finished last lecture.

We've got kind of instruction tuning. We've got RLHF. We got some good stuff. That's chat GPG. And what remains is kind of the modern developments in thinking models. Uh these abil the model ability to do very long coot and solve these hard verifiable problems like mathematics uh and coding in some cases as well. Um and remember that I ended last lecture with kind of this uh little bit of a downer note. the RLHF wasn't really going to get us where we wanted to go because of this problem called overoptimization. Right? So, we're going to go and collect a bunch of preference data. Preference data is very useful. Uh we will build some reward model and then we will RL against that reward model.

But this thing is fundamentally very annotation bottlenecked, right? Because we have a reward model and you can't keep putting compute into the same reward model. Eventually, you're going to overfit your reward model. Um, and no matter how good of a job you do at regularizing, you know, eventually you're going to run into this this problem with overfitting. Um, and so really this is very depressing and it's not really the the potential of reinforcement learning. If you're a big reinforcement learning believer, you know, you're going to look at all sorts of things like Alph Go and so on where you say, well, in these domains, you know, RL really did an amazing job. um what is the difference between what we're doing in RHF and what you know has been achieved in these more RL native or RL natural domains um and really the the difference I would say or one major difference between something like Alph Go and something like RHF is that in Alph Go we are optimizing exactly what we want right like you get the win- loss conditions of the game of go you don't have any sort of like sloppiness to that definition and so you can just put in as much compute as you want and as long as the objective improves, you're doing well, right? Um, in some sense, these are search problems, whereas the top one is much more of a learning problem. Not quite a a precise distinction, but that's one way of thinking about it. So, now there might be other domains like formal mathematics or even natural language mathematics that have this flavor of being more verifiable and therefore much more amendable to reinforcement learning. So, that's going to be kind of the motivation for why we're going to study some of these. and the algorithms aren't going to be that different fundamentally, but where we will end up will actually be surprisingly different. Um, and so the lecture is going to be in two parts today. This kind of mirrors a lot of how we've structured the course in other parts. Um, so in the first part of the lecture, I'm going to talk about the core algorithms. This is like basic knowledge for you all. You will use some of this for your assignments. Um, and so I'm going to talk about things like what is gpo, like how does PO work? And then after I've done that, I'm going to dig through a bunch of uh open source releases and show you how what I've taught you in the first part sort of reflects in the technical reports of many of these uh major open- source model releases. Okay, so we're going to dive in to the methods and like sort of basic section. Um and the first thing that I'm going to start with is PO, right? Like we can't really discuss RL uh for language models without discussing PO. Um and even though we've done it once, PO is confusing enough that I think you will benefit from doing it twice, right? Um the most important thing to identity let's say to remember in RL for language modeling is policy gradients and in in particular kind of this like reinforce gradient trick, right? Um what we are always going to be doing is gradient descent on our rewards and we are going to do so by taking essentially weighted uh SFT updates where the weights might be positive or negative. We will define what R is but this equation is important because it will reappear throughout this lecture. This is in some sense the core from which everything else is derived.

Now you know building upon that remember that the intuition from last lecture was we want to take more than one step at a time. A policy gradient requires us to sample from our policy every time we want to take a gradient step. Can we reuse our rollouts? Well, you can do TRPO, you can do PO, and so on and so forth, right? Um, I'll talk about PO in a minute, and I've already talked about TRPO. I won't really get too much into this. If you've taken an RL course, PO is of course, you know, well understood to you, like you understand this algorithm. Um, but I don't assume that all of you have taken an RL class and or understand, you know, RL. Um and so PO in case you're not familiar, you know, is a kind of real workhorse um of RL.

It's been used in a variety of very difficult RL settings. Um OpenAI has done quite a few of these in their early early days of being, you know, very RL pill. Um they had, you know, this was the I think OpenAI gym. Um

where they were talking about the PO algorithm and they're like, hey, we can get these these uh people to walk around using reinforcement learning. And a much more dramatic demonstration of this was their open AI bot which they trained using PO.

Um, and so this can excel with kind of deep RL and like highdimensional action uh in state spaces where it's much more complicated to do RL than it is usually. Um, oh yeah, and there's there's you know the the open AI man running around. All right. Um, so at a conceptual level, PO is actually very simple, right? Remember that I um told you, you know, a important part of all of this is getting rid of PO. That's been a motivating factor for a lot of people. But if you look at the pseudo code of PO, you know, this is from uh, you know, the OpenAI is like spinning up on RL documentation.

You look at this and you say this is not that bad. This is actually like pretty easy. I could implement this in one go. You sample some trajectories. I compute something called an an advantage using any method of advantage estimation. I clip the advantage in this like kind of slightly strange but okay way and then I update the policy under this clipped advantage and that's fine right and then I can fit something called a value function as well okay fine this is all relatively easy looking in practice um and so I think because of this you know people are like ah it's okay po's totally fine but then you see blog posts that look like this and this should strike fear into your heart right because if you see a blog post that says the 37 implementation details of PP EPO, you know that this is an algorithm that is very sensitive to your implementation decisions, right? Um, and you know, there's all sorts of things that you can do and all sorts of different libraries and implementations and they give totally different numbers based on which ones you use. Um, and it turns out that, you know, many people have implemented this wrong. And so there's papers saying like actually the baselines that some people use in PPOs aren't even baselines at all. They fundamentally change the optimization problem. Um and so you know basically because of this I think I would you know not be able to get through the rest of the lecture without telling you well we actually maybe need to start looking at some implementations of PO and the implementations of PO let me tell you for language models are not particularly pleasant. Um you know this is a I mean a very good representative slide. If anything this is a very good description of what PO looks like. You know you've got uh advantage estimation.

this big block in the middle. That's this advantage thing. You've got an experience buffer because you're going to keep some of the old stuff. It's going to be training this value model and the value models you be being used in the advantage estimation calculation. Notice how the the green box appears twice. Um, and importantly, you know, some parts of my objective, the KL term actually operates token by token. So, it's not actually just a bandit problem.

It's like a whole multi-step RL problem. And this is all very difficult and complicated. Um and so this has led to a variety of you know really tricky implementation things that you have to do. And if you look at a variety of implementations you know you find often that there are things that people do for PO that are a little bit strange uh and tricky at times. Um let me see if I can find um you know so uh one of my students implemented PO for a project a while back doing RLHF right so PO if you remember was used in RHF and so you know we had this nice robust reference implementation of PO um and it took a long time to get working so I like looked into the

details like what did we actually do to get this thing to work the outer loop is fine the outer loop is something like you go through your rollouts and you're going to take some PO steps you're going to compute some loss and clip some gradients and take some steps right totally reasonable outer loop Um but then you start kind of looking into it. Um for now the the actual like inner compute loss part follows almost exactly the PO update. So this also looks mostly good. We're computing things like you know the advantages and you know we're computing the clipping ratios and then we're updating the model based on that. So this is all kind of good but then you kind of get to some of the messier parts. Um actually I'll skip the rollouts. um where you have things like well maybe we need KL penalties to sort of you know keep the original model close to the reference but actually this only works if you clip the KL off at zero which of course if you know anything about KL divergences that totally ruins the point of a KL divergence right you have both positive and negative values being summed um if you remove this blows up immediately right all sorts of things like this can happen this is not to say that this is the way you should implement PO if you were doing it I think the reason why I bring this up is to say implement mentations for RL algorithms can be somewhat tricky and somewhat sensitive because of all the moving pieces in PO and because of the high variance in the gradient estimates like these are all very very tricky things um that are happening and you know another thing that you often see this one is not just our implementation it's very common um in the original PO paper you have this thing called a generalized advantage estimator that's basically looking at this like um you know gamma discounted reward where you have this value function that's estimating the reward at every token generation step. Um, but it turns out that, you know, people often just use gamma equals lambda equals 1, which is just a degenerate setting that turns this back into a bandit problem.

