[SQUEAKING]

[RUSTLING]

[CLICKING]

PHILLIP ISOLA: So today, we're going to talk about scaling laws. And this lecture, we're at the final few weeks of this semester so we're moving into territory which is much more ongoing science. I'm going to talk about recent papers and results that are empirical in nature for which we don't know if these results will really hold up, stand the test of time. But this is the latest research on deep learning. And a lot of the next few lectures will be of that flavor.

So what are scaling laws? Why do we care about scaling in deep learning? And the reason is that deep learning is something that works really well once you have a lot of scale, once you have a lot of data, a lot of compute, a lot of flops, a lot of parameters, big models on big data with large machines. That's kind of the regime in which deep learning is winning and doing very well.

And a big part of the story of deep learning is how do you scale things to make efficient use of the resources that you have. And a remarkable thing that's been observed is that, sometimes, the way that models behave as they scale is quite predictable. And so you can define these scaling laws, which are predictions of if I double my compute, how will the model behave? If I double my model size, how will the model behave?

And if we can identify those types of laws, then this will be really useful for planning our resources and our extrapolations into the future. We can predict what next year will bring. How competent will be the models of next year? Well, if have a law of how does the performance change as a function of data points, then we can know how well the models will perform next year. And that's very useful for us to plan.

So here is one example of a scaling law that people have noticed. This is from an OpenAI blog. And on the x-axis, they're plotting year. And on the y-axis, they're plotting the scale of the compute that's used to train each model. So this is saying that as years go on, the amount of compute that we use to train models is increasing as some kind of exponential, or it may, in fact, be a power law in this case.

But basically, for every additional year that we go on, we get a kind of big increase in the amount of compute that we use in these models. So this should be reminiscent of things like Moore's law. So Moore's law said that we double the number of transistors in our machines every two years or however much it was.

And this scaling law says that every year-- actually, every three months, I think, you double the amount of compute that you use to train the models. And I think this law has actually been increasing. The slope is actually increasing a little bit, so it's not quite even three months now that it takes to double the amount of compute, it might be even less. See, this ends in 2018.

So deep learning, predictably, is using more and more scale as time goes on. And now, we want to understand how that relates to the model performance and at what scale will we achieve what types of abilities. So another motivation for studying these questions comes from this really influential essay called "The Bitter Lesson" from Richard Sutton.

So raise your hand if you've read this. It's a one-pager. So you should all take a look at it. It caused quite a splash in the field when it came out in 2019. A lot of people really didn't like it. A lot of people loved it.

But I think there's a lot of truth to it. And it's an important lesson for us to understand and take to heart, despite some nuances and caveats that you can add to it.

So what Rich Sutton was saying is that-- he's an AI researcher. He's been working in the field for a long time. And what he observed is that all of our clever ideas and algorithms, most of them don't really pan out. And the only thing that systematically pans out is designing simple algorithms that scale well.

And so he's basically saying that you need to think about how models scale as a function of resources in order to identify the winning models because, as time goes on, whatever the ranking, whatever model is doing better or worse than any other model, it doesn't really matter. What matters is how they'll scale because, eventually, as time goes on, the one that scales better will win.

So it was a little bit of a critique of all of the clever algorithms in the field. No, it turns out that the really simple algorithms turn out to scale better, and those are taking over. So interesting essay, but it has really led us to take scale quite seriously.

Of course, scaling laws and thinking about how algorithms scale as a function of resources is not at all new. It goes back to the foundations of computer science. So in Intro CS or in Algorithms, you learn about asymptotic analysis, big O notation. That's a scaling law.

That's saying how does my time complexity of an algorithm scale as a function of the input complexity. Like if I have a graph, I'm trying to find shortest path as a function of the number of nodes in the graph, how does the model's behavior, how does the algorithm's behavior scale? How long does it take to find the shortest path?

So this idea of asymptotic analysis, looking at as resources or complexity or as some parameter of the input scales up, how does some property of the model scale, that's a standard part of computer science. And it's very much something that you want in order to make predictions about the future because things are scaling.

That's kind of the bottom line of this graph here is that things are scaling. Eventually, they might saturate. We might not have any more resources. But we're not really anywhere near that point, so we need to be able to predict how things will scale.

So one of the fundamental questions of the study of scaling laws is if I have a budget of compute or a budget of hardware, how should I optimally allocate that budget? So, let's see. What model should I train on, how much data, in order to make the best use of my budget?

