[SQUEAKING]

[RUSTLING]

[CLICKING]

PHILLIP ISOLA: OK. Hi, everyone. Welcome. Thanks for sticking through and coming for the last few lectures. I know things are probably busy. So today, I'm going to give a brand new lecture. So this is a topic that wasn't really a big thing last year and in previous years, but this year, has suddenly become quite the hot topic in deep learning in AI.

Of course, the ideas in this lecture are going to be very old, and they've existed for a long time, but it just has become one of the new things that's happening in 2024. So we thought we really needed to have this as a lecture topic. So this is all about what I'm going to call inference methods for deep learning.

Give a shout out. Some of this was inspired by this tutorial from Sasha Rush, and you should definitely check out that tutorial if you're interested in learning more. And the payoff for this lecture will be that you'll learn how the very latest models, like OpenAI's O1, presumably work. We don't know exactly how they work, but this lecture is my best guess as to how those types of models work.

Oh, before I start, let me say that on Thursday we're going to have-- the class lecture will be office hours. So we will have the various TAs and instructors here. And you're welcome to come to this room and ask us questions, use that entire lecture session. There won't be any talk or any presentation. It will just be for you to come and get advice on your final projects and work together with everyone. So we'll make an announcement about that on Piazza as well.

So inference-- what is inference? OK, this is a little confusing if you have taken a statistics class because the definition of inference in statistics is different than in machine learning. So the statistics definition is that inference is about figuring out properties of a data distribution from samples from that distribution, or figuring out properties about a data generating process. Maybe it's not characterized as a probability distribution, but some generative process, trying to figure out properties of that process from samples from that process.

So what do we call that in machine learning? It can go under different names, but maybe density estimation, or typically, it's some kind of learning or training. So the thing that they talk about in statistics as inferring the parameters that explain the data-generating process-- we call that training a model.

So the ML definition of inference is what statisticians call prediction. It's using a model to make predictions about a new query point, so to make inferences, in the more colloquial sense, about the answer to a new question or the interpretation of a new image, or whatever data you're processing, using the model to answer questions about that data.

So we're going to go with the ML definition. And that's the one that's becoming-- taking over. But statisticians will still complain about this, so be careful when you talk to statisticians

So here's a outline for some of the topics that we'll see. Well, I'm only going to cover some of these today. But this is how I tried to parse the landscape of training machine learning models versus using them to make predictions, so training versus inference. And most of this course has been about the training side. And indeed, most of deep learning as a field has been about training models on data to learn to learn properties of that data.

And the use of those models at test time for what I'm calling inference has been a more minor subject. So far in this class. It's almost entirely been just a forward pass through the trained model. That was all you do at test time. But it turns out you can do much more than that at test time, and if you do, you'll get a lot of advantage.

So we've seen some of these topics, but let me just quickly go over this diagram, we'll come back to it a few times in this lecture So we have two regimes. We have training and inference. So training is what you do in the factory, at the company. It's what you do before you actually come across new problems in the wild. An inference is what you do at test time, in the wild, on deployment, on device. It's what a robot would do as it walks around this room and understands what's going on. So we have those two different phases.

And by the way, these terms and these distinctions are my own attempt at categorizing things. There's a lot of different ways of looking at this. These are not hard boundaries. A few of the terms could go under different categories.

So training is also called learning. It's called statistical inference. And in machine learning, we might call it amortized inference, meaning I'm going to be amortized-- inference is about solving a problem, and training learns a set of weights that directly solves the problem without having to go through the step-by-step reasoning process. So it's amortization of the more extended-thinking inference process,

Inference might go into the prediction, or thinking, reasoning, cognition is, I have a new query, and I want to understand what the answer will be by thinking through things, as opposed to learning, which is more about coming up with a direct, feedforward heuristic mapping from data to answers.

But there's a lot in between. So the most extreme version of training is pre-training, and that's what most of this course has really been about. It's been about representation learning, generative modeling, coming up with foundation models, doing next-token prediction on the internet, coming up with a really good set of weights that is generally useful for a lot of tasks.

But then we said that you can also take that pre-trained representation and apply it to your problem of interest by fine-tuning or transfer learning or adaptation of various kinds. So pre-training-- given a model-- sorry, given data, learn a model or a representation. The training part of post-training or adaptation would be, given a model and new data, or a new task specification, update that model to solve that new task. And the main method for that is fine-tuning, but there's other methods like reinforcement learning from human feedback that we heard a bit about in Jacob's lecture.

Then we're going to move over to the, upon deployment side. So this dotted line is meant to be, now I've taken my AI system, and it's not in the factory anymore. It's not in the server farm. It's out in the world. It's being used by a user. So at deployment time, we can also adapt a model's behavior.

There's a few ways of doing that. We could prompt it with different commands. We could do in-context learning, which I'll mention again, but Jacob also talked about in his lecture. And then there's a few new concepts that I'll introduce-- test-time training.

And there's a few concepts which I won't have time to get to but are also important, like any time you have a model that is using feedback in order to update its behavior, like a robot walking around and feeling its contact and the gravity, direction, direction, and its torques and its joints, and so forth, it's going to use that feedback to change its behavior. So any time you have this feedback control, this is changing a model's behavior based on the test data itself.

And then we have the most extreme side of inference, where we're not actually updating or changing the model's behavior at all. We're just using the model to search through a space of possibilities and find the best one. So given a model and a query, find the best answer to that query. And this is where we get to things that are really outside the realm of traditional deep learning, but now increasingly popular. This is more classical AI search methods, like tree search and breadth-first search and depth-first search and trial and error search and so forth. Cool.

And then I'm going to, at the end of this lecture, talk a little bit about this interesting interplay between search and learning, where you can use search to learn new things, and you can use learning to speed up your search over the space of possibilities.

And one thing I realized is that a very general-purpose name for all the methods that do that are basically reinforcement learning. So reinforcement learning-- we'll come back to this-- as the idea of using search to improve learning. But there's also some more modern things that are really exactly this, like the STAR method and then, presumably, the O1 method from OpenAI, but we don't really know.

OK, so let me come back to "The Bitter Lesson" that I mentioned a few lectures ago. So "The Bitter Lesson," as you might recall, is this essay from Rich Sutton. And he said, the big lesson in AI has been that hand-designed systems with clever inductive biases and hand-wrought algorithms-- they just don't work that well. There's only, really, two things that work. It's simple methods at scale. And the two that we know of are search and learning.

So a lot of people have focused on the learning part. That's the deep learning idea. Just collect a lot of data, and train a network to imitate that data. But Sutton specified search as the other option, and so let's talk about what is search.

And this is what's happening, really, in industry right now, is this move away from only focusing on train-time compute, but now using a lot of test-time compute or search at test time. So this is a plot from the blog post on the OpenAI O1 model, their latest model.

And what this plot is showing on the left is that as I train with more flops, I get better results. That's the scaling law. Train-time compute is I'm training on more data, or I'm doing more iterations of gradient descent, or this might, in fact, be some kind of RL training part of the pipeline. But whatever it is, if I train on more and more data, I'm going to get better and better accuracy, and there might be some kind of power law that relates my training time to my test performance.

So notice the slope here is maybe 0.5. The really interesting thing is they said, well, wait a second. What if I scale up the amount of compute I apply at test time to do intelligent inference or search through my model? We'll talk about how that can be done. And they saw that you also can get a power law. And in fact, the slope in this particular case is even steeper than the train-time power law.