So, you've kind of thrown away a lot of the structure that you get from uh PO. And so, like if you run this algorithm, you know, after a bunch of engineering, you will get what you expect. Then you know when you run your your RL algorithms for the assignment you should expect to see similar things like the rewards go up the reward model rewards go up if you're doing RHF and then the negative KL regularizers kind of go down. This is a reasonable thing to expect to see. But anyway, you know the the point of all this is to say implementation of PO can be quite painful depending on the environment. It can require hacks to stabilize. All of this is kind of painful. Not to say that it's not possible. Lots of people have trained very good models on PO. Um, many of the labs have sort of very turnkey solutions for getting POS to to work at scale. So, this is not impossible. But for people like researchers who are, you know, implementing this from scratch, PO can just be really finicky and complicated. Another reason why PO is disfavored in many cases is that requires a value model to estimate kind of the value at each token as you go. Um and how big is the value model? Well, it's you know as big as the original model. So this consumes some memory that you would rather be using for other stuff like you know models or inference uh servers. So now you might say well in last lecture you told us about this DPO thing and DPO seems pretty good. Um why don't we just use DPO for everything?

Well the problem is DPO is a very specific solution to a very specific problem. DPO is good for pair-wise feedback in the form of Bradley Perry comparisons. that is very specific, right? Um, and if I want to solve math problems, you know, my math problems don't come in the form of inherently pair-wise comparisons. There are other DPO variants that are designed to sort of break that pair wise structure, but really you're just kind of using the wrong hammer for the job, right? You do not want to use DPO necessarily for what you would normally use PO for. PO is the more general hammer that you can hit kind of everything with. um DPO is generally offline

although I think this this uh distinction is very overstated because it can be made online by just iterating DPO repeatedly. So I don't think this is necessarily a big difference. Okay, so that's the context for you know uh all of this. Um and then so you know that similarly to DPO there's just an enormous desire by the research community to not have to use PO right um hopefully the fact that you know DPO and GRPO have gotten adoption tells you how painful it was to get this to work in many cases um and so GRPO much like DPO is the alternative it is the simpler way to do RL for verifiable task has taken over for the most part um RLVR in the open-source you know open knowledge let's call it uh community um and it is very very similar to PO and spirit. It just strips out a couple of the most complicated parts and by doing so um allows us to have a much simpler algorithm. So GRPO is by um uh a Deepseek paper um I think it was their like deepseek math paper um where you start with PO right it conceptually it says PO is a good idea but we want to change just a few things and what does it change it changes it's arguably the most complicated and annoying part of PO which is the value function right so the value function is the thing that you subtract as a baseline to reduce variance of your gradient updates um the value function is a whole neural network it destabilizes training we don't want the value network. So what do we do?

Well, we get rid of the value network, but we still need an advantage. I don't want to do reinforce vanilla reinforce because the variance of that is going to be really, really high. So instead, what we're going to do is I'm going to compute advantage as a zcore within a group. And I'll explain what that means in a moment. But basically, the idea is kind of the following. Normally, you kind of look at a reward that you get.

And if you had a value function, you would compare it to your your predicted value. You would say my neural network says I should have gotten a score of five. I got a score of six. So this is a good roll out. Right? That's usually what you do. Instead, what you would do is you would get let's say your your roll out with a score of five. You would now sample 10 other rollouts and you would say how good was I compared to my 10 other rollouts, right? If I'm doing better than my mean, then I have a high advantage, right? In some ways, a very natural way of computing advantage uh in context where you can multiple sample from the same prompt or something like that. Um so GRPO is exactly this idea.

If you look at the the you know initial definition of GRPO that appeared in the Deepseek paper you know you see this is the uh uh the objective that's being optimized. They're going to do exactly the PO update where they have this like min clipped advantage plus a KL term to keep you yourself close to the reference. Um and then KL is computed in a particular way that detail is not particularly important. And importantly, this advantage is computed as a zcore over samples of a group of outputs O of one through O of G. Right? So that's uh or within each group you have oh sorry this advantage of I this is the place where you compute the advantage. What you do is you take the reward over your rollouts within each group you subtract the mean and you divide by the standard deviation. Right? So you get a zcore for each group. Um so uh in the online case and this is an important distinction because in many cases you may actually end up running this online rather than offline. Um in the online case the clipping right just kind of disappears because the ratio between pi theta old and pi theta is one. This clipping operator never does anything. Um and you just get min a of i a of i. So this is just advantage minus a kl penalty.

Right? So I'm going to do rl on my advantage which is defined by this. So in the online rollout case, it is a very very simple object uh that kind of uh makes sense as a RL uh target. So I can pause here for a moment in case any of the conceptual bits of gRP are not clear. I mean in some sense this is a foundation of all the exciting uh math stuff or you know you can replicate all the exciting math stuff using gRPO. So it's worth understanding in a little bit of detail before we sort of move on to to some of the nitty-gritty details.

Okay, good. Simple enough that everyone understands in one go. I will take it. Um, and gpo is very very simple. Um, you know, because of the lack of value function, you can basically write down kind of all of gRPO in a very simple block of code where you just say I'm going to roll out, you know, k times. I'm going to compare my, you know, observed roll out to those K sort of references, Zcore it, and then just take a reinforced gradient against those weights, right? Um, so you will write this in your assignment. Um, and it's going to involve a very simple set of steps. You know, compute a reward for each of your K rollouts. You know, normalize your rollouts, compute the KL term, um, and then just take gradient updates on the combination of your KL term and these rollouts. um you'll have to do a little bit of thinking because in order to do this using um autodiff you will have to do a stop grad somewhere but it's actually not that complicated and you can sort of see that you can do a onepage implementation uh of this uh basic idea right um and this is a reference implementation of gRPO that I took um from you know some some folks at McGill which has a really nice little toy version of this whole thing um and the computation of the group indices and like the standard devi ation calculation and all of this just kind of fits in one half of a slide. Um, the only thing that they do differently than like the literal paper write up of GRPO here is they add a little tiny 1g4 to the standard deviation calculation to prevent it from blowing up when you only have a single sample or if your samples happen to have the exact same rewards, right? Which which does happen if you're in a domain where you can get exactly numerically zero rewards, like you failed to solve a math problem. Um but hopefully this um you know emphasizes to you kind of how straightforward gpo is and why almost all open- source work has been built on this algorithm. It's easy to implement, easy to understand. Um and if you as we'll see in the results, it provides fairly compelling results in terms of uh emulating a lot of what the the closed source labs have done using potentially bo.