So here is a scaling law, another scaling law, from OpenAI that they use to decide on how to train the GPT-4 model. So now, we have on the x-axis is the compute, and on the y-axis is the performance of the model. So performance is measured in bits per word. So this is something related to the cross-entropy of the next-word prediction. So lower means that I'm making more accurate predictions.

And as I train my model with more compute, so more flops, more steps of SGD, more gradients, more updates, so compute is now measured in just flops on the hardware. And that is related in a complicated way to the number of tokens that I observe, the depth of the transformer. All these things affect the compute, but we're just measuring it all in flops, like how many floating point operations do I do on my hardware?

So as I use more compute, of course, I will get better models with lower loss. And so 1 here is like a petaflop, or I'm not entirely sure what the axis is, but this is a lot, a lot, of compute. And if you read this paper and saw that training GPT-4 cost $10 million or however much it costs to do one training, like a petaflop of compute was $1 million, let's say, then you might have thought, well, obviously, they did a hyperparameter sweep, and they tried a lot of model variations.

If it took $1 million to train the final model, it must have taken $100 million to design it, right? But that's not actually the case. I don't know how much it took to design it or run the final model, but they did this trick, which is they don't design it by sweeping hyperparameters and testing models at this scale.

They design it at 1/1,000 the scale, or maybe in this case, it's 10,000, 1/10,000 the scale. So they do all of their design and hyperparameter sweeps and ablations and model comparisons at small scale. They do it at multiple small scales, and they find this curve that will predict performance as a function of compute, and then extrapolate that curve to the scale that they'll train their final model at.

And if they can predict that the final model will get such and such a score, they will choose the model that will get the best score by running these curves for a bunch of different variations. Then they'll choose the model that they predict will have the lowest score at the skill that they actually care about. And then they only train once at that massive scale.

So that's the strategy that a lot of companies are using now. Yeah, question in the back.

AUDIENCE: My question is, is compute proportional to time? Like, let's take into account the amount of time? Or is it something that's time-normalized?

PHILLIP ISOLA: Yeah. That's a great question. Is compute proportional to time? So here, you're just kind of flattening all uses of compute into just flops. So flops could be allocated sequentially or in parallel. So compute does not have a direct relationship to time.

So twice the compute can be twice the time, or it could be twice the number of GPUs in parallel. So this is just measuring compute in terms of flops, which can factor into different mappings onto time. Yeah.

AUDIENCE: Is this graph for a constant amount of data?

PHILLIP ISOLA: So I'll get to that. Is this graph for a constant amount of data? I don't know for this graph in particular, but usually, for scaling laws, we will try to say we're going to predict the performance at a given compute level for the optimal amount of data for that compute level. We'll optimize it for all other factors when we make one of these graphs, but I'll show you that in more detail.

And these types of scaling laws, they don't only appear for the loss. So bits per word is closely related to the loss function of next-character prediction. It's like the amount of information you-- the amount of uncertainty you have over the next word, essentially, measured in bits.

But what if we try to predict performance in terms of something more meaningful, like how many problems you pass on some math test or some benchmark? And it turns out that there are also fairly predictable trends in scaling laws for the downstream performance, not just for the loss. So a lot of scaling, a lot of research, is on predicting the loss on the training set as a function of resources.

But you can also make scaling laws for test performance or for downstream performance, in other senses, according to other measures. So this was capability on 23 coding problems. And again, you have this nice curve. And what I'll tell you is that these curves turn out to be power laws with respect to the underlying resource.

So scaling laws are things that try to predict some parameter of interest, so, oftentimes, the test loss as a function of the amount of data you train on, the amount of compute you use, the number of parameters in your model, these critical resources that you care about. And what you're going to try to do is select these resources-- these will inform you on how to select these resources so as to get the lowest possible test loss. So that's why we care about scaling laws because they tell us what should the batch size be that will achieve the lowest cost when I scale things way up, the lowest loss when I scale things way up.

So for a given architecture, how much data, what model, would be needed to get a target performance? If I know I need to be like 99% accurate on my self-driving car assessment, then can I predict at what scale will I have that? And that would be useful for regulatory bodies. I want to tell them in 2025, we'll have 99% performance, and you can start planning around that.