So the big move that a lot of companies are making right now is from only focusing on deploying compute at train time to now deploying a lot more compute at test time, not just the forward pass through the model, but many, many passes through the model at test time. So that's the big change from a few years ago.

OK, so what is search? I think a lot of you have taken Intro AI and Algorithms. So you know, basically, that search is methods from those classes, but let's go over it. So search is the way that I used to be done. So in 1994, maybe, Garry Kasparov was beaten by this chess machine called Deep Blue. And the way that AI worked was it was just based on search. It had very little or maybe no learning or anything you'd really want to call learning.

It just did look-ahead search. It said, well, if Kasparov makes this move, then I could respond with this move, and then he might make that move, and I could make this move. And it looked through this chain of all the possibilities, 12, 10, 20, 30 moves in advance. And it just made this huge, huge chain. The branching factor of that huge, huge tree-- the branching factor of that tree was, in chess, however many moves you can make at each position in the chess game.

So it turned out that chess does not have that big a branching factor, so you could enumerate all possible futures and just take the set of actions that will lead to the best possible future. So just searching over all the things that you could do. Any of you play chess-- you probably do this in your head. You think about all the possibilities, and then you pick the move that will maximize your expected reward against all those possibilities.

OK, then some years later, AI systems won that Go with the AlphaGo system, which you've probably heard about. And AlphaGo also heavily uses search. But AlphaGo uses both search and learning. So it's one of these methods that uses a combination of the two. But I'm going to focus on the search part right now.

So what they did is they, again, said here's a particular game state. Let's just expand out the tree of all possible futures. So after this game state, my opponent could move in these two locations. Maybe there's other locations they can move to. And then I'll descend the tree and say, well, if they move here, then here's what I should do. And I'll search through this space of all these possibilities.

Now, the branching factor, the number of branches in that tree for the game of Go is much higher than the game of chess. And that's why they weren't able to do brute-force search like they did in chess. Instead, they had to do search with heuristics-- with learned heuristics on top. We'll come back to that, too.

OK, so there's this general observation that there's these two things that just get better with more compute. One is learning, and the other is search. And sometimes they trade off, and you can get a good AI system with one or the other.

And this is an interesting paper from Jones on scaling laws of board games, where they actually said, well, we can solve chess either by taking a million chess games or a billion chess games and learning the mapping from the board state to the best action that Garry Kasparov would make. Or we could solve chess by just not having any pre-trained learning of optimal moves, but instead just, for every game position, search ahead in time and see that tree of possibilities, and take the move that maximizes the expected value.

And there's a trade off, like, a Pareto front, like, I can-- well, yeah, it's just a trade-off curve. It's saying that I can do more search, and then I can get the same elo for less training, or I can do less search, and then I can get the same elo for more training. So there's two different ways of being smart, search or learning.

And sometimes it's better to use search because I don't have to spend as much time learning, and maybe it scales better in some sense. And sometimes it's better to do learning because in other situations, that might scale better. And also, learning means that I don't, at test time, have to do a lot of compute. So search is test time. Learning is train time. OK.

So now let's talk about some search methods that are used in deep learning. And I'm going to talk about these three in particular, Best-of-N, beam search, and Chain-of-Thought. We already saw Chain-of-Thought a little bit.

So the first thing that I want to convince you of is that the language models that we have, which are autoregressive models, can be improved by sampling multiple possible completions and picking the best one. So why can they be improved? Let's look at how these systems are trained. So this slide is a little bit old, so I was showing the example with an LSTM. You can replace it with a transformer, but it doesn't really matter. So let's just keep it as an LSTM. So here, we have a molecule. Anyone recognize what that molecule is? Somebody knows it.

AUDIENCE: Caffeine.

PHILLIP ISOLA: Caffeine, yes. So we're going to have a graph net that will process this. Maybe your final project, you're putting together different architectures that suit the data. So graph net makes sense on a molecule. And then we're going to have an LSTM that will spit out a sentence that describes, this is molecule to sentence, captioning a molecule. So it's going to say, oh, this is a mild stimulant that enhances cognitive abilities. That's caffeine.

So here's how you train that system. We're going to have a data that we're imitating. This will be just generative modeling. So we're going to try to maximize the likelihood of the target data, which is just sentences from some corpus. And we will output a categorical distribution over the next token and a sample from that in order to produce sentences.

So we're going to try to maximize the likelihood of the sentence, or the characters y at every point in this sequence. That other picture of that y is going to be a one-hot probability vector. And we're going to make a prediction. I'm going to take the cross entropy. That's just doing the pointwise product between the one-hot vector and the predicted probability vector. So that's cross-entropy minimization, so this is stuff you've seen before. Oh, the color changed. OK, anyway.

So there's this interesting thing that you do. In transformers, this stopped being a term. But in the LSTM era, we used to talk about the way of training this thing as teacher forcing. So what does teacher forcing mean? Teacher forcing means I'm going to take the target sentence. I'm going to shift it by 1, just like I would normally do in a transformer training, an autoregressive model.

And I'm actually going to tell the LSTM-- rather than just reading off the entire sentence, I will tell it the ground-truth previous word at each time step is whatever it was in the training data. So I'm going to condition each next-word prediction on all of the ground-truth previous words.

And this is where you can see there's a deviation between how we'll use this system at test time and how it was trained. It's trained to predict the next word given the ground-truth previous words. But at test time, at deployment time, we're going to condition each next prediction on the model's own self-generated previous prediction.

So that's a difference. That means that maximizing the likelihood on the training data will have a gap from what we really would want, which is to maximize the likelihood of entire sequences sampled from the model. So it's actually going to be maximizing the likelihood of the model's predictions of every next character or word, given the ground-truth previous words, which is not what you actually have at test time.

So this teacher forcing way of training means that you're not actually going to be maximizing the likelihood of the caption given the data at test time, because here's what we do at inference or at test time. We sample from the predicted distribution over the next words. Or maybe we do greedy sampling, where we just take the most likely next token given the previous tokens.

So first, we'll sample the word A, and then we'll condition the model on that we sampled A. So now what comes next? A strong and so forth. And this is how you sample from a autoregressive language model. And again, I'm showing it with an LSTM, but you do exactly the same thing in transformer. You condition every next-token prediction on all the previous sampled tokens. You're just writing one word at a time, or one token at a time.

So now, because these are samples from the model, they might differ from the ground-truth distribution that the model was trained on. And there'll be this kind of drift that occurs. The model will produce something that's a little bit different than the true data distribution, and then its next predictions will be of distribution compared to how it was trained. OK. Yeah. So then it gets to the end.

So the important point here is that autoregressive greedy sampling-- the way that-- either greedy sampling or non-greedy sampling, but let's say autoregressive greedy sampling does not maximize the joint probability of the entire sequence. So remember, the autoregressive model is a model of the probability of a sequence of tokens. And it decomposes by this chain rule as probability of the first token, then the second one given the first one. This is the underlying probability model of autoregressive models.

Now, I really want to find the sequence, in this case, the caption or the sentence, that maximizes the joint probability that my model assigns to the data. That will be some sequence. But with greedy sampling, I will first take the most likely first word. And then I'll say, given that word, take the most likely second word, and so on. This sequence, x1 star, x2 star, xn star, is not going to equal the optimal joint configuration. I've used the same notation, but those are not the same values. You can construct counterexamples where that's not the case.

So an observation here is, could we do better than greedy sampling? Could we come up with a different way of sampling that gets higher likelihood configurations of the data? Yeah, question.

AUDIENCE: So just to confirm what you said. It's not an equal sign, then, right? [INAUDIBLE] are not the same.

