Stefano Soatto: Thank you all for coming. It's a distinct honor and pleasure to be here to celebrate the achievements of Dr. Amari. I have been a fan of his work since early on. In fact, I've had a copy of his book on my shelf since I was a student. To be precise, I had a Xerox copy of his book on my shelves that I inherited from a friend who thought that it was worth spending half a day copying it. But I'll remind you that this happened before there was a thing called Portable Document Format or PDF, before the existed company where you could buy a book on the Internet and it would show up at your door, so that will date me. But nonetheless, his work has had a profound influence on my thinking and my work. There you have your celebrated natural gradient, and the beauty of the work is that establishes connections between the world of geometry, which is nice and elegant, and it has differentiable manifolds and vectors and parallel transports and remanotrics. To messy things at the time like measurable functions, probability distributions, and concepts that are quite squishy like entropy or if you have two random variables or mutual information, KL divergence, and of course, this object, the fisher information matrix that looks remarkably ugly when you view it written like this, but in fact, it's remarkably elegant when you think of it as the curvature of a smooth manifold. That was the beauty of the work that Dr. Amari unveiled to us. Today, we live in a world that is a lot messier, and often I find myself thinking that we're operating in a way which is not unlike that of Vesalius when they were trying to figure out how the brain works, and the way they did it is they opened one up and they messed around and poked and tried to figure it out. Not very successful. Modern neuroscience is slightly more sophisticated, but still we're planting electrodes and so on and so forth. What I would like to do today is I would like to give you a sense of how some of the ideas in the work on information geometry percolated down to evolving and forming theories that are emerging of how these artifacts work. It's a remarkable time because these are objects that we built, yet they exhibit behaviors that were not designed for that would not foresee. I think it's a unique time in life when I'm not aware of anything like this ever happened before, and it's an exciting time from my perspective. I understand many students are scared, but it's also a tremendous opportunity. The word information appears often here, and information is an abused word, especially when it comes to trained models to the objects of machine learning. I'd like to spend a few minutes getting into a little bit of detail to tell you how tricky some of these concepts are once you go down to the details and to shine a light on how the thinking of Dr. Amari actually helps us view these things. Bear with me with a little bit of technicalities here. Here's the different in network. That's the type of things that we work with today. If you use an AI bot these days, you would certainly say that it has a lot of information. You ask questions, you get answered. But what does that mean exactly? Whatever information is in that model before you started training is in the dataset that you use to train the model because there's nothing else before that. After you've done training it, it must be in the parameters or the weights of the train model because there's nothing else left. But when you try to use classical notions of information theory like Shannon Entropy there's no information in the data. It's instantiated, and the mutual information with the train model is also zero unless you view it in a continuum and then you have differential motor information, and then you will see that it's infinity. It's either zero infinity, neither of which makes sense relative to the intuitive notion. What about when you use these objects? When you use presumably in the future, you're going to sample some data which you don't have yet, so that is a random variable. It's not yet instantiated and then if you use the Tree model to select the hypotheses that score highest with that model, you can see that most of the time it works. Then we have theory that says that, yes, it works because so long as the train model optimizes a loss with some regularizers, it's not too far from the error you will make in the future. These are called uniform generalization bounds. They look like this, so it says that the error you will make in the future, which, of course, you cannot compute because you haven't seen the data yet, is not too far from the error you make in the past, which is entirely under your control. It's what you optimize during learning. You can make this zero, so long as the complexity of the model, which is measured by the information that the weights contain about the dataset is not too large. This is Occam's razor. It has the basis of regularization and so on and so forth. This object, the information that the weights contain about the dataset, you could make it zero by just dumping noise into your parameters. Of course, then you will