Okay. So how well does it work? um in the original uh Deepseek math paper, which if you're into sort of math AI stuff, I would encourage you to read. It's a it's a nice and really interesting set of results. Um you know, they show that GRPO uh the yellow and the blue lines does much better than RFP, like rejection fine-tuning. You'll implement this as a baseline as well. That's just taking the correct answers that your model generates and training on them and throwing away everything else, right?

um and they seem to show process supervision which is grading not just the final answer but grading the intermediates give you some gains. I'll talk more about that later. That is an important thing to uh discuss and cover because that's one of the big sort of design decisions uh for these kinds of RL problems. But for now, you know, you can just internalize the fact that GRPO works. The blue and yellow lines are above the others.

Okay. Um so now let's think a little bit more about the GRPO objective like what is happening in this objective are we actually taking policy gradients there are important conceptual questions uh that we should really think through um and so if we think about the difference between GRPO and PO right remember that PO had this

value function thing which feeds into the advantage and in GRPO we stripped that out and we replace that with the zcore guy right this a of i ratio of r minus mean divided by standard deviation So, is this a good advantage? Right? I think for those of you that have been in an RL class, you kind of already know the answer to this question. Like, is this a good advantage function? Um, you know, why what makes a good advantage or what makes a valid advantage function?

Well, you know, if you look at Senardo, you know, the the big kind of classic book for reinforcement learning, you know, they will say, well, there's this algorithm called reinforce with baseline. And the things you're allowed to do is that you know if you're doing policy gradients right which is this this line over here this very first equation right I want to take my gradient step on the direction of my rewards and the thing you can do is you can take your rewards and I can subtract any what they call state dependent baseline in this case our state if we're in the bandit world is just the prompt right so you can have a prompt dependent baseline that you can subtract and anything that does just this is a valid form of a policy gradient. So you're still going to descend in the same direction as long as you do this. And depending on your choice of B, you will either have lower or higher variance um in in doing this gradient descent process. Now you should probably notice that what we're doing is not this, right? We are not just subtracting a constant value. We're also dividing by the standard deviation. Um and so that is kind of a problem it turns out.

Well, it's in some ways a problem, in other ways it's not. Right? If you really want a conceptually clear algorithm that really does what's like written on the tin, like it actually descends the reward, GRPO does not do that. Right? Because as I've highlighted in red here, right, GRPO divides by the standard deviation, which breaks sort of this kind of, you know, baseline contract. We're not just dividing or sorry, subtracting a baseline, we're also normalizing by the standard deviation uh of my rewards. Um you could of course oh there's another thing uh that GRPO does which I didn't mention earlier um as an implementation detail that is kind of tricky which is GRPO does this kind of almost per token right so they'll divide by the total length of the sequence um as a normalization factor but if you sort of try to derive gpo from first principles following the policy gradient and baseline theorems you'll end up with something different you won't have a length normalizer and you won't have the standard deviation normalization, right? Um and some folks, you know, soon after GRPO came out noticed this. Um and they wrote a paper basically saying, well, if you don't, you know, do these like two things that are different, then you get very different potentially very positive behavior. I will talk about that, you know, um also when we get to the uh not open AI, sorry, uh Deepseek R1 uh paper.

Okay, but the main thing here is to say, you know, GRPO is not the first principles derivation of this idea. It is actually doing something slightly different with both pros and cons. So what are these two terms doing? We now know that GRPO is not actually directly descending on the reward objective. Um, we know that it has these two correction factors. What do they do? Well, the length normalization is easier to see, right? So what are we doing? we are dividing you know longer sequences with a bigger number and so what this will do is that it's going to encourage the model to generate long outputs right because or sorry when you're wrong it will encourage you to generate wrong outputs um because let's say I'm I know that I'm going to get a math proof wrong I'm going to incur my negative reward of let's say negative one I'm just going to generate an infinitely long string if I do that I'll get to divide by infinity um and I'll get to totally get rid of my negative penalty right That's a that's an extreme case, but you kind of understand that if you divide by the output, you encourage the

model to blab on once it realizes that it cannot actually solve the problem. Right? So this is a length problem that you'll end up getting. Um if you fix that, um you'll find that often you know what people have observed in GRPO of like you know coot like chain of thought thinking length growing and growing and growing that actually turns out to just cap off at a constant rather than just continually and forever grow. So this seems like a potentially good thing. Um and especially on the incorrect cases, you really don't want to be, you know, generating longer and longer and longer um outputs. Um so the standard deviation normalization, this one's a little bit more complicated, but if you think about it, what I'm doing is I'm dividing by the standard deviation. And so this can be thought of as emphasizing problems where the standard deviation is small, right? And when is my standard deviation small for a binary reward problem? Well, that's when the problems are too easy or too hard, right? If my problem is really easy and I get 100% all the time, I'm going to divide by one over or you know the the number the I'm going to divide by basically zero, right? Because there's zero variation if I always get a question correct. Similarly, if I always get a question wrong, I have zero variation in my rewards. You know, I'm going to upweight that significantly, right? So the standard deviation term is upweing sort of the two sides both easy and hard questions. That seems like clearly a thing that maybe we don't want because we want our models to potentially learn on kind of things that are within its solvability range.

Okay, so that's kind of the algorithmic components um of RLVR and I'm going to go through um several of the different recent model releases and kind of talk through all of the algorithmic components um that I think are quite important um including um some of the stuff that I added this year that's new which is really the agentic stuff that I think has been catching on and is increasingly important uh for using and deploying uh these models. But before I do that, I can pause here and if anyone has algorithmic questions, I'll I'll take them before I move on.

Okay, good. All right, so we're going to start with DeepSeek R1 because I think this is, you know, you know, a bit of a social phenomenon. I think everyone should know, you know, a bunch about Deepseek R1. I think it's a lovely paper. Um, kicked off the wave of open-source uh RLVR models. Um, and what's remarkable about it, I think it was really the first uh thing to have matched OpenAI 01's sort of behavior, right? So, the key things about OpenAI1 was you have very long chains of thought. It's, you know, clearly RL um and you have really good performance on hard math problems, right? Um, and they also provided a RL recipe that anyone could do. And I think from the perspective of R&D, this is very important, right? If your solution was like you have this horrible PO thing that no one but deep sea can run um then that is much less of an impact um at least for us researchers than this like kind of GRPO thing which anyone can really uh play with even you in sort of your assignments and finally they have some really interesting sort of distillation insights that I think you know have held up um you know even even now okay so the starting point of deepseek R1 and I think you can kind of see how a lot of deepseek's like expertise build on each So, you know, they had this deepseek math paper where they were doing gpo. They had already understood a lot of the nuances of rlinging on math problems. Um, and so because of this, you know, they started off with a lot of the components of deepseek math. Um, but one of the really important differences, um, and I'll sort of highlight an expert just because I think it's particularly important is they abandon process supervision, which was the thing that worked really well in deepseek math and they go only with outcome supervision. And just to be clear, outcome supervision is when you have reward only for whether the final answer is correct or incorrect. And process supervision is when you have a grading rubric, let's say, to check this, uh, validity of intermediate steps in a proof. Um, a lot of people thought process

supervision was important.