PHILLIP ISOLA: Sorry, the equal sign is true, but the argmax solution is not the same for the argmax over the joint configuration, argmax over the individual conditionals. Yeah. Yeah, I probably should have had a prime or something to indicate that those were different values.

So question for you to think about. So this is true for autoregressive models, that if you take the samples that maximize the sequence of conditional probabilities, you don't necessarily get the max likelihood joint configuration. What about for VAEs? What if you sample from a VAE and you take the argmax, the most likely sample-- like, you just-- you encode, and now it's a Gaussian VAE, so the most likely sample is the mean of that VAE. Do you get the same issue? What do you think? This takes a little bit of thought.

How about for a diffusion model? If I do denoising step by step, is my final output-- and I do greedy denoising, right? So I'm denoising, and I'm going to predict the most likely noiseless-- less noisy image, and then I'll do that iteratively. Will I get the most likely estimate of the noiseless image given whatever I condition on? What do you think? Anyone want to give a guess? It's a little tricky.

OK, so it's the same exact thing with diffusion models. So diffusion models are also iterative step by step. In every sampling step, you're going to try to remove a little bit of noise but not all the noise. And if you do that and you make your prediction and it's going to try to maximize the conditional probability of slightly less noisy given slightly more noisy, than that won't necessarily result in a sequence of samples that- a sequence of denoising steps that maximizes the probability at the end of the sequence for the same kind of reason.

But with VAEs, it's not really sequential. There's just one step. There's just encode, decode. So I think with VAEs, if I'm actually having a VAE that models a full sentence, there's no sequential process where my sampling could be deviating from the most likely sample for the entire joint configuration. OK, something more for you to think about after the lecture.

But I will say one thing. I mentioned the diffusion models because I think there's not as much work in trying to do intelligent inference through diffusion models. There's a lot of work on doing intelligent inference that's better than greedy sampling for language models that are autoregressive, but less for diffusion models. So maybe an interesting open topic for you to work on. How do you take higher likelihood samples from a diffusion model than you'd get with the regular denoising process? I'm sure there's some work on that, but just not as much.

So now let's look at some fancier inference strategies than just doing autoregressive greedy sampling. Also, remember that in most language models, you just take random samples. You predict a distribution over the next token, and then you randomly sample from that distribution.

And certainly, a random sample is not necessarily going to be the most likely sample. So if I want to answer a question like, what is the caption of this molecule, I might get a random description that is low probability just by chance. That's why we want to do better inference to get higher probability samples so we can answer questions more accurately.

So the Best-of-N approach is really simple. It just says, go with your random sampling, and do it N times. Take N samples from your autoregressive models, independently and identically, and then evaluate the total likelihood of each sequence that you generated.

So remember, when you're sampling these sequences, you're not sampling the max likelihood sequence. You're sampling a random sequence from the distribution. But you can evaluate the likelihood of any sequence using the chain rule. So I can evaluate the probability of whatever sequence I generated by just multiplying all the conditional probabilities under my model together.

So here, I sample one sequence, a stimulant that enhances cognitive function. And then I run that through my autoregressive model by just multiplying all the conditional probabilities together, and I get 0.7. And then I sample another time. This time, I got a different random sample a stimulate that improves thinking. When you play with a language model, type in the same prompt twice. You'll get two different answers, typically, because it's sampling from a distribution.

OK, here's another one. And then maybe I'll say-- I'll do this N times, and I will say, this is the one that maximizes the probability. So this is the most likely caption in my-- according to my model or my data distribution. Yeah.

AUDIENCE: At each time step, when you sample the next token in the output sequence, is it with proportion to the probability distribution?

PHILLIP ISOLA: Yeah. So the way that autoregressive sampling usually works is you output a categorical distribution over the next token, given the previous ones. That's each of the conditionals. And then you just take a sample from that distribution. There's a parameter called the temperature, where you can scale how flat or spiky the distribution is.

And so if you set the temperature to 1, then you're taking unbiased samples from the data distribution. But sometimes you want to scale it differently in order to get mode-seeking behavior, like you just want to take the likelihood samples. Yeah.

AUDIENCE: Does this mean that you're going to try with, say, all possible combinations of [INAUDIBLE]? Because I guess it's going to be very computationally expensive.

PHILLIP ISOLA: Right, good question. So are you to try with all possible combinations? You can make that argument and say, well, if I did try with all possible combinations, then this would work. Like, I could find, truly, the max likelihood sample under my model. But that would be incredibly expensive. So instead, you only-- you sample from the model, and maybe you turn the temperature down a little bit. So you take-- it's trying to take high-probability samples.

But if you turn it down to 0, then it's going to take the greedy sample, and that might not maximize probability. So you want it to be a temperature that's non-zero, but not like infinity, where you'd just be searching over the space of all possible random sequences. Yeah.

AUDIENCE: Would it make sense to think of a heuristic where this is your starting point, but then you improve it? For example, you have to take the 0.8, and you create a small improvement where you take chemical out, you put stimulant in, and then maybe now you get 0.9.

PHILLIP ISOLA: Yeah, that's a great idea. So could you take this solution and then say, what if I change one of the words? Will I get a better solution? So in general, this is the whole space of optimization and statistical inference. Like, you could do MCMC, where you start to-- you could do Gibbs sampling. You could do all kinds of interesting things where you take your current solution and you try to just modify one dimension of it at a time, or you try to do hill climbing, where you just look for a local perturbation that improves it.

So there's just a very rich space of methods that can try to optimize over that sequence to improve it. You could take a language model and say, hey, rewrite this in 10 different ways that you think might improve the performance. That's another thing that people are doing. OK. But Best-of-N-- interestingly enough, to my knowledge, this Best-of-N sampling is about as good as anything right now. And it's like most naive-- It's rejection sampling, essentially. Or it's just, sample 10 independent things or N independent things, and take the best one, and reject all the other ones.

But there's smarter sampling methods. And one of the other sampling methods that's quite popular is called beam search. So this is, like, a tree search method. It's a little bit like what you were suggesting. So here, what you do is at every step of your autoregressive generation, you maintain a memory of the top-k best next words, and you start expanding out not just the best sample or a single sample, but more than one sample at a time.

So maybe I'll say, well, strong or mild are both equally probable. And then given strong, then there's these two other possibilities that are the highest probability possibilities. And I'll expand this tree out with some branching factor k. And then at the end of the day I'll just look at the path along that tree that has max probability. So a little bit of-- it's like a tree search method, as opposed to just like an independent random search method, but otherwise it's the same idea.

So again, there's a lot of other fancy search methods that you could use. Best-of-N seems to be the simplest and also works about as well as anything, as far as I can tell right now. And beam search is maybe the most popular, slightly more advanced method. So in language models, you might have an option like beam search on or off, so you can try playing with that.

Now, what about Chain-of-Thought? We talked about chain of thought, or Jacob Andreas, in his talk, talked about chain of thought. Chain-of-Thought is-- I'll refer to it as an emergent or learned search. So here's an example. This is just from ChatGPT last night. So I want you to use beam search to solve the following problem.

I could just use beam search. I could say, what is 4 times 5 plus 10? That's the problem. I could put that through my LLM, expand out all the possible sequences of completions, and then take the one that has highest probability, and that might be a better solution. As long as probability is related to actual accuracy on math, which it might or might not be, that would be a better solution.

But instead, I'll just say, maybe if I prompt you, you can do that beam search yourself because you've observed, in your training corpus, examples of this kind of behavior. Or maybe you've been tuned with reinforcement learning and other methods. But basically, it does work.