pay a price in the training error, or you can make the training error zero, but then the cost of storing the dataset into the weights will be very high. In general, you try to trade these off, and this is classical inductive generalization theory. What's the problem with this? What is this object? What is it when you try to compute it? You can write it as the Kullback-Leibler divergence between some type of prior distribution on the weights. Then given this prior, which you can think of as a code to compress the weights, you look at the coding length of the weights. Now, the prior that makes this thing make sense is what's called the adopted prior, which works as follows. Take all the datasets in the world. On each of them, train your model. You get a set of weights every time, then average out these weights with the distribution with respect to these weights, and then compute the coding cost of storing the weights that you train on your particular dataset, and you get a number. It's crazy. Nobody's going to train a model on every single dataset of the world and then marginalize out all the datasets. It's obviously only conceptually viable, but even if you could do it, doesn't make sense, and the short answer is does not. Here's how we see it. Let's say you train a dataset on some classification task. Let's say you want to discriminate airplanes from cartoon characters. Very useful task. You train a model however you want, you get a set of weights. These are numbers sitting on the memory of your computer. Now you can take every single datum and attach it to the weights after the least significant bit because typically we have many more weights than data. Now what happened is that, of course, the coding length of the weights will be bigger because you have more stuff to store. But you expect that the performance of the network will improve. Not a bit because you already optimize the training loss, and so this type of complexity or information that you stored is useless, and yet it compounds into the Classical Shannon information theory. How do you solve this problem? It turns out Shannon had the right idea in a paper in 1950, where he tried to measure the entropy of the English language, and the idea was as follows, write a sentence and then mess with some words or some characters, and then see if you could reconstruct the exact sentence despite these perturbations. If I can change this M to any letter I want, and I still am able to say that this was supposed to be an N, there's no information in this M, so I can just remove it or replace it with something at random. I certainly will not pay any cost to store it. Now, for the case of Shannon, it was easy because the task was a dataset. You take the data, store it, transmit it, you want to recreate it. But more in general, you have arbitrary tasks. Let's go back to cartoon characters. But the same idea can be applied, but except with one twist. Now, the amount of information in tree models depends on the type of performance you want. If I'm happy with 50% performance, to use Al Pizano's example, if you want a weather predictor for San Diego that is right 50% of the time, I need zero information. I just return sunny every day, and I'm right, how many 49% of the time? Zero bits. But if I want higher percentage, I need to be more sophisticated, and what happens is that if I use a trivial classifier and I'm happy with whatever percentage time, in this case, let's say 75%, I use a color togram and say, when there's blue, it's an airplane. Otherwise it's cartoon character. Even if I perturb the data a little bit or I perturb a classifier, which is the same, the trust doesn't change. It stays within 75%. But if I want 99.9%, now I have to account for the fact that there may be airplanes that look like cartoon characters. Now that's reflected in the fact that as soon as I make some perturbations, now my training loss goes up, and so now I need to allocate a lot more bits to this boundary. Here are how it would look in formulas. You train a network however you want, it doesn't matter. Stochastic deterministic, you guess, whatever, doesn't matter. You get a set of weights. Now, on these weights, you can perturb them using samples from whatever distribution you choose. Let's say Gaussian, pick whatever you want. You do so and you add more and more perturbation until your training loss reaches a threshold. Now, the maximum amount of noise that you can add to your weights that will not change the training loss is information that is not accessible. It is useless. The model doesn't see it. You might as well throw it away, and whatever is left after you remove that is the amount of information that the train weights contain. Even though the weights are not a random variable, so entropy is zero, but you can define a notion of information that is well-defined, well posed, computable. It requires solving a constraint optimization problem just to get a number. But the beauty of this is that if you write this in optimization in Lagrangian form and look at the second order approximation of that, you get your Fisher information, and that's one of the

key objects in information geometry. The official information, which is a matrix that you can compute with the weights of a tree model. It doesn't require you to believe in the existence of some underlying distributions or model. It's fully instantiated, computable, concrete. You get numbers out of it. Here's a connection already that emerges the moment you try to get into the details of what does it mean when you say that model has a lot of information. It does in the sense of information geometry, not in the sense of classical information theoretician. It turns out that in this equation, if you use this funky prior that I mentioned, you would get this bound. But again, the Shannon information is not well-defined, but it does relate to generalization. Now, the Fisher information is well-defined. It's what happens if you plug in an uninformative prior. Except that it does not relate to generalization, and then you have also Kolmogorov algorithmic information, which does relate to generalization, and it is well-defined except that it's not computable. You