Turned out it wasn't critical for a lot of things. So um R1 was interesting um in that it has a lot of different components but one of the very interesting components um is R10 where they don't really do that much post-training. So they have a base model. Of course the base model as we all know now is is also mid-trained. And so this guy can can do some amount of instruction following. And then on top of this base model they do you know basically RLVR where the rewards for their GRPO algorithm is accuracy rewards for whether they correctly solve a collection of math problems. Um and format rewards. And this one's important because the format rewards basically allow them to strip out the chain of thought later. They want the model to properly use thinking tags to enclose their coot. Um, but this is a very simple recipe. GRPO is very simple.

Accuracy rewards are very simple. Um, in fact, you will implement and replicate essentially this set of results in some ways. Um, and you do just this very simple recipe and where do you end up? Well, you end up something that is only a little bit worse than OpenAI 01. Right? So this is really really really clean and I like this result because it has none of the messes of like a real production post-training pipeline thrown in. You don't you know really question whether like oh was it because of the RHF helping or this or that. It's a very simple base model plus GRPO. Your math abilities are quite good. Um and I think R1 also kind of took off partially because of the interesting phenomena they claim to have observed in the paper. So one of the things they claim to have observed is you know as we go longer and longer and longer during training the model is able to think for longer and longer and longer the coots get longer and longer and longer. Um and they also highlighted this thing that I I believe went viral on social media which is like ah the aha moment where the model is like you know if you train it for enough it coot is able to do things like oh there's an aha moment that I can flag here. Um and you know they thought this was really interesting. Um it is unclear to me whether either of these phenomena are really particularly uh impressive in many ways. like we now know that longer coots is arguably a a natural side effect of the length normalization of the gpo algorithm and the aha moment um you know others have showed um is actually the appears in even the base model and so clearly it can't just be a result of the RL algorithm right like you know lots of people say aha I can use this as part of solving the math problem the model has learned it during pre-training you know it's not surprising at all that somehow during RL just happens to get extracted because it's you know emitting a lot of math tokens right um so I think the phenomena that was highlighted is not particularly salient or or exciting but I do think R1 was a really important milestone in that it highlighted just how simple RLVR could be right this is very in some sense clean as an algorithm and as a way of solving these really difficult problems um and so R10 was this very nice, clean, controlled setting. And then uh R1 was their attempt to kind of productionize this system. And I think one thing that's nice about R1 and and I'll show you the Quinn uh equivalent later is you can kind of see how you kind of stack all the pieces together, right? I think in we've studied all the pieces in isolation. Like we know where pre-training is, we know that there's RHF, we know SFT, but how do we compose all the pieces to actually get a system at the end? Um well usually the thing that you do is something like well we have our mid-trained model we're going to do a bunch of reasoning training maybe some long context extension somewhere too and then we're going to do RLHF at the end because that's in some sense the most userfacing object right like we want to make sure that the formatting and all this is nice RLHF can very much help with that process um one of the other things that they did was um they added a language consistency reward for the chain of thought for their production model um I think If I remember the report right, this was because um the R10 style training without a consistency reward would like

kind of language switch in coot and they thought this was like very uninterpretable and slightly disturbing and so they wanted to just have it output a single consistent language and so they do this just for sort of interpretability uh reasons. Um, and then they have some of the non-verifiable rewards as well, uh, thrown in to GRPO almost as kind of blending into the RLHF process, but otherwise it's a very very similar thing and the whole pipeline I think is very straightforward. Um, so for SFT, what they kind of do is pretty simple. So in R10, they don't want to do any SFT at all, but in R1, they're willing to do long coot data. And you know, when you're reading these open source tech reports, it's kind of fun to try to read between the lines. When they say something like, "We construct and collect a small amount of long coot data." You know, you might wonder, I wonder if that was distilled from some other models. You know, not that that is a particularly like negative thing for open source models. Um, everyone is nowadays doing this, but I, you know, I just kind of find it funny that it's like very carefully written. You know, collect a small amount of long coot data. Um, and so this is used to fine-tune the model. We know that for a very good base model just SFT on long coot can unlock a lot of 01 style capabilities. Um and you know that's a great starting point for RL. So they they start there and then they add some verification to filter out these coots.

Um and then they you know continue on with with uh processing. Um and you know I think many papers since then um some including some some from my students have shown things that you know if you have the right kind of distillation procedure and you have the right kinds of base model you can basically get a lot of the long coot sort of reasoning juice just from SFT right and I think this is maybe interesting because one of the things that I think is is still an open question is like do you really need RL for some of this and I think maybe one way of thinking about the role of RL in language modeling is that RL RL is a great source of supervision, right? Like if you're solving frontier math problems, you just don't have the supervision to get detailed long coots.

And RL allows you to self-generate that. But once someone has generated these long coots, you could potentially also learn um from imitation. And that seems to be kind of what a lot of these like distillation results uh partially show. Okay. Um so the RL part for R1 is basically the same as R10. Um minus the language consistency loss that I mentioned. It's basically just we run the RL training using gRPO. Um, and you know that leads to essentially the final phase which is what we learned about last lecture. So we do some basic instruction tuning style SFT. Um, and then RLHF. Um, for non-verifiable tasks, they basically use the same thing as uh, Deepseek V3. So there's really no surprises uh, at all here. How well does it work? You know, it's kind of really really good, right? Um, and I think this is the reason why there was all that kind of panic or like big uh sort of noticing of DeepSeek because R1 was genuinely a very good model, you know, like it beat 01 on many of the of the categories. Like it it matched a lot of the test time scaling behavior that people expected. Um, and it came from a very simple recipe. So, you know, it was very easy to understand where the gains uh were coming from.

Um, and the last thing related to some of the distillation discussions that that I was saying is you could then take R1's coots and you could put it into Quent 2.5 and if you did so you could really significantly boost the performance of these models like in some cases sort of matching a lot of these uh specialized thinking models and so in other words you know really if you can get the right kinds of long coots that are legible to these models um often it is possible to get them to to reason for for long periods of time like the base models are already uh surprisingly good and this works even for llama models um as well which I I think is quite uh fascinating.

Okay, so to close out kind of the um uh sort of deepseek saga um I quite like a lot of the deepseek tech reports because they often do both ablations and they also tell you failed things that didn't work. Um, and if you've been following, you know, what they've been doing, Deepseek Math has this all this like process reward model stuff. It's like we got to verify all the steps and then you read R1 and you say like where did the process reward models go? Well, they tell you where the process reward models go, which is we tried to to get process reward models to work. They just didn't do very much for us. It turns out that, you know, um, outcome reward models are great and they're good enough and you can scale the data for those a lot better. Um, I think a big debate for PRMs was where are you going to get these like step-by-step rubrics, right?