So here's what it does. It output a whole bunch of preamble that I'm going to skip. And then it says, OK, here's my beam search of two different sampling trajectories. And it turns out I think the better one is the 4 times 5 plus 10 equals 30. So it could go 4 times 5 plus 10. And then I say, my next sequence of tokens will be- I'll parse that as 20 plus 10, and then finally, I'll get to 30.

So what is Chain-of-Thought? Chain-of-Thought is like doing a reasoning or search process that could be written with a classical algorithm, but it's the LLM doing it itself and writing the intermediate steps into the text space, into its context, and then autoregressively continuing on. So one way of thinking about Chain-of-Thought.

So that was all about how to get higher likelihood sequences from an autoregressive model by doing some kind of search or optimization over the space of possible sequences that are somewhat likely under the model. So you're being guided by the model, the model's likelihood, but then you're trying to find even more likely configurations. But it turns out you can do the same exact thing, but not for likelihood, but for any kind of scoring function.

So I could say, what if my scoring function is a measure of how fun or whatever fun-ness? So this caption's not very fun. It sounds kind of scientific. OK, here's one. So what is-- this molecule is nature's tiny cheerleader, jumping into your bloodstream with pompoms. The language model gave me that. I didn't come up with that. Anyway, that's more fun. And then maybe it will output emojis, and that's the best.

And so now what I can do is I can sample Best-of-N, but not according to likelihood or a learned model but something else. Maybe this is people online. This is people voting online. Maybe it's humans. And then I'll say-- OK, I'll take that one.

And this is a really popular thing to do right now. This is the basis of what's called reinforcement learning from human feedback, where we give humans-- rate all the sequences, and then you pick the best one that the humans rated. And you could change it to any function. Here's scientific accuracy. It's a different function. Maybe I downloaded a model of scientific accuracy, and now it picks this as the best description. Yeah.

AUDIENCE: So in this as your score, is that another model that you're training?

PHILLIP ISOLA: It can be absolutely anything. So something that takes in a sentence and gives you a scalar out. It could be another model that you've trained. It could be a human brain. It could be an animal brain. It could be a formal proof checker. I'll show you another example. This is a really popular type of scoring function called verification.

Verification usually has the connotation that you're going to actually check that the output is correct using some formal process. In this case, the formal process might be a compiler or a mathematical proof checker, or maybe the verifier here.

So here, we have- the input is, write a Python program to compute the Fibonacci sequence. The language model outputs two possibilities. It could output infinite possibilities, but I'm just showing two. And then the verifier could be something that has- it knows what the Fibonacci sequence is. So it can check. It can run the program and see if the program matches the Fibonacci sequence.

So the verifier, in this case-- if I have the Fibonacci sequence, I can check that the output of the LLM matches. And if it does, then I say that that's my solution, and I can just do Best-of-N in this fashion.

And why is this a good idea? Fundamentally, it's a good idea because verification is easier than generation. So checking that I correctly solved a math problem or I correctly made code that has some desirable property is just fundamentally easier than coming up with the code that has that property, or coming up with the solution to the math problem.

So proving a theorem is fundamentally harder than checking that a proof is correct. This goes back to things like t the class of problems that these are checkable in polynomial time, but we don't know if-- and we expect that they don't have a way of solving the problem in polynomial time, as long as P does not equal NP. So verification can be really easy. You just check that you actually output the Fibonacci sequence. But how did I write an algorithm that does that? It might not be trivial.

Another example of this would be verifying that-- doing prime factorization, coming up with-- finding the prime factors for some big number is a hard problem, but checking that some set of factors multiply to equal the number is an easy problem. So verification can be the scoring method.

So you can do all these same methods, not just for language models and autoregressive models, but also for other types of generative models. So I'm going to take a little aside into some work that I did, now quite a few years ago, but I have I have slides on it, so I thought it would be fun. So this is going to be trying to steer models toward human preferences, so a little like this thing, RLHF.

So what are the great objectives for AI? This class is almost entirely focused on just one, which is, given lots of wonderful data, just imitate it. So this is supervised learning. This is generative modeling, maximizing the likelihood of the data under the model, self-supervised learning, same thing. What other objectives could there be for training an AI system besides this? This was, like, 90% of the class. What do you think? What other objective might you want to maximize for an AI system? I guess somebody-- yeah?

AUDIENCE: I guess-- say if it's some sort of game or something with scoring, you could try to surpass--

PHILLIP ISOLA: Try to beat the opponent, win the game if it's a game. That's a good one. Yeah. Any others? Yeah.

AUDIENCE: Interpretability?

PHILLIP ISOLA: Yeah, I want to make the most interpretable system. So all of those are usually handcrafted, but there's this whole other-- so I just said, there could be a lot of things, but I'm going to suggest one. But basically, all of the other ones go under this area of reinforcement learning in some way or another. Or they can often be phrased that way, as you have some other criteria that you care about, like winning the game or being interpretable, and then you have some score that you design for that, and you optimize that score.

So other than imitate wonderful data, I'm going to suggest this one in this section, which is, could we make imagery that people find meaningful, that they find aesthetic or beautiful or informative or memorable or interesting? So we're going to take a generative model and tune it to make outputs-- an image-generative model and tune it to make outputs that humans rate as being more desirable images?

So let's look at how you can do this search over through the space of generative outputs using a scoring function, but not with an autoregressive model, but now with an image generator. And it's going to be the type of image generator that has latent variables, z, and maps z through g of z to an output image. So that could be a GAN or the decoder of a VAE, or it could be a diffusion model where z is the noise. And I'll come to the question in a second.

So we're also going to have a scoring function, which will, in this case, be derived from CLIP as just an example. So it's going to be an encoder for an image and an encoder for text. And the score will be, how well does the image match the text/ So the text input is, what is the answer to the ultimate question of life, the universe, and everything? I'm going to try to maximize the dot product between the text and the image.

The image encoder and text encoder are pre-trained with the CLIP objective, so remember, that will try to make it so that the caption that matches an image will have the maximal dot product. So this is basically going to try to search. If I search over the space, z, optimize over z, I'm going to search over the latent variables that will create an image that matches this text according to the CLIP model.

So this is kind of constrained optimization. I'm not searching over the space of all possible pixels. I'm using my image generator. I'm only searching through the latent space. So I'm only going to be making outputs that are high-probability under the generative models distribution, as long as I'm constraining z to be high prior probability. Remember that things like VAEs have a prior over z, which is Gaussian.

So if I just search over the high-probability part of the unit Gaussian, maybe I just search over the unit hypersphere, then I'm going to be staying within the manifold of high-probability images according to the image generator, and I'll find the one that matches this text

So here's what that looks like So this was-- the first batch of text-to-image generative models that were really impressive were of this kind. This method is called CLIP + VQGAN. It was very popular for a year or two in, like, 2020, and then it got replaced by diffusion models. But it was a pretty cool thing to do, and it already worked back then. OK, there was a question.

AUDIENCE: Yeah. I just had a question out of the top of my head. You said that you may have a different objective for a problem. Let's say you have two things that you want-- for example-- I don't know-- likelihood plus-- I would say creativity. How can you manage to have a model that builds both [INAUDIBLE]? Because I feel like usually they would just create a loss function that would account for these.

But either [INAUDIBLE] way, for example, in the middle of the training, you'll change the objective function, or during the reinforcement learning period, you'll change the function, whatever [INAUDIBLE].