have all these notions of information none of them quite fits. But I will argue now that they're actually quite connected, so the beauty of this way of measuring information is that you can actually measure it while you're training models. Because the official information tells you the curvature of the loss landscape, not just at convergence, but as your solution moves along. If you do that for a typical neural network, it looks like this, which is super weird. The blue curve is the trace of the fisher information as you train a model for a number of epochs, and so the fisher information initially goes up, you accrue a lot of information, you memorize. Then at some point, you start forgetting. You start shedding information. Until you settle to a point where there's very little information, and yet the accuracy in the test set keeps going up. This means that to learn, you have to forget, which I'm very happy to tell my wife when I forget something that I'm learning. It doesn't quite work, but I want to do a slide detour on biology. Many of you may be familiar with Torsten Wiesel and Paul Hubel, the pioneers of modern neuroscience. They did some experiments on critical learning periods. Critical learning periods is the phenomenon by which if you don't learn a skill early enough, doesn't matter what you do, you're not learn it. This is why you cannot teach old dogs new tricks. This is why if my son was born with an optical defect, and so if you don't correct it early enough, you remain embryopic. What they did is they took cats that had congenital cataracts, and they remove the cataract after one day, two days, five days, 10 days, 100 days. Then they observed the visual acuity of the cat at full maturity. A normal cat has visual acuity. The units here don't matter, but that increases up to a point after about 100 days, the visual system is fully formed. This is the normal cat. Now, if you have a cat with cataract and you remove the cataract on day zero, there's no more optical defect, and so eventually the cat can see as well as the normal cat. But if you remove the cataract after 10 days, or 20 days, or 50 days, or 100 days, the cat is functionally blind. It's not an optical problem. It's a neurological problem because your visual system doesn't form, and there is an irreversibility whereby if you don't get it right early on, you don't learn. This is a skill. It's true for songbirds, it's true for locomotion. It's true for a variety of skills across species. There's another way of viewing this, which is instead of correcting the deficit after a certain window of time, you pick a window of time and apply the deficit on a sliding window. If I take my glasses off now for a month and then I put them back on, there's no difference. But if I need glasses and they don't put them on me when I am a baby, I will never see normally. The sensitivity to critical periods in cats is a curve that looks like this. It is maximal after about 25 days, and then it decreases. Even if you mess up with the optics, after 100 days, the cut doesn't matter. It doesn't mind. This is what these plots for cats look like, and this is what they look like for neural networks. When my then student, Alexandra Kelle, suggested doing this experiment and said, You're out of your mind, critical periods are due to loss of neuronoplasticity, which is a biochemical phenomenon. There's no biochemistry here. The connectivity is fixed at the outset, stays fixed forever. But it turns out that this is not a biochemical phenomenon. It's an information phenomenon. There is a concept of information plasticity. By the way, also, I forgot this, which is that if you take this plot, which I showed here, and overlay it to the trace of the Fisher information, this is what it looks like. Isn't that funny? This is, again, okay, the spirit of Dr. Amari showing up where the maximum of Fisher information during the training process is exactly the maximum sensitivity to critical period. You don't want to mess with your data during this period. You don't want to have any deficit. After that, it doesn't matter. The

Fisher information connects to the curvature of the lost landscape, connects to sensitivity to critical periods. There is this notion of information plasticity, whereby if you remove the deficit early enough, you can see the information in different layers cross paths. Information gets exchanged between different parts of the network. But if you wait too long, then they stay monotonic, and so there's no more exchange of information across networks. What happens if you want to visualize it geometrically is that the visual information corresponds to the curvature along the path. These networks, as you train them, their solution in this very high-dimensional space has to find this narrow corridor, high curvature, after which the valley opens up into a wide landscape, and where you converge there, it doesn't matter. Classical Konov regularization theory is about smoothing out the lost landscape at convergence. It's not relevant here. The beauty of this is you don't need to perturb the weights. You don't need to add noise to the weights because, as it turns out, stochastic gradient descent already computes the scores, the gradient of the log posterior. Effectively, stochastic descent implicitly imposes this regularizer when you train the model. This is called the information lagrange. I mentioned that there are these three types of information, Shannon, Fisher, Kolmogorov, and they're disjoint in general. But when you're training a deep neural network using Stochastic Ring Descent, they are linked because you cannot minimize one without minimizing