And it's very hard to scale that up. Um, I think at this point it's quite clear now that outcome reward models are very very good and that's kind of the bulk of where the action is happening. Um, similarly if you know you were kind of around for the initial 01 release a lot of people speculated about what was going on inside OpenAI's01. They were like oh are they doing PRMs? are they doing tree search like Alph Go, right?

Um, you know, they too tried a lot of MCTS and they sort of describe we couldn't get it to work very well. Um, I just very much want to highlight and sort of like the fact that they're very open about all these explorations and the things that they tried and didn't work. Um, rather than to say like, you know, they didn't try it at all. Cool. All right. Um and you know if I talk about um actually I'll pause here in case anyone has questions although I don't know if yeah explain more because I guess you said for like >> yeah so the question was like for positive cases what does the length normalizer do and of course in those cases the length normalizer like encourage enourages you to shorten the coot. Um, which is good if you want to save on inference cost. It's bad if that degrades your accuracy. Um, but I think the main issue is that for for the long uh for the negative ones like you get like some really long responses. In the positive cases, it can't shrink it that much because there's a lower bound to how small your coot can go, you know, to solve a particular problem.

>> Right? I guess I'm trying to understand that explanation for the uh for for the diagram. >> Yeah, I think Dr. GRPO has a cleaner diagram of this. So like you know in the in the R1 diagram, this is all aggregated. This is like all the you know responses across their evals. It's not just the positive ones. Um but uh if we look at the doctor gpo plot over here. Um so this is like the average output length and this is the incorrect output length, right? And so you you kind of see that and the correct output length is here, right? So um you kind of see somewhat that like this is really being driven by the incorrect ones as as we kind of would expect from the explanation. So the the story at least in the these sets of controlled comparisons is is quite clean and clear.

Yeah. Cool. Um so now I'm also going to talk about an alternative method um Kimmy K1.5 or alternative uh paper and approach. Um and I always feel like when I talk about R1 I als I'm obligated to talk about Kimmy K1.5 because they kind of came out the same time as R1. They also beat 01. And yet everyone is, you know, all about DeepSk. And Kimmy is not quite as often brought up in the conversation, even though their models are extremely good.

And I'm not just sort of talking about them out of pity, per se. Um, I think Kimmy has this nice thing that they do something some sets of things quite differently um than Deepseek. And we can learn quite a bit from like the

fact that both of these work, right? I think this is kind of part of the theme of this class which is by reading all of these tech reports and by looking at broader patterns we can roughly start to understand what are kind of the the valid or easy spaces to to get these algorithms to work. So um they do actually quite a bit more detail in talking about the data set construction and like curriculum generation for the RL process which many have said is quite important. Um, and they also have a different RL algorithm which in some ways like has similar intuitions but is not exactly GRPO. And so this is another nice validation that technically speaking um I don't think you necessarily need something like GRPO to get these guys um to work.

Okay. Um so the first part I want to talk about is the data. Um the data remains very important um for all of these processes. Um and for RL there's an additional wrinkle of kind of curriculum like what kinds of difficulties of problems do you want to throw in right in SF you don't really think about difficulty very much because you just sort of jam the data in there right like for whatever data that you want you just do pre-training losses and the model is forced to learn from these in SF there's a little bit more of a subtlety we talked quite a bit about sort of the hallucination issues and maybe sometimes you don't want to train on some of this data um but for RL there's this additional wrinkle that if your problems are too hard, you get no rewards, right? And if you get no rewards, you have no signal. And if you have no signal, you can't learn. So it's very important to kind of have broad coverage of the right kinds uh of examples. Um and so what uh the Kimmy folks do is first of all, they like try to get a whole big broad range of coverage of data. Unsurprising. Um they exclude multiple choice because that's kind of covered in lots of different domains. They want things that require long deep thought and their argument is maybe multiple choice doesn't have that.

But maybe this last one I think is maybe the most important idea. This appears in a lot of different uh RL papers, right? You basically filter all the examples for things that uh sort of either pass um or or sorry you filter them based on a best of K filter. So here um you take examples um and if they succeed on best of eight like if you sample eight times and the model can do it at least once then that's not a great RL example because it's kind of already at the edge of the model's capabilities right you're not going to teach it anything particularly new and so you only look at examples that you know uh uh basically fail this test um because that allows us to essentially um pick the right kinds of of problems s and so this saves compute. Um it allows you to sort of skip problems entirely. Um you can also sort of filter on both sides to get problems that are neither too hard nor too easy. I think the general consensus in the research community is that doing this kind of like medium range difficulty filtering is very good if really what you want is for RL to like progress at a steady pace. Um, you know, as with DeepSeek, no description about SFT. We can speculate about what that is, but we have no concrete information.

Um, so I think the interesting thing about Kimmy is that they have a DPO inspired argument, but they end up actually in a place that's very similar as uh GRPO. And I think the fact that they end up in similar places is suggestive of which of these components are are potentially useful. uh if you're designing a new URL algorithm for example. So they start at the exact same place as everyone else right so this should now look like very familiar to all of you it's just you know maximize the expected reward under my policy and I've got this KL regularizer on the right that sort of keeps me close to uh my base policy or maybe even the previous iterations policy depending on how you're going to do this uh do this RL. So they're going to follow the same DPO style derivation where they say assume that I can maximize analytically and I solve for the reward model that does that maximization or corresponding to that maximization and that's going to give me this like ratio of policies

um and I'm going to plug that back in to my objective up there um and try to solve for what is happening. Well, what they're kind of now saying is they want to try to match the reward model. Um, and for this equality to be they want this equality to be true, which is true at the minimizer. And therefore, this is a big heristic. You know, if something's equal at the minimizer, why don't we just put a squared loss on that thing and minimize the squared loss? Um, I think optimization people looking at this would be horrified at this. But I think this is a totally reasonable intuition in some ways, right? We know that these two things should be close uh when the model is good. So it is also maybe okay to treat making those two ceilings explicitly close as a surrogate. Um the this derivation as you can see is very very different from gRPO right we're not going through the PO path to derive this uh this uh update but now actually lo and behold you take a gradient of this L you end up with something that looks surprisingly like uh GRPO with an added regularizer term right what you end up with is this thing that is a policy gradient with a baseline RB bar um and that R bar is actually the mean of each sort of condition these X's um and you have this additional regularizer term uh which is essentially the KL term that would appear in GRPO. So in some sense, you know, we've reinvented the group mean normalized baseline um through quite different means.

Um and so, you know, the the Kimmy folks, I think, in some ways, had a a nicer or or better view uh of the length problem. I think the GRPO folks were like, "Isn't it great that the length is growing uncontrollably? I'm sure our model is being smarter." Right? That's a little bit of an uncharitable take, but like when you present this plot as a positive thing, the implied statement is like it's great that our model is thinking for longer. Um, but the Kimmy folks kind of look at this like long coot problem and they say, well, you know, if we're making long coots, potentially that's very wasteful, right?