PHILLIP ISOLA: Right. So yeah, how do you trade off between-- if you have two different loss functions or criteria-- I care about creativity, and I also care about high probability, and I care about fairness and not being toxic. I care about all these things.

The general name for the field that studies this is multi-objective optimization. And one way of doing it is you just add all your objectives together, weighted by your-- how much you care about each one. But sometimes this-- it could be one objective just scaled very differently than another one. And it's not easy to come up with weights that will give you a good trade-off between them.

So in general, there's a whole set of other techniques. And one of the concepts that comes up in that is the Pareto frontier. I want to find the frontier where I'm maximizing every objective as much as possible without trading-- without causing other loss functions to decrease in their measure. Yeah. So multi-objective optimization and Pareto frontier are some terms to look up.

Right. That's actually a costly optimization process. This is test-time compute to solve a problem that in 2020 you could not solve at train time. So there are no generative models that made the text to image with nearly this quality in 2020. But CLIP + VQGAN, or maybe it was 2021. It was whenever it was-- it could do better because it was using a lot of test-time compute to solve a search process at test time that hadn't yet been amortized into a feed-forward generative model. So this was a fairly weak model, and it was doing optimization plus the CLIP encoder model. And that made this weak model much stronger.

So another thing that you can do that's become quite popular is to say, why don't we replace the evaluation function with a model of humans? Sometimes it's called a reward model. And I'm going to go back to work, now, from 2018 or so that was fun.

So basically, what we did in that work is we have a model of how much people will remember an image or not. And I'm sorry to show you this photo because you won't ever be able to forget it. This is the most memorable photo in our entire data set. Actually, have any of you seen this photo before? I've given talks like this a bunch of times. No? OK. If you've seen me talk about memorability, maybe you would have remembered that photo. Anyway.

So the human looks at this, and we measure how well they remember it using a psychology test, psychophysics test. And then we model that behavior, and then we'll try to make images that are as memorable as possible.

So here was the way we measured memory. We had people watch movies with images flashing on and off, and they would say whenever they saw a repeat. And this guy is the most memorable, and these are the least memorable. And it turns out people are super memorable, so that was the main finding. OK, so now can we optimize images to be better remembered?

OK, this is work that Laura and Alex led, and here's how it worked. So basically, you had a generative model, and you had a predictor of how memorable an image is. And you had a manipulation, a search process that you would do over the latent variable space of that generative model. And now, what you can do is you can just-- actually, this is all differentiable. So this type of search process is, in some ways, easier than with autoregressive language models. In language models, you have to have these tree search, these symbolic search methods.

In this pixel search problem, everything is differentiable. So you just can do gradient descent to do the search. So you can just backprop all the way to the manipulation of the latent variables that will maximize the objective of being more memorable.

OK, so here's this dog, and here's what it looks like when you make it more or less memorable. So you zoom in, and the eyes get really big. That's making it more memorable. You can actually measure that, and people do remember that photo better. And then if you make it less memorable, it will become grainy and hard to interpret and small.

So we presented that as a probe into human perception. But this type of method has become more popular as-- under names like reinforcement learning from human feedback, where you're trying to make language models output data that humans-- a model of a human ends up saying is good data and is preferable or aesthetic or meaningful or non-toxic or fair or whatever.

So here's a few more examples. So you can also optimize in the latent space to maximize a model of aesthetic preference-- that's in the middle-- or to optimize the model of emotional-- sorry. In the middle row is aesthetics, and on the bottom is emotional valence. So each of these concepts is a little bit different, and you can visualize them in this way.

So simple symmetric structures, big, centered objects are easy to remember. So this is the most memorable version of that snake. But the most aesthetic version of the snake is actually more like-- it has this depth-of-field effect. It's a little bit hard to see. It's at a 3/4 angle. The most aesthetic landscape is going to be really saturated, but the most memorable landscape will make everything big. People remember things when they're big.

The most aesthetic dog is really cute, but doesn't have this big head, but the big head and big eyes make it more memorable. So you can see interesting differences. You're just optimizing-- searching through the space of generative possibilities to find the one that maximizes the score function. Yeah.

AUDIENCE: Is the significance from this result really-- it's of decoding how our brain works already.

PHILLIP ISOLA: That was the intent of the paper was it was more like a psychology paper. But I'm trying to connect it here to the trend of just optimized human preferences. But yeah, that's exactly the intent. Yep.

AUDIENCE: I wonder how much, in the research, if the representation of the objects in an image versus how the image looks was evaluated in terms of how the model understands whether the aesthetic or the emotional balance is connected with the dog or the layout of the picture. Yeah. I wonder if there were any takeaways in that aspect.

PHILLIP ISOLA: Yeah, I would say, look up the paper. It's called GANalyze-- it has-- GANalyze is the keyword. So what factors determine aesthetics or memorability? Is it layout, or is it the central object, or is it some of the other properties, the color? Is it shape? We have some analysis of that. But generally, I would say it's-- the main object matters the most, and being centered. So layout does matter. Being centered helps a lot, and being close up helps a lot.

AUDIENCE: And in the loss function, there wasn't some sort of-- there wasn't some of characterization of where the object that we're interested in is in the image. It was just [INAUDIBLE] that was there.

PHILLIP ISOLA: Yeah, so the loss function is simply a neural net that is trained to predict-- to regress how likely a human is to remember this image, and it's fit to human data. OK. Yeah, question over there.

AUDIENCE: [INAUDIBLE] wondering about the perception. So when we're running things where it's subjective to human perception-- so this one-- memorability, aesthetics, we're measuring something-- to what extent do we need to account for cultural differences or demographic, age differences?

PHILLIP ISOLA: Yeah. So to what extent is this dependent on culture, age, individual characteristics? So in psychology, there's this term, individual differences, which is about all of those factors. It's like there's one set of factors that are common and shared across all humans, and there's other factors-- you model humans as being a common peace and a peace that's individual to them.

And this is only looking at the common piece, and we didn't do any controls on culture or age or background. I think there are statistics on the participant demographics in the paper, but we never made any comparison. That would be really interesting.

But oftentimes, what you might want to do is find the common thing that characterizes a general population first, and then-- and then the second-order effect is maybe younger people like this more than that, and older people like that more than this, and it's a secondary effect. So I think that would be a really interesting future work.

So that was a little bit of a tangent into memorability and aesthetics. But the point is that if you have a really good generative model, then you don't have to just use its raw outputs. You can actually optimize and search and take different samples and do multiple rollouts and change the latent variables to maximize some function of interest, either the likelihood or a different scoring function.

Well, there's also a new paradigm to do all of this stuff, which is what I think Eric Jang nicely described in this blog post as the "just ask" paradigm. It's the prompting paradigm. It's the Chain-of-Thought paradigm. Rather than doing beam search, you can say-- just ask the model to do beam search. Don't actually do it. Just say, do beam search, or, let's think step by step. Now it's going to do some kind of sequential reasoning process, instead of having to hard code some sequential reasoning process.

So Chain-of-Thought is, just ask for symbolic search over reasoning that's written in language. And can we do the same? Just ask for this continuous reward optimization. I have some aesthetics reward function. Can I just say, give me an image that's aesthetic level 10, aesthetic level 11?

So interestingly enough, this works. So here was just another fun example from a few years ago. So here is a prompt to DALL-E 1, I believe, which is a painting of a mountain next to a waterfall. You get some OK paintings. But what if I say, a painting of a beautiful mountain-- or a beautiful painting of a mountain next to waterfall? It's a little bit more saturated. It's a little more beautiful. It's like I steered it in the direction that the model has interpreted as being related to beauty and aesthetics.