So it's good for making predictions about the future and planning ahead. It's also good for choosing parameters that maximize performance at a given scale. So generally, the kind of scaling laws we'll look at are going to be of the form as a function of parameters. And we'll look at number of-- as a function of resources.

So we'll look at N, the number of parameters, and D, the number of data points. So D is data points now. It's not depth. Just these terms are all overloaded. D could be a lot of things. But here, D is not dimensionality. It's not depth. It's data points.

So we're going to try to find what is the optimal number of parameters and what is the optimal number of data points for a fixed compute budget C. So C is going to be the flops that are required to train a model of size N on a data set of size D.

So roughly, to train a model with N parameters on a data set of size D, we'll have something like N times D amount of flops. It might be a little more complicated than that, but every additional data point, I need proportionally more flops, every additional parameter, I need proportionally more flops.

So for a fixed compute budget, if I want to minimize my performance-- sorry, I want to minimize my loss, so maximize performance. There we go. Then what should I set? How big should my model be and how many data points should I train on?

So we can train a really big model on a few data points for the same cost as a small model on a lot of data points. So what should I do? Should I train a huge trillion-parameter transformer on 10 data points? Or should I train a 1 billion-parameter transformer on a million data points? Those might have roughly the same cost in terms of flops.

So the paper that really introduced or popularized the idea of scaling laws for neural nets was this one from Jared Kaplan and Sam McCandlish and folks at OpenAI. This team subsequently peeled off and formed the company Anthropic, but a few of them stayed at OpenAI. So basically, the two big AGI players, or two of the big AGI players, is a schism of this team.

And so the scaling laws are a huge part of the philosophy behind those two companies and also part of their success, I would say. So they studied scaling laws in 2020 or so, or maybe this is like 2018, for autoregressive transformer models, so language models with a transformer. And they tried to solve the equation I had on the last slide. How big should the model be, and how many data points should we train it on if we have this compute budget?

Maybe they have a billion dollars' worth of flops that they can allocate, and they want to how should we spend that billion dollars. It's measured in flops, not dollars. So again, there's a lot of different transformations you'd want to apply in order to really ground this out in raw economics. But flops is the one that this paper looked at.

So they trained on some data sets of webtext. This is just next-character prediction. And they varied a lot of different properties of the model, like the model size, how many parameters does it have, the data set size, so D, how many tokens are you training it on. And they also tried other things like the batch size-- we'll come back to that at the end of this lecture-- the shape of the network, what's the aspect ratio, like the depth compared to the width of the network.

So they looked at all of these factors. It's a really rich paper. So we assign this as a reading because it's just a beautiful piece of science and full of insight. A lot of the findings have been modified since-- they found their empirical fits to the data, and there's better empirical fits in recent years. But it's a good lesson in how to do this kind of work, and there's still a lot of great insights in the paper.

So here's how they computed. Here's how they estimated compute. So I said that compute should be something like number of parameters times number of data points. So they used that as their rough approximation to compute. Again, this is just an approximation to how GPUs actually run because it's going to depend on things like what's the precision of-- is it a 16-bit number?

These things, there's a lot of details that matter. But basically, compute equals number of parameters times number of data points observed. So number of data points is batch size times steps of gradient descent. So that's batch size times steps of gradient descent.

And then times parameters, that's like how many flops. And then for some reason, you multiply by 6. I don't quite know where that comes from. But that's the constant that turns the units into flops. So they ran these for petaflop days. They ran this on a lot of compute.

So the main finding of this paper is that there is a power law relationship between the test loss and the resources, either the number of parameters or the number of data points. So resource X, again, that could be data or parameters or even compute in test loss are related by this type of power law, which is a 1 over the resource value to the alpha exponent. So alpha is a constant that will depend on other properties, properties of the data, the world, the task, the architecture.

But for a particular architecture or particular task like language modeling with a transformer, then we'll have some alpha that can be estimated, and that's the parameter of the power law. So what does a power law say? It basically says if I double my resources, I double my number of data points, my loss should go down by-- if I double, my loss should go down by 2 to the power.

Alpha is the parameter of the power law that describes its shape. If I quadruple my data points, my loss should go down by 4 to the negative alpha. So let's say that alpha is 1, just for simplicity. Then it's like if I double my number of data points I train on, then my loss will go down by half. So 2 to the negative 1 will be my loss goes down by half.