Like we don't really want long coots because coots cost inference, right? So they not only not not only do they not have a length problem because if you look at this objective they don't normalize by sequence length um but they actually want to go even further and they want to compress the length of the response and this is another sort of important sort of feature that we kind of see generally occurring in language model development like we want the coots to be as short as possible we want to solve hard problems we want don't want to think for very long because when we think for very long we have to kind of you know essentially subsidize the user If you're OpenAI and your users have um uh the $200 pro plan and your models are thinking for like an hour at a time, that's not a very good place to be in, right? Whereas if your models are thinking for 5 minutes at a time, that is a great place to be in. Um and so for all these reasons, um Kimmy wants to go and compress the coots. How do you do that? Well, you know, the simplest RL thing to do is you come up with a sort of heristic uh length reward um and you just tack it on to the uh to the uh RL process. This reward is surprisingly complicated. Um and the thing that they're doing is you're trying to make longer sequences shorter and shorter. Um correct answers also should be short. Um the interesting sort of balance here is if you make the incorrect answers too short, they no longer have a way to recover, right? It's like so let me give you an example. Like imagine uh I'm bad at geometry, right? Like I'm a I'm a AI.

I'm bad at geometry. I get lots of incorrect answers and the penalty makes my geometry coots really short. Right now my geometry coots are zero. I'm really bad at geometry. I will never recover from this. Right? I will never get a positive geometry reward ever again. And I'm stuck. And so what you do is you don't sort of force the incorrect answers to be super short. Um you incentivize them to be just a little bit shorter than sort of the

average, right?

So that the the incorrect stuff doesn't grow unboundedly. Um and so this has like uh in in the end like a surprisingly reasonable set of outcomes. Um and so as I said before, you know, they have a really nice sort of RL set of views on how they do their their data set curation. So not only do they have this data set that they filtered, they also look at the success rates of how often they're able to solve particular problems. And once a model masters a particular problem, it gets taken out of the problem set. So you save some compute doing that. I think basically everyone doing RL does these kinds of like uh success rate filtering to avoid both wasting compute or uh working on problems that are way too hard uh for the model. Um and they you know basically have uh rewards that for code they take ground solutions and they generate some new test cases and for math they actually have a reward model to check for answer equivalence checks and you know this is a really funny thing in some ways because we started out this lecture by saying you know we want to work on like you know formal math or something something truly verifiable right where where you can a compiler can check the correctness of your math and then we've like gone through like most of the lecture And then in the end, where have we ended up?

Well, we ended up with a reward model. A reward model that checks the correctness of math answers. Um, once again, you'll you'll kind of see why in uh the assignment, or maybe if you play with the assignment, you'll see why. Um, you'll quickly find out that answer equivalence checking is a very difficult problem. Um, because in math, you can like write, you know, equivalent things in many ways. And not only that, a language model can give the answer back in many ways. Even if you prompt it to say give the answer back in like latte boxed, maybe sometimes it skips the boxed, maybe add some extra stuff to the box, right? There are lots of ways that the model can just fail to get the right answer by a strict correctness checker even though it has the right idea. And so for these reasons, basically, you know, most RL sort of projects have a very complicated like answer checker, either a reg x or a model or who knows what in order to avoid these kinds of problems. It's a real like rabbit hole um getting like the verified part of RLVR right.

Cool. Um the last thing which you know connects to some of the the systems bits uh that we had talked about earlier and the inference bits is RL uh optimization and RL infra is really quite hard. Um you know training is hard and inference is hard and RL puts the two together. So in some ways no wonder is it really horrible and difficult. Um, even worse, I think one of the things that you don't initially appreciate about, you know, how hard it is to make RL efficient is kind of think of the following scenario.

Now, imagine, you know, you've got your rollouts, but you've got one really hard math problem, like let's say, like, you know, one of them is like the Remon hypothesis, and your model's like really chugging along on the remon hypothesis. It's got this gigantic coot, right? Now, what's happening in the meantime? And if you're doing naive inference, everyone else is waiting on this one roll out to complete in order to sort of move on to the next phase, right? So if you're doing these batch things, long rollouts can really hurt you. Um, and so you know, RL really needs to deal with all of these sort of like really tricky low-level details. Um, so you need to deal with the fact that some coots can get really long. Do you need to truncate them? Do you somehow like set them into a different machine? Who knows, right?

These are all decisions that you can make. Um, if you switch from training to rollouts, like you roll out once and then you train and you roll out and you train, you have to somehow, you know, deal with this. Either some of your machines are pure rollout machines and some of them are in uh uh training machines or you're like switching out frameworks all the time. Both of them are very costly. Um, and then finally, you have this really difficult and horrible trade-off, which is on policy stuff is mathematically and training dynamics wise very nice, right? So GRPO in its simple on policy form behaves very nicely. You will experience this in your assignments and then you will get kind of greedy. You will say ah but my systems utilization is so low, right? I could do so much better if only I could reuse my rollouts. Then I can overlap my inference and computation and do all sorts of clever things. And so you will attempt to reuse these rollouts. and that will lead to off policy problems which then lead to de uh destabilizing your training and all sorts of things that are very difficult. Um so you know these are all sorts of things that end up getting very very tricky. Um and so I think most open-source uh technical reports will have these days a section talking about their RL infra. Um, and their RL infra, you know, is going to have to have, you know, both a training part, that's the blue boxes over here, and an inference part, which is the green part on the right. Um, and you're going to have to move weights from the training part to the inference part. So, you need some sort of story for how to move those around. Um, and you need to coordinate them closely. And in some cases, maybe they even share the same machines because as inference is running, the training one might be idle, right? So all of these are kind of complexities and you have lots of very complicated uh systems things to do. Um so we already know uh that uh Kimmy beats openio1. Um but we also see some really nice things showing that like as RL proceeds you're able to think longer and performance goes up. But in some cases it's not that we're just unboundedly increasing tokens. You know we end up getting cases where I think omni math is a good example. you're not thinking for that much longer, but your performance continues to go up. So maybe this is a good uh example of sort of this length control kicking in and doing good things.

Um the final thing I want to talk about is you know you might wonder is this actual RL stuff much better than just training on you know correct answers? You know this is called expert iteration. It worked very well in a number of past papers. um in cases where you're dealing with very unstable stuff, you may actually want to do expert iteration instead. Um you know, Kimmy K1 has very large scale ablations showing that um uh these kinds of RL methods uh work consistently better than expert iteration. So that's the orange beating the blue over here. So you can't really avoid RL if you want to squeeze out uh all of your performance.

Okay, I can I can pause here in case anyone has questions about the Kimmy stuff and then I'll uh move on to the the last of the three which is uh Quen 3. Great. Okay. Um so the last one that I want to talk about is Quen 3 and uh also Quen 3.5 Next Coder. The names are getting complicated. Um and Quen 3 is later but they have actually really interesting scaling and data results. So I want to talk about those. Um and the successor to this which is Quen 3.5 Next Coder um is interesting because it I think has the most details of I would say like Asentic RLVR training out of the the major tech reports. Um and they do you know lots of lots of cool things.

Okay. So once again as with Deepseek uh one of the nice things that we see with kind of the Quen report is the full way that all of these are organized. And this is very similar to the deepseek structure, right? Um we do base model to to basically SFT then reasoning RL um and then they have this like thinking mode fusion. I'll talk about that later. And then they do RLHF and then they have a model that gets shipped. But of course they don't want to serve that directly.