What if I say, a very beautiful painting of a mountain next to a waterfall? It's getting a little better. What if I say a, very, very-- now, two units of reward? A little better still. OK. How many times can I do it, do you think? What about, very, very, very, very, very, very, very? It's definitely more beautiful, right? So then you can go extreme. I think it maxed out around there.

So these models did learn this conditional distribution of conditioned on the reward I want to achieve, what should the output look like? I don't actually have to necessarily search explicitly for reward-maximizing images. I can just tell it, conditioned on this reward, you should make a painting that matches that reward.

So people have shown this in a lot of different contexts. Now, here's another fun one from Jon Barron. He says, this is a Level 2 Reuben. And now he says, make me a level 1 to the-- 1e15 Reuben. So you can check out his these things. There was another trend a few years ago called, make it more. You just take an image, and you say, make it more X; make it more X; make it more X. You keep on doing it. And eventually, it always devolves into this like picture of the universe. That's the most-- this is a 1e40 Reuben. And in this sequence of tweets, there's more. There's level 2, 5, 10, and so forth. And it does scale.

But it's not perfect. These emergent abilities, these learned abilities-- they're not like the abilities that you get by doing well-understood search algorithms because they're just going to depend on what the model-- how the model learned from the data to do that type of search. So it's going to depend on the data distribution on which it was trained. It's not going to be something we have guarantees or proofs for.

So here, we say 1e negative 40 Reuben, and it doesn't make a tiny, tiny Reuben. It, instead, makes this microscope because it has that connotation. That's what the data told it to do. OK. Yeah, question.

AUDIENCE: So understand this for images-- on the first slide, you had just prompt for-- in the text form, for a Chain-of-Thought and like beam search, I think. Do you explicitly have to code that into text models if you want to implement that kind of search at the end because of the autoregressive nature of it?

PHILLIP ISOLA: Do you mean do you have to have the language model trained in a way that anticipates that you'll be using it with beam search?

AUDIENCE: Yeah.

PHILLIP ISOLA: Yeah, that could potentially help. I think a lot of search algorithms are meant to be very broad-use, that whatever-- any autoregressive model will be improved by this generic, Best-of-N search approach. It's very safe. But if you have some search approach which is more like, my search will be-- I'll prompt the system to, let's think step by step, then that will depend a lot on how it was trained, and you want to couple those two processes.

You want to make sure that the language model was trained on data where whenever it said, let's think step by step, it had a really good reasoning process. And that was an empirical discovery that that works well, but you could also design your data set in conjunction with the search algorithm if you wanted to. We'll talk a little bit about that at the end.

So that was all about search. Now let's go into inference methods that actually change the model's behavior, maybe update its weights, maybe do something else. And then they update the model's behavior in such a way that over time, during the test sequence-- maybe I have a self-driving car, and it's driving around, and it's getting smarter and smarter as it drives around the city. So it's actually a different model at the end of the trip than it was at the beginning. It's a smarter model at the end of the trip. It's learned something new.

So there's a lot of methods that do this. The first I'll mention is in-context learning, but this is just a reminder. So in-context learning is-- I have a data sequence that I'm processing, and the sequence includes examples of desirable behavior. So the sequence is a supervised learning sequence. Input x, output y, another input x, another output y. So x1, y1, x2, y2, and so forth.

And then at the very end of the sequence I have x query, and I have-- and the language model, the autoregressive model should learn to complete that sequence. And what would be a natural completion? It would be to follow the pattern.

And what is the name of following a pattern of x, y example pairs? The name of that is supervised learning. So in-context learning is just the definition of supervised learning. It's learning to do supervised learning. It's a transformer that is-- or an autoregressive language model, in this case, that's learning to do supervised learning because supervised learning is the problem of taking a sequence of x, y examples followed by a query and outputting the answer to the query.

So we saw that before. So I want to now talk about a slightly different way of doing that, which is called test-time training. So there's this really interesting paper from a few years ago that tried to understand, what is in-context learning doing? Is in-context learning a learning algorithm like the ones we know, like gradient descent on a supervised regression loss? Or is in-context learning something fundamentally new?

And they set things up like this. They said, let's let f parameterized by theta be a transformer; x is a query; and c is the context. So the context is going to be the in-context examples, x, y, demonstrations, and then we have a new query. We want to complete it based on the patterns in the context. So we're going to rewrite f of x comma context. That's like-- can you concatenate the context with the query? That's the input to the language model. We're going to rewrite that as another function, f of x, with parameters theta prime And we're going to try to see if these two things can be equal.

So what's the relationship between theta and theta prime? There's some way of taking all the information in the context and the attention mechanism and getting rid of all of that, but somehow baking it, instead, into a set of weights that has the same effect. Let's say that the transformer is infinitely big. These are universal approximators. So this approximator can certainly approximate that approximator. This is a universal approximator, right?

But what's the relationship between theta and theta prime? And the really interesting thing that they showed is that there are cases where the relationship between theta and theta prime is theta prime is gradient descent on theta with respect to a particular loss function.

So I take my theta from my original transformer. I then do gradient descent to update the parameters according to some loss, which is dependent on the context. The context is part of that loss. And I will get a new transformer that, when not showing any context at all, just via the weights, it will output the same result as the original transformer with that context.

So what they're saying is that this original transformer with this context is doing something like what you would have gotten if you just did gradient descent on that context with respect to the weights. It's a little confusing. Does it make sense?

So in certain cases-- they showed this for one layer, linear self-attention. It's not the full picture, but they showed this for a simple toy case. And in certain cases, like that one that they showed, you can express in-context learning as gradient descent over the context examples to update the final mapping from the query to the answer, where that mapping is only specified by the weights and biases of the transformer, not by the context. Yeah.

AUDIENCE: Is the argument something to the effect of showing that the activations within the transformer, after having processed the context, could be computed as the original with gradient descent step?

PHILLIP ISOLA: Exactly, yeah. So I didn't dig too much into this paper. I don't remember if it's purely an empirical argument that they showed this, or they actually proved the cases under which it works, but you can do either one. I think that if they proved something, it's for a linear case and something very simple. But there's a lot of follow-up papers to this, too.

AUDIENCE: I just remember this is very similar to a paper I read. And I think it's in some inverse reinforcement learning, when you give a human correct interventions, and you assume that those interventions are intentional, and you want to update your weights based on that intervention. They use a similar method. And they don't take the gradient. They just do a difference between trajectories and the feature counts. But I was wondering if-- is this the same idea or--

PHILLIP ISOLA: Yeah. OK, so the question is, in inverse reinforcement learning, there might be similar methods where you have implicitly defined functions defined by the reward. I need to think more and parse that question a bit more, but I would say, yes, I think there should be connections to that.

OK. So basically, this raises this very tantalizing question. Well, if in-context learning is somehow related to or even, in some cases, equivalent to gradient descent over the context, why not just do gradient descent over the context? That might have some advantages. So what are the advantages it might have? Well, one is that in-context learning is learned learning there's no guarantees on its behavior. It's just going to behave in a way that depends on the data distribution it was trained on.

So in-context learning might not converge. It doesn't have these properties like gradient descent has. Gradient descent, if I take small enough step sizes, will always find a local minimum. In-context learning-- there's no guarantee that if I show it more and more context, it will get better and better, or it will find a local minimum at least. So it's less well-understood. It's something that is learned and we don't have as much control over. We only control it via the data and the learning process. So if I can replace in-context learning with gradient descent, then I might have an object that has better guarantees, that I can understand better.