the other. Their value doesn't matter, but they all go up or down in sync with a loss. This is not widely known, but it is a connection to classical information geometry that is quite striking in my opinion. When people talk about information, I'm still nitpicking on the word here. There are two distinct aspects of information. There's the carrier, the object that carries information, and then there's the number, the value of information. In Shannon, the carrier of information is not the data. The data contains no information. It is the probability distribution from which the data could have been drawn. That probability does not exist. Not in my opinion, this is definity. The probability is carried by measurable functions, random variables, and the amount of information they carry is the entropy of those random variables. In Kolmogorov complexity, the carrier of information is the program that if you execute through a Turing machine, generates the data you have observed, and the amount of information is the length of that program. Completely different. Now, neither of them addresses the semantic aspect of information. Does the amount of information go up with the meaning of a concept or a datum? Shannon averted the problem, meaning has nothing to do with information with communication engineering. We're not going to worry about it. Kolmogorov tried to establish a connection between his notion of complexity and the structure in the data as opposed to the noise in the data, and he failed. The Kolmogov structure function, which was his last work, failed in the sense that every time you try to say something universal about semantics, it becomes trivial. Every day, Tom has the same amount of information. How do we think about meaning today? Because when you play with these AI agents, they seem to be saying meaningful things to you. They seem to be taking meaningful actions. It turns out there are measurable functions, carrier statistical information, or Shannon's. There's computable functions, carrier algorithmic information, and there's conceivable functions, and these are LLMs, large language models. I'll say what conceivable functions mean. These are concepts. Abstract concepts are computable functions. They're finitely encoded in any of your brains, finitely computed, finite memory, finite time, finite compute. Now, the problem that has, in my opinion, dominated classical machine learning for the past 50 years, so much so that we don't even talk about it anymore, is the problem of induction. In machine learning, when I showed you generalization bound, I said, well, future rewards, past performance is indicative of future reward, so long as you forget. I tried to convince my financial advisor that is a theorem. He didn't quite buy it. The reason is, it may be true in many cases like weather in San Diego. But otherwise, there is a problem, and not my discovery. This is David Hume, 1736, who was after tried to understand the logic of induction. Why do we humans apply inductions? Because we learn something, it works, and then we apply it forever after. Induction is from specific data that we've seen in the past to any general decision or action that we want to take in the future. If there is a logic, Hume says that logic would imply that future data must behave like past data, and that the course of nature is stationary, which is patently false, and therefore, there is no logic to indoge. Yet, this is what we do. The issue is that in philosophy and epistemology, this

problem is unsolvable. There are thousands of papers and books written, but you cannot solve it because it always rests on the belief that the future will behave like the past. In mathematics, we bypass it. We say, all data is restricted to be generated by logical production rule, which we decide at the outset. Somebody indose marker decide some axioms, and therefore, everything is contained within that system of deduction. In biology, this is what we do. It's called System 1 behavior, reactive behavior, because it works most of the times. When it doesn't, evolution takes care of it. You die, but your species phylogenetically learns the lesson. But the game changer is that in agentic AI, the problem of induction is absent. That has been a game-changer since 2023 that has forced me to let go of all the things I've learned, all the things I've taught, and all the beliefs I held firm, and it's been quite a riot since. I will not get into too many details about that, but I want to give you a sense of how, again, some of the teachings from Professor Amari come back even in this very abstract concept. What are meanings? I went to look up a few books on epistemology, primary epistemic philosophy. Every time you take all these words and try to come up with a formal definition, it's circular. Either it's circular or ends up with consciousness somehow. I don't know what that means, but I like this so-called deflationary theory of meaning, which, in the words of Tarski are basically the meaning of the sky is blue is the sky is blue. If you're an engineer, you're thinking, okay, what exactly does that mean? Well, the sky is blue is a series of tokens in some language. The sky is blue without quotes in a different fonts is another sequence of token in some language. It is yet another sequence of token and so on and so forth. All of these expressions, data, are equivalent in the sense that none of them is special. Therefore, their meaning is none of them, but it's whatever relations they have among each other, which is an equivalence class. It's an equivalence relation. Meanings are equivalence classes of expressions. Simple as that. Now, how do