So they do distillation to get their smaller models. Right? I mean you know Quen 3 is is quite a respectable open- source model. You can basically have this be your mental picture of how, you know, frontierish uh language models are built, you know, putting all the components together and I think you would have a pretty reasonable picture of what is happening, right? Um and Quinn uses basically the at this point tried and tested playbook for RLVR. Um I think they take lots of the best parts of Kimmy and and Deepseek and they sort of get it right. Um they do a lot of filtering for difficulty because we know that saves compute. that's a good way to get, you know, good data.

Um, they also do other things like, you know, they remove things that the model can get right without coot because we know that's not a thinking problem. Um, they remove things that are too similar to validation data to try to decontaminate things. Um, and then they sort of do a little bit of manual filtering on sort of reference coots. Um maybe most remarkably about Quen 3 is they actually do their RL on very few examples, just 4,000 examples. Um but once again, if you kind of have the rest of the pipeline right, you can actually get surprisingly far. Um so the core of the RLVR components are maybe not super surprising because they are really, you know, building off of uh DeepSeek uh R1 and also Kimmy. Um there are some interesting Quen 3 specific things um some of which you know they've ended up like getting rid of but I think some of the ideas are actually quite interesting. Um so one of the things that they do um is they mix sort of thinking and non-thinking things with tags. Um and that's kind of how they end up having a thinking mode where you know the model has a long coot and they can mix that with a non-thinking mode. So both the instant response model and the long coot model basically live in the same model and this was not true uh in many cases you know like there was a often a thinking mode and a non-thinking model uh even at open AI they also have uh I thought this was quite interesting um which is they have this like way of early exiting thinking where if they append the special string they'll immediately stop the coot and the model is forced to give an answer to whatever you know prompt is given um we see this even in in sort of we see this affordance in interfaces like chatgpt where you're able to like terminate the thinking early and try to get an immediate answer uh for example but what we see with this and I've always thought this was a a kind of surprising and interesting thing which is that as you vary the thinking budget doing like kind of this early termination trick um you end up finding that the performance of the model kind of degrades gracefully like even though you at the lower thinking budgets, these models are kind of getting truncated mid-thought.

They're able to give surprisingly reasonable uh responses um even at that point. And we see that, you know, consistently even with very small thinking budgets, the thinking mode models are much better on all these sort of like mathematical or coding tasks uh compared to uh the sort of instant response mode which is much more of like the classic instruction following plus uh RHF models. Um and so you can then you know one of the nice things they do for us is look at the contribution of all the different components to performance. And in many ways this is not something that should surprise you too much. Um you know if you do sort of this this uh you know reasoning RL and then general RL you're able to slowly get improvements um in all the different sort of tasks you know things like arena hard or counterfact QA. These are general tasks for which you know you want more normal RLHF and we see significant boosts to that performance across the board. Um there are some degradations that we see in sort of math and coding because we fuse together non-thinking uh components but the degradation is not so bad. I think actually you know even though these numbers in an absolute sense are small um in later releases I think in some of the quen 3.5s um they've gone back on fusing both thinking and

non-thinking into a single model um I think what they used to they used to call this hybrid models because they found this kind of drop uh kind of unacceptable they wanted to squeeze out all the juice possible on thinking modes and so they've now I think separated uh these models from each other okay um the very last uh topic and paper uh that I want to cover um is Quen 3 oh sorry I flipped the two it's coder next not next coder um which is one of the newer uh models to come out and I think if you're interested in like you know general agent stuff of course there's other agents things to think about like hardnesses but if you're interested in like how do you actually build an agent um this is probably a good report to go read clen coder next great sort of agent post-training uh paper Um but in some ways post training for an agent is not very different from from everything that we've described. It's not like there's a new like agent training algorithm. Um really um you should internalize this lesson throughout the entire class. Data is the important thing, right? So if you're going to train an agent or like post train for an agent, what are you going to do? You're going to get the data together, right? So there's two parts to the data, you know, for to get a lot of the capabilities. You can't just inject them at the end. like you have to probably start a little bit earlier. And so there's an extensive mid-training phase that happens to try to get as much coding and sort of agent-like capabilities into the model as possible.

Um so they'll do things like they'll take repositories and concatenate the files in the repository to generate these very long context data and put it into the model. Right? We know that the model is eventually going to see these very long coot traces where maybe the agent has opened a bunch of files very useful sort of pre-trading style data. They take pull requests um and then they try to construct like synthetic context for it using rag that is potentially helpful for understanding the pull request. That all goes in mid-training.

um they will take documents that they detect using an automated method to have text and code and then they'll use an LLM to transform this to a nice markdown format. Throw that all in as well. Um and then finally they're going to have a language model you know uh talk about coding on web documents that are coding related to generate sort of codingish synthetic data. Um, and then they'll take publicly available coding agents, run them on various environments, which I'll describe later, and the traces of those runs all go into mid-raining. Um, they'll do some fairly standard stuff as well like instruction following, and they'll have an additional task, which I think is not super relevant um, for us um, for filling in the middle. Like if you're coding and you want to fill in the middle of a span, you know, it's useful to have that ability. So they they throw some of that data into mid-training. Um, but otherwise, you know, this is a very uh standard but kind of in-depth data collection procedure to try to get some of these agentic abilities uh into mid-training.

And then the Quinn folks do something that I don't know if I've seen before. It's actually quite interesting. Um, they'll take the mid-trained Quen 3 Next model um, and they are actually just going to train a whole bunch of different expert models for different kinds of coding adjacent tasks. um and they will end up with four different uh you know agent experts and then they're just going to distill them all back into the same model. Um I can only speculate as to why they do this. I don't think I've seen it in other works. The closest one I know of is DeepSeek V3 or was it V3? V3.5 sorry Deepseek V3.5 um or it's 3.2 maybe 3.2 sorry 3.2 I think uh did this thing where they have different experts whose only job it is to sort of format and process data. So they have data processing experts which generate the data for the full training.

But I don't think I've seen this thing of uh generating different model experts and then distilling them back. That's actually closer to like some academic work like branch train merge um and so on. Those are definitely things people do. I haven't seen it much in frontier model training. That said, each of these experts is actually kind of interesting and what they do is they basically do, you know, full-on RL or andor SFT training on a whole bunch of different subtasks that they've like defined. So, they have a webdev expert where they SFT on valid web code based on different kinds of checks. They'll do things like build UX experts uh trained on many different tool formats. Um, and they'll have QA. So the QA agent is just you know trained on more uh synthetic data for code. Um but then maybe the most interesting for us and the most involved is kind of their uh software engineering agent. Um and what they do is they construct agent environments at scale.

Um so you know Swebench is the is the gold standard of these kinds of things. And so what they want to do is they want SWEBench but more. they go out and they have various automated ways based on GitHub to generate a ton of different uh issues and you know given this they want to basically do RL on these environments in some ways very unsurprising um and what do they get they can get sort of RL performance to go up um one thing I will say here you know and this is very kind of important through the the broader context of what we're talking about you know the reason why we can sort of put more and more compute into RL is because we believe that our reward models are unhackable or difficult to hack, right?