If I quadruple, my loss goes down by 4. So my loss becomes a fourth. So that's the idea of a power law. If I scale my resource, I will scale my loss. So if alpha is 2, then if I double my resource, then I'll get 2 to the negative 2. So my loss will go down by one fourth of its previous value.

So here's the empirical fit that they found. So on the y-axis is going to be the test loss. And on the x-axis is going to be the number of tokens that they train on. So that's the data set size.

Remember, this is autoregressive transformer, so tokens is we tokenize the words, and we're just predicting next token. So this is how many tokens we predict. One important point here is that we're not taking into account multi-epoch training. So remember, I could have a data set of some size but revisit data points multiple times.

So what does that count on? How do I measure the number of tokens? This is all one epoch training, so it's very easy. It's just I visit each data point exactly once. And so the number of tokens that are processed is exactly equal to the number of tokens in my data set.

So this is a log-log plot. So a power law on a log-log plot looks like a line with a slope, which is equal to the exponent of the power law. So there's a great fit. That was the first main finding, that, empirically, there exists a power law relationship between data set size and test loss.

So what are the equations down here? So this is the test loss as a function of the data set size. We're going to have data set size. It'll be 1 over data set size to the alpha. And alpha here, it turns out, empirically, to be about 0.1.

And they also have this Dc, which is some constant. This is another parameter of the power law. That just is going to change the offset of this line. So think about take a log-log plot is like take a log of this side and a log of that side, that's what we're plotting, one against the other.

If we take a log of this side, then the exponent just becomes we multiply-- it's like the slope of log of what's on the inside and log of this over that is log of this minus that. So this is just a constant offset. That's like an offset on the y-axis. This is the y-intercept. Yeah, question.

AUDIENCE: Is the form of the loss function important? Should I understand the loss function itself? It doesn't matter too much?

PHILLIP ISOLA: The form of the loss function, is it important? It is. And I think people have observed power laws with respect to both cross-entropy as well as mean squared error, different loss functions will result in different power laws.

But they can be explained by power laws. In this case, I think we're looking at cross-entropy type loss. So next-word prediction, like cross-entropy. Yeah?

AUDIENCE: So there was [INAUDIBLE] of the maximum, essentially, attention length or the number of tokens [INAUDIBLE]. How does that relate to the [INAUDIBLE] here?

PHILLIP ISOLA: So how does the attention length relate to this plot? That's not really showing up here. I think they're choosing some fixed attention, context length. You could also derive scaling laws with respect to context length. I believe people have looked at that, too.

So as I scale my context, how does performance change? Can I predictably say that if I want performance to be 99% accurate, what's the context I need? There probably are scaling laws like this.

I can't think of any off the top of my head, but that's not what they're looking at here. So I believe they're either choosing a fixed context length for all the models they're comparing, or maybe in some subsection of the paper, they try different context lengths. I don't remember.

So all of these things are simplifications. They're saying let's pick one setting, this transformer with this context length with cross-entropy loss, and let's measure how it scales with respect to one interesting parameter. And will that hold for an RNN with an MSE loss? Maybe, maybe not.

So you should always keep that in the back of your mind. This is characterizing one popular family of models. But there are other models outside that family, and we don't yet know how well these laws will extrapolate. Let's go here.

AUDIENCE: Is there any empirical interpretation of the value of DC? Does it represent actually something meaningful?

PHILLIP ISOLA: Yeah. Does DC represent something meaningful? So I would say we have an equation with two free parameters, and we're trying to explain points with two free parameters.

And we say that a good explanation of a set of points, it turns out to be a slope and an offset on a log-log plot. And what's the interpretation of that offset? I'm not sure if I have a crisp explanation. Yeah.

AUDIENCE: Yeah. I was just wondering if there's a discrepancy between the formula at the top of the plot and then the order it is at the bottom of the--

PHILLIP ISOLA: Oh, I see. Interesting. So do we flip-- this is D over the offset, and this is offset over D. But there's a negative here. Yeah, so I think it's consistent. Yeah.

AUDIENCE: Maybe we'll get to this later, but I'm just curious. Do you have some intuition for why it would be a law-- or, so sorry, a power law, say, an exponential decay or some other?

PHILLIP ISOLA: Yeah, I'll get to a theoretical model for why this would emerge. But I would say I'm not sure how much I trust the theory behind this. I think that's still ongoing. I would say it's more just, empirically, a power law fits the data better than an exponential.

AUDIENCE: Do you feel like this is probably more likely due to the data or the architecture or-- I guess it's not so clear.