Additionally, gradient descent can leverage more test-time compute than in-context learning because in-context learning is just feed-forward pass through the transformer. But gradient dissent can be an iterative optimization algorithm that does forward and backwards. That's more compute, and it can do it many, many, many times.

So one thing you want to look out for is how can you leverage compute? The more compute you can leverage-- oftentimes, there will be some scaling law with that. So gradient descent has this possibility of leveraging more compute at test time. But there's possible disadvantages. Gradient descent is not the smartest learning algorithm. In-context learning could learn something that takes better updates, better steps than gradient descent. So it could be better than gradient descent.

But the papers, the theory papers suggest that maybe it's not that much different than gradient descent, but that's still only for simple toy models. Maybe in huge language models, in-context learning is actually smarter than gradient descent. That could be a possibility. Yeah.

AUDIENCE: You say doing gradient descent over the context. Are you suggesting retraining the model on input sequences, where you have this context included, or are you talking about-- there's prompt optimization papers where you have some prefix tokens, and then you do gradient descent to optimize the prefix tokens for your downstream?

PHILLIP ISOLA: So the question is, are you doing gradient descent on f, x, comma c or f, x, I think, essentially? I mean, there's a few ways you could do it, but in this case, I'm suggesting gradient descent on queries without context and trying to use the context examples just to improve the mapping from the query directly to the answer without the context. But the other ideas are good, too. Yeah. Question.

AUDIENCE: Oh, yeah, I was thinking intuitively about this other assertion of gradient descent over in context learning. And is it not fair to say that there's a limit to how much you could train the model, just like an expert can do a PhD in multiple fields? If you do it in one and you gain the tools to develop logical reasoning and capabilities, then inference time might be the more adequate place to do learning at.

PHILLIP ISOLA: Yeah, so I think the question is, if you do learning at training time, then you have to be a generalist. And maybe there's an infinite number of scenarios you might come across, and how can you learn to do well in all of them? But if you do training at test time, when you have a context of very related examples you're given, then you, on the fly, learn a specialized skill, and you don't have to have memorized the infinite number of things you need to do. And I think that's exactly why people like in-context learning and test-time compute. That's a good point.

OK, so anyway, the name for this is test-time training, or that's one of the names that's emerged for this thing. And the idea, just roughly, is you're going to update the weights of your model as a function of the test query, or maybe the entire test set. You might have a sequence of test queries. The test query could include these in-context examples. And you update on the basis of those. You train on those.

But there's also another, more extreme scenario where the test query is just an x. You don't get supervised in-context examples. So can you still do test-time training when I just get a single photo at test time, or maybe a movie at test time, or maybe a book at test time? It turns out you can. I just have an image. I don't have an image and labels. I just have an image. Can I improve my performance at predicting the label of an image?

So it's not really in-context supervised learning anymore, but you can do self-supervised test-time training. And this paper from Sun and folks did that, and it's really cool. So what they did is they said, I have a system that's going to try to label a photo. Here's x, and it's going to try to predict the label, bird. And I can decompose that system into a first trunk of the network and then a task-specific head.

And it turns out that trunk of the network I can train with a self-supervised head as well. So I'm going to update the weights of f in order to make them do better at solving a self-supervised task like masked autoencoding. So this is an MA, a masked autoencoder. I remove some patches, and I try to predict the missing patches. I will do that to learn-- to fine-tune f. I might have a pre-trained f, and now I'm going to try to fine tune it on the test example with a self-supervised loss. And then I will simply have a better representation. And this can actually improve your performance on correctly identifying that this is a bird.

So test-time training can be done with an unsupervised objective at test time. And this is a very general-purpose thing you can do. You're just adapting to your test scenario. It's exactly like the point that the student over here was making, that you can't pre-train for everything. So let's say you go to Mars. You want to, on the fly, adapt your representation to be more sensitive to red or, somehow, the distribution of images looks different and you don't have to have labels. You can just adapt your representation and get better performance. Yeah.

AUDIENCE: So here, you're updating the weights to f or g task?

PHILLIP ISOLA: You're updating the weights to f only. And then g task-- I believe g task stays frozen, but maybe you have to do some stitching operation.

AUDIENCE: How do you ensure you don't forget?

PHILLIP ISOLA: Yeah. I'd have to look into the paper. I don't remember the details. But I think it's just if you don't change f that much, you just perturb it slightly, there's a regime in which it works. If you had trained f with a really big learning rate and changed the weights entirely, then no guarantees. But they show cases in which it works.

So test-time training is really cool. And here's a paper from about a week or two ago, which is a really exciting new result. This is from folks at MIT, so maybe some of them. Jacob Andreas and Yoon Kim are the professors, and then the students. And what they showed is that there's this challenge called the arc challenge, which is supposed to be a really hard challenge for deep learning to solve.

And the arc challenge is cases like this, that you get in-context examples-- you get a few examples of input and output, and humans can look at this last one, and you can answer what the pattern is. It's not trivial, but if you think of it, you can get 50% or maybe higher on this benchmark. So expert humans can get 80%, 90%.

And this challenge is kind of meant to fool the deep nets of a few years ago. So the deep nets of a few years ago didn't really do any search or test-time optimization. They just learned a general-purpose rule that would map from data in the training set to correct answers in the training set. And so people thought, well, we better-- to solve problems like this, we need to do symbolic reasoning. We need to do search. We need to add all these fancy methods.

And what they showed in this paper is you can get pretty far by only doing this test-time training method. So you simply do the regular test-time training with supervised examples, not self-supervised. You pre-train a model to be generally good at this type of pattern recognition problem. And then you-- given the few in-context examples, you update the model to do better at those in-context examples, gradient descent on these supervised examples. And then you apply it to your test query, and you get the right answer.

And if, instead, you do-- fine-tuning is a training time optimization, so fine tuning is not on the in-context examples, but it's on the training set of similar problems. Then it doesn't work as well because the in-context examples are the most relevant. This is exactly about your special-- you can't learn to do everything at train time, so instead, you wait until test time. You see the most relevant examples, and then you fit to those examples, but you just fit to them with gradient descent.

So I thought this is really interesting because this arc challenge says, AGI has stalled. New ideas are needed. But this paper is kind of saying, wait a second. Maybe a little bit of a new idea is needed, but actually, they got the highest performance that's been reported so far, and they did it with old methods. They did it with gradient descent supervised learning.

Now, it happens to be on the in-context examples. That's a new idea. And they have some other tricks in the paper. But otherwise, it's fairly old methods. And it's interesting. It raises the question, to me, that, do we really need fancy new symbolic reasoning methods? Or maybe gradient descent is still the workhorse, and is still-- even for inference time, gradient descent is one of the best things that you can do? Yeah.

AUDIENCE: Sorry. I'm maybe just a little confused on what they did. So did they just feed in the two examples, like and then the next one and say, what should this be?

PHILLIP ISOLA: Yeah, no. So it's not like it's a transformer that takes these in context. That doesn't work as well. This is the training set for a new phase of learning with gradient descent. So they're simply feeding in this single image, looking at the output, penalizing the difference from this single image, backpropping the loss to update the parameters of that mapping, and then doing it on this single image and doing that some number of times. Make sense? They're just doing supervised learning. There's no transformer that's processing a context in this case.

AUDIENCE: Yeah. Isn't that, I guess, in a sense, cheating, like, looking at your test loss?