If that assumption breaks down, your RL method will find increasingly obscure ways of cheating you out of your performance. Um, and so they have this great plot of um, basically, you know, in git you can kind of look at future commits or different commits and if you have an issue and there's future commits, you can just kind of look up what the fix was, right? And this is a very easy hack for for models to learn.

Um, and so they basically have a whole reward whose entire purpose it is to prevent the agent from messing with the gate history. And if you don't do this, you end up with a plot that looks like the right uh where it's learning learning learning learning learning and suddenly you get this like kind of emergent jump where the emergent jump was actually it learned how to manipulate the git calls to get sort of the the history. Um, and in some cases it can sort of even sort of hack around certain constraints you give it. Like if you tell it you can't use git log, you know, it might add an origin like a remote and then like query the remote for uh what happened in certain commits.

So there's all sorts of hacking you can do. Um, RLVR is only as robust as your reward. Um, and your rewards can sometimes be not very robust at all. Um, you know, as a as a sort of side comment, you know, one of my students and I were working on um a project that was like RL on lean. Lean is this formal math, you know, verifiable language. Um, and we naively thought at the time, there's no way this can go wrong. You know, lots of people have worked on lean. The lean compiler is bulletproof.

Turns out the lean compiler is not adversarially robust. There are sort of strings that you can put in it that will allow you to verify proofs that are not meant to be verified in certain modes. So, you know, I think the the notion of verifiable rewards is actually much trickier than than many of you might initially think. Um, regardless, you know, you do this process of constructing the these GitHub repos at scale, you go through this

process of doing RL, you know, you see your scores go up, you know, uh, uh, slow but steady, and in the end, you will actually end up getting a system that is surprisingly good, a model that, you know, achieves, uh, something like 70.6% 6% on uh SweetBench. Um even though you have what is basically a really tiny model, a three billion active parameter model, you're able to do, you know, surprisingly well um on these tasks. In some ways though, I don't know if this is super surprising. Like RL, of course, you're going to be able to do well on the environments for which you've like kind of trained. You might even get generalization to your validation set as you do here. Um, but you always want to be a little bit careful about comparing performances. Like task specific performance doesn't necessarily mean it'll generalize uh to broader domains.

Okay. Um, so that's basically all I've got for the RLVR lecture. Um, you know, remember the the core takeaways of this lecture, right? Is it's really all about the reward for RL, right? RLHF and RLVR arguably they're very similar problems but you know with the difference being really just we want more unhackable rewards so that we can actually put in much more compute and get these systems to be much better. Um GRPO at least for the research community is something that you know really enabled a lot of this and you should all know GRPO very well.

You should know you know what the functional form is and how the updates are. This is something that you should know as well as you know uh just pre-training losses. Um and then finally at this point you know I think lots of people know how to do RLVR. Um as you will find out unfortunately RL remains very finicky and noisy and it's kind of painful to work with but it's not that hard. It's not like you know the old days of like you doing PO on various kinds of like really tricky environments. It's actually a lot smoother than you might um think. Okay, I'll stop here for a moment and take any final questions uh before I let you all go in case anyone has questions.

>> Or do they play prompting or do they have different underlying models that they >> Right. Yeah. So the question was about like thinking mode and what's happening on the back. Um so the interesting thing about thinking mode really is that it's actually one model, right? and there's just like a little prompt tag that switches them between long and short co like long coott and no coot almost mode. Um whereas if you just had like a API flag or something, you know, that's not very difficult. Um the fact that they were putting both of them in together is kind of the interesting bit there.

>> So you can actually choose not to but then you add extra prompt yourself to do coot. >> Yeah. Or the control mechanism in this case is in the prompt rather than at the API or serving layer or something like that. Yeah. Great. >> Oh yeah. Um so you mentioned like the phase of the mid training. I was like wondering like how much of the formatting um that goes into the mid training process like informs what is able to be learned during the RLVR and post training and then like as a secondary to that question. Um if you don't have like the correct data in mid training then like is it safe to assume that the reinforcement learning can't like sample solution and therefore you can't learn it?

>> Yeah. So the question was about like the role of mid training in RL. Um, I think you know there there are a lot of like empirical subtleties here like you know I I don't have like a one-sizefits all like it'll definitely not work

kind of an answer but to give a little bit of the nuance I think pre-training and SFT are doing a lot of the heavy lifting as long as you have coverage like if pre-training just doesn't have any code data then you're in trouble you need mid training right but if your pre-training was very diverse and it covered let's let's go back to the mid-training slide you know if it covered a lot of stuff like text code data and it covered a bunch of like GitHub you data, then this is very nice to have, but maybe not critical because you're going to SFT anyway. And the SFT will allow you to get close enough to get start getting some rewards for RL. If you didn't have SFT, then you know, then you're in deep trouble, but the fact that we're always going to be doing some amount of SFT prels you. Um, and pre-training has enough coverage hopefully that you're able to to get some reasonable generalization.

So, mid-training is very nice to have, very important in order to get slightly better generalization. but not necessarily make or break. I think >> for model you said there's a step after train each expert model we're going to distill uh into a single model again so does that how does this kind of distillation work do you kind of have to design a data mixed procedure >> yeah yeah so so that's right like so so this distillation procedure is going to require you to write down a sequence of prompts on which the experts will be distilling into the final model. Yeah.

Um I can definitely see the advantages of of this approach which is that you can have a separate team working on each expert like it's much easier to like sort of you know have teams work on this and then the aggregation could potentially be simple as well if you have enough compute. Um it's just that you know this you know if you have all the objectives you might as well just throw it into the big training loop.

Usually I think that's how you would prefer to do things. That way you don't have to deal with this distillation complication. And you could just have one big training objective that you then do one you know one training loop. >> Yeah. >> Yeah. So the question was is the long reasoning training part of mid-training? Um so I think so there there's long coot SFT in in both like R1 and uh Kimmy K5 1.1. Um, long coot is not traditionally part of like mid-training per se, but it is often like long coott like data is used in long context extension. So it's a little bit more subtle like I didn't talk about long context extension at all and I regret that. Um, but usually that's like an additional phase right before RLHF or other things in this case also long coot where you sort of extend the context using any long context data you have. Usually that's a combination of like books, code, uh synthetic data because those are the things that are long enough to do extension on.

>> Oh yeah. Um so like I guess like related to this diagram that's currently up. Uh when you're doing this RO process, let's say you have like math, chemistry, many other domains. Um is it common to do it in parallel one uh I guess like reinforcement learning training um um like run or like do you do it sequentially and if you do it sequentially like how do you avoid like forgetting failure? Yeah, I think the closest answer to this question of like do you do things sequentially or in parallel is you know these kinds of diagrams where um the two splits that you do is you kind of split amongst like reasoning problems and non-reasoning problems and the reasoning problems all go in one bucket and they get executed at stage two um and then sort of the non-reasoning problems kind of happen at the final RLHF phase um and this includes stuff like chattiness and so on and so forth and those all go into the one later bid which is the the stage four Okay, great. Thanks a lot. See you next

week.