PHILLIP ISOLA: So in the theory that we'll get to, it comes out of just high-dimensional statistics. If I'm trying to fit a function in high-dimensional space, then there's a dependence on the dimensionality, and that will turn into the alpha.

So I think it's something that's more related to just high-dimensional statistics than it is to data distribution. It's not specific. These power laws show up for all kinds of data distributions. Yeah. Power law relationship.

So I said that that's true for the data set size. What about for other things, like as I scale the number of parameters in my model, the weights in my model, how does test loss behave? And it turns out that's also a power law. So empirically, the dots are empirical measurements, and the line is at best fit.

So you can say this sounds like some physical law that we're driving from first principles. It's not, really. It's much more empirical. It's like science.

We observe. We want to understand the relationship between a number of parameters and test loss, so we just start plotting the points. That could have come out as some parabola or something crazy, but it turned out to come out as a line on a log-log plot.

So that's what led people to say, oh, that's well explained by a power law. Why? This is more of an open question. We'll talk about some reasons. Yeah.

AUDIENCE: Just one question. Based on the two expressions, does it mean that it's better to increase the data set to lower than loss rather than--

PHILLIP ISOLA: Yeah. So is it better to increase data set size or number of parameters? What can you infer? Think about it. We're going to show the answer in a minute.

And same for compute. So now, we have three power laws for if I scale the compute, test loss behaves as a power law with respect to compute. If I scale the data set size, same thing.

But notice that each of these power laws has a different constant offset and a different slope. So that will have implications about how to optimally allocate our resources. We want to go down on the y-axis. We want to get lower loss.

And so we can get lower loss by increasing data set size or number of parameters. And if those have equal cost, then we would prefer to-- we'd prefer, in this case, to increase our data set size. I might be saying that wrong.

Let me wait until I get to the slide so I make sure to say it correctly. So in this plot, I'm pointing out these blue curves. So what are these blue curves? These blue curves are, as I scale my compute, it turns out I also need to change the amount of data that I train on, and I need to change the number of parameters of the model.

So it turns out these power laws, they show up as the Pareto front, the frontier of all of the different curves, for different specific models of different specific sizes trained on different amounts of data. So it's like if I optimally use an infinitely large data set with an infinitely large model, or I pick just the perfect model size that will minimize the test loss, then the frontier that I arrive at is a power law with respect to compute.

And I think it's the same for these other factors as well. So if you are starting to look for a power law, and you take one architecture, and you just scale the compute, you won't see a power law. You'll see one of these blue curves.

What you need to do is you need to tune the data set size and parameters for every level of compute. And if you tune them optimally, then the frontier you'll arrive at is the power law.

AUDIENCE: Sorry. What is one blue line across [INAUDIBLE]?

PHILLIP ISOLA: So one blue line is I take one data set, a fixed size, one model, a fixed number of parameters, and I train it with different amounts of compute. And I plot out what the loss is.

And you can see that they start to maybe even overfit a little. It's like as I add more compute, I'm no longer decreasing the loss. Maybe even it goes up a little bit. There's not very much evidence of overfitting here. It kind of plateaus.

But for a bigger model, I need to scale up the amount of data I train it on proportionally. And then if I scale it up, then I can use that more compute to get a lower loss. That's the point. Cool.

So, yeah, finding one is there's a power law relationship between each factor and performance when all other factors are set optimally. Finding number two is that you can actually understand the relationship between parameter count, number of flops, and data set size that will achieve the best performance.

So this is really the point that I was just making. But maybe it will be a little bit clearer on this slide. So on the y-axis now, we have the loss. And on the x-axis, we have the token. So same thing as before.

But now, each of these curves is going to be a different model size. So what you're seeing is you get power law behavior if the model is sufficiently large. But if the model is not very large, then adding more tokens to your data set doesn't improve performance. So you have to set your model to be large enough that it can fit all those tokens.

And if it's a small model, if it's like a linear model, it can only find a linear fit. It just doesn't have enough capacity. So you have to scale your data set size in tandem with-- or you have to scale your model size in tandem with your data set size in order to observe the power law and get your best possible test loss.

So we can actually now model L, the test loss, as a function of both, and the number of parameters, the model size, and D, the data set size. And now, we have a more complicated functional fit. Again, this is just we have empirical measurements, and we're trying to find a simple function that fits them.