PHILLIP ISOLA: Well, that's not cheating according to this challenge. The name of the challenge is given a few examples of a new pattern, apply that pattern to a query, a test query. So they don't have the answer to the test query. For a human, this is exactly how the challenge is stated. Here is example behavior. What is the correct behavior on this new query? And they don't get to see the answer on this new query. They only got to see the answer on the examples. So this is the definition of the problem. It's not cheating. OK. Cool.

But humans can do 80%, 90%. Maybe François Chollet, who made it, can probably do 100% on this. So 62%-- maybe we're not quite there yet, but that's interesting that you can go pretty far.

OK, so for the final few minutes, I want to talk about this really exciting, current way of doing things, which is to use search and inference and optimization at test time to come up with better solutions, and then distill those solutions back into the pre-training the base model itself. So fit to those optimized solutions and then repeat. So this is how you can use search to give better examples to train on.

So it's really a simple idea. It's an old idea. It's been around since the beginning of AI, I guess, in some way or another. It is, OK, come up with multiple answers. Now we have some way of doing Best-of-N or tree search or some kind of optimization to find better answers than the LLM output by default. And now I can simply take that better answer and call it a training pair for supervised learning to fit the LLM to directly output that better answer.

So I'm just improving my performance by my own model search, plus the search algorithm. Sometimes people call these self-improvement or recursive self-improvement. It's just a model that is initially crummy. It does some optimization or verification process to come up with better solutions. And then it trains on those better solutions to be a better model.

So you might be wondering, well, wait a second. The information came in through the verifier here. And that is-- that's one of the powerful ways that a model can improve. And you might not want to call that, self-improve. That's improving with an external verifier. Sometimes people mix the terms and might still call that self-improve. But you could also have this verifier be a model itself.

Remember, it's easier to learn to verify than to generate. So I can learn to verify easily and quickly and then use the verifier to improve the generator, and then it's purely self-improvement.

So you can also do it the other way around. So you can use search to improve learning and use learning to improve search. And this is basically the fundamental concept in reinforcement learning, which we didn't really have time to cover. But I'll just tell you the basic loop in reinforcement learning. Let's say you're playing the video game, *Mario.* So you have a few actions you can make. You want to jump over that-- whatever that thing is, that-- yeah, that tunnel thing-- that pipe. He wants to jump over, that pipe.

So you're going to search for random action sequences until you find one that will get a high score in the game. So jumping will turn out to get a high score. Then you're going to take that-- you searched. You found the good sequence, the good sentence. Here's the good sequence of actions. And now you're going to fit a neural network to directly predict that sequence.

So now you're going to say, in that situation, my search algorithm found that it was good to jump, so I'll train a neural network to directly go from that-- I don't have to do a search process anymore. Now I can just directly map. It's called a policy that directly maps from the state to the optimal action. And then I'll repeat the process. Now I'm going to repeat, and I'm going to guide the search to be preferentially doing things that the neural network thought were good. And so I'm not going to search over random sequences, but sequences that the model-- the policy already thinks are good sequences.

And then that iteration just keeps on improving. And if you have some score in the game, like a reward function in reinforcement learning, then this will provably get to maximal reward under some conditions. So that's the name of the game in reinforcement learning. Yep

AUDIENCE: I had a quick question. These examples that are reinforcement learning assume something that might seem simple, but that might not be general, or at least I don't see it as general, which is this ability to engage in self-play, which you have by virtue of being in the game environment, but it might be hard to do. So how are some ways that we can translate that to more general examples of--

PHILLIP ISOLA: Yeah. So this has interaction in a world. There's a reward function the world is giving you. How can we do this for something like just improving reasoning in a language model? I'll show you in a second. But I'll just say that, yeah, there's a lot-- this is-- just all over reinforcement learning, this is the thing you do. This is how AlphaGo works. You do some tree search over the possible moves you can make. You find the best move according to the search.

And then you back up that solution into a heuristic, which is able to predict-- without expanding the tree, it's able to predict, if I were to expand the tree, I would have found this solution, a good solution. So therefore, that's a good node to be at. I won't go into the details of this, but it's a search process that then gets distilled back into a learned heuristic.

Another one that does this-- and now we're getting to how this can just improve general-purpose language models and reasoning-- is called STaR. So what STaR does is you start with a language model. You ask a question. It's prompted with a kind of chain of thought. So it says, tell me your steps of thinking-- they call that a rationale-- and then produce an answer.

And it says, if I got the answer wrong, which I can check. They assume they have some way of checking if the answer is right or wrong, so they have a verifier here. If I get the answer wrong, then I will-- I'll do something. I won't talk about the bottom branch. But if I get the answer right, then I will reinforce that behavior. I will take that answer and that rationale and say, that was a good answer and rationale, and now let me train my language model to preferentially produce these good rationales, these good chains of thought that lead to the good answer. If they get it wrong, they also have this hint operation that tries to give them some signal that maybe that's not critical.

So this is what a lot of people seem to think that this O1 model does. So I'll leave-- I'll end the talk on this kind of speculation of because this is the latest-- OpenAI thing. But I think that we have a pretty good-- Sasha Rush gives this as one of his candidate explanations for O1, but I feel like it's got to be something basically like this.

So one is you search for chains of thoughts that solve the problem, but you're going to randomly sample different chains of thoughts. So some chain of thought sample-- some-- it's like beam search. I'm looking over all the possible different ways the language model could start reasoning and thinking aloud until it comes up with the answer. And now I'm going to find the ones, according to some scoring function, that actually got the right answer. And then I'll fine-tune my language model on-- to directly output those good chains of thought.

OK, how will I verify if I got the right answer? It could be a learned verifier. That's another language model. Or it could be some kind of humans on the internet that are saying, this is correct or that is incorrect, or it could be a compiler or a proof checker that actually symbolically checks if you're correct or incorrect. We don't know.

And then I just repeat. So now the LLM gets better at outputting better chains of thought. And that makes my search converge on a narrower space of really good chains of thoughts. But I still-- I can still sample different possibilities in that space, see which ones are the best and repeat the process. It's just reinforcement learning applied to reasoning with a language model.

And at test time, at deployment time, you can just use your fine-tuned LLM, or you can use your LLM plus additional search on top of it. And that's why if you run O1 or any of the other models that do this type of search, they will take longer because they're doing extra search on top of your solution. That also improves things.

AUDIENCE: That feels like more rounds of searching, so how could this be more efficient as they claim?

PHILLIP ISOLA: This feels like more round-- has it more efficient? So they're not-- I think they're not claiming it's more efficient. They're claiming it achieves higher accuracy. So by thinking more, by doing a lot more compute on every single query to come up with better and better answers, you can distill that back into your base model. That base model will be just as efficient. But if you use that base model in conjunction with additional search on top, you can always do as much search as you want. You could do infinite search on top. Under some conditions, if you do Best-of-N with a very uniform prior and you do infinite search, then you'll be the best of all worlds. And so this is just a way of using more compute to get better answers. Yep.

AUDIENCE: On the previous slide, it just assumed a new training instead of question rationale and answer. Is there any idea on the scale of that training set that's needed at this point?

PHILLIP ISOLA: I think that this might have some supervised rationales or some seed, but then it can generate its own-- the language model will output a rationale, and the good rationales that led to the correct answer get added to the training set. So it generates its own rationales.

AUDIENCE: Do you know what they started at, though, to--

PHILLIP ISOLA: I don't remember, but I think they only had a handful or some human-written rationales. Yeah. OK, cool. So that's the latest, greatest thing in AI. We don't know exactly how it works, but it's very much a cool frontier. So I will end there.