And it turns out it's like this power law of both factors with some nested exponents. So is this something fundamental? Does theory predict this? Not in this paper. This is just a simple equation they found that explains the data pretty well.

But it's like a power law in terms of we have these constant offsets. And then it's a power law with respect to the data set size and with respect to the model size. But there's an interaction between them.

So empirically, they made some measurements. Other people make other measurements. You don't need to worry about what those numbers mean exactly. But this is just the fit they found to the data.

There might be a little bit of reading of tea leaves here. One of the famous quotes from some physicist, maybe John Von Neumann or somebody like this, was that with five parameters, I can fit an elephant. Any physicist know who said that? Anyone? Does anyone know? Yeah?

AUDIENCE: I heard it attributed to Anderson.

PHILLIP ISOLA: Anderson? OK. So, yeah, I just attribute everything in physics to Neumann or Einstein, but it's probably wrong. It's probably Anderson. But yeah, with five parameters, you can fit an elephant.

An elephant is a complicated shape, but with five free parameters, you can make really complicated shapes. So here's, what? 1, 2. N and D are not free parameters. They're constants-- NC, DC, alpha-N, there's five, right? So I can fit an elephant.

So maybe the reading of tea leaves with this functional form. Maybe there's other functional forms that could do just as well. So we can do the same type of analysis now with respect to not model size and data set size, but model size and number of steps of gradient descent.

Remember, these are our three fundamental resources, model size, data set size, and compute, like number of steps that we take. Well, number of steps will be related to compute. It'll be like number of steps times the batch size is compute.

So they found this even simpler functional form for explaining the relationship between the test loss and these two parameters. So now, it just becomes the sum of two power laws. And that's plotted on the left here, where we have the amount of compute that you need versus the test loss.

And you get kind of a power law, but if you increase the number of parameters of your model, then you'll get a lower loss. And again, to reach that frontier, you need to increase the number of parameters. And if you increase the parameters sufficiently, then you'll reach a frontier that looks like a power law, just in terms of the number of steps of gradient descent.

Yeah. Some constants in that equation, too. So here are some of the implications that they derived. So they fit all these relationships, and then they got these exponents and these constant offsets. And they said, well, based on those exponents, what can we say about how large should we make our model, what batch size should we make the model have, et cetera?

And they said that the number of-- let's say that you have compute resource C, then the number of parameters that you'll want should be proportional to C to the 0.7. And the number of data points that you'll want to train on should be proportional to C to the 0.27.

Take these numbers with a grain of salt because they change with different models and different eras. But they tell us some recipe for in order to get the minimal loss, then it will be better, in this case, to increase my-- well, I should set my parameter count to be a higher proportion of my total compute count compared to my data set size.

So the implication is, essentially, I should train relatively big models on relatively small data. So I should train a billion-size parameter model on a million-size data set. And that would be the best way to minimize my test loss according to this law.

So people have studied this in a lot of different contexts. And I just want to give a flavor of the other power laws that people have found. So it's not just about autoregressive transformers. These power laws have showed up in computer vision, in reinforcement learning, in a lot of areas.

Here's a paper on vision power laws. So they are fitting, I think, a vision transformer to do-- or a computer vision model. I think they might have tried a few architectures to do CIFAR-10 classification or some other type of classification problem. And here, they're plotting the power law in two dimensions.

So we have our loss on the y-axis. And we have our data set size and our model size on the two other axes. So this vertical axis is the loss. And there's kind of a power law in both dimensions, but they interact in a weird way.

So again, it's not something that you can easily wrap your mind around. But we do have a simple functional form, which is just the sum of these two power laws. So same as they found in language modeling.

One of the questions that you might have with this kind of reading tea leaves concern is, do these power laws actually make predictions that extrapolate across orders of magnitude? If I fit them on some subset of my data at some scale, do they make predictions that are valid on held-out data at larger scale?

And the answer is yes. So in this paper, they ran this experiment where they fit the model on these green points. So they have models of different size, but they did this out-of-distribution held-out test.

So they fit the model on some scale. And then they scaled up the model to a larger scale, or in this case, it might be a smaller scale, but you can do it in either direction. So you fit the model on these data points. And you ask, does it make valid predictions for models that are of a different scale? And it does.

So you can do cross-validation. We're not just reading tea leaves. That's really all that this says. So finding one, we have a power law relationship between resources and performance. Finding two, the--