modern AI systems establish systems of meanings? When you train a transformer, which is the backbone that drives LLMs, you have a sequence of vectorized tokens. These are objects that represent words or pieces of words, but they're vector in some embedding linear space. You feed them through your parametric model, and you do it sequentially, word after word, until a special character, which is the end of sequence. It's the end of the world for the model appears. Every time you pass a token, you get a vector in this embedding space, and you train the model to minimize the log posterior, which is the exponential of the inner product between the vector that you produced and the one-hot encoding of the next word, which are the coordinate axes of this 100,000-dimensional space. In other words, the training process takes the entire mass of all the data you use for training, which is pretty much a big chunk of the Internet, and squeezes it onto the coordinate axis of this 5,250,000-dimensional space. In the process of doing that, you establish an inner product, you built it. The entire space is empty because you only use the coordinate axis. Once you get to the end of the world, the model still generates a vector. But the vector has no token that it represents, but it represents the entire sequence. All the inputs that map after the end of sequence token to the same vector as measured by an inner product are equivalent, and the equivalence class is represented by an inner product, and that is the meaning of those sentences. All of the sentences share the same meanings, which is an equivalence class represented by this vector. You have your inner product between scores. The last gradient which you obtained during training is the score. Now, there's another view that is almost equivalent, but not quite, which is when you take the model and, instead of waiting until the end of the world, you generate continuations by just feeding the next token auto-aggressively. If you do it many times, you generate many different trajectories of token. Is there a notion of provisional meaning? Until you've said everything, maybe not because if you've seen the movie Borat, the joke, This suit is black, not. Until you've said not means the suit is black. After you said no, it means it's everything but black. But it turns out that even though all these continuations are stochastic because we inject stochasticity by sampling from the posterior, the conditional distribution of continuation given the input is deterministic. It's a deterministic function of the train model. If you look at all the inputs that produce the same conditional distribution of a continuation measured by Kubacheer, that's also an equivalence class. That's also a representation of meanings. It is a representative meaning that has some advantages because it allows you to represent not only symmetric relations like linear product you can measure

semantic similarity, but also you can measure things like entailment, containment, and causal relations, which is what we want these models to do. In what sense are LLMs then allowed to define concepts? I have to open a small parenthesis because when I realized that everything I held dear didn't help, I had to go back and understand how people were thinking about these problems before these theories were established. It turns out that Solomonoff, in 1964, tried to design a decision algorithm, an inference algorithm that is optimal, not just with respect to data sampled from the same distribution, whatever it is that you saw in the training set, but with respect to any distribution, a universal predictor. You design an algorithm that does just that. The only problem is it takes forever. It's incomputable. But in 1973, in the same three-page paper in Russian where Leonid Levin introduced the notion of NP completeness, he also introduced an algorithm which is not only universally optimal but universally fast. Meaning that if you give me any task, which I can represent by a function that for every query and for every possible outcome gives me a score. It could be a verifier score of zero or one or a reward function. Then you design a solver specifically optimal for solving this task, which it does in time T_A. Then there exists a universal solver that, for any task A, solves the problem also optimally in time no greater than the minimum time with a constant that's due to the length of the program. You cannot beat this. You cannot beat this, but this constant is astronomical. One important thing is that this is optimal decision-making. There's no learning. Something is off here. It turns out the same Solomonoff, in 1985, pointed out, What is the role of learning in optimal decision-making? He said, Well, the length of the solution to an optimal algorithm, you can think of it as the probability of sampling a program that if executed by a Turing machine, will solve your task. This is just Shannon's coding theorem. This is where experience comes in. This is where learning comes in. If you're a mathematician, you have proven many theorems. You know many lemmas; you have a lot of tricks. Every time a new theorem comes to you, you can deploy tricks that allow you to find the solution much faster. You can always find a solution. You put the monk in front of a typewriter, start typing, sooner or later, you will find proof. Of course, it will take a while. But experience comes in in reducing the time it takes to find optimal solutions to unforeseen tasks. Except that in 1985, who was going to give you a probability distribution of programs that the front-ray machine will solve an unforeseen task? Completely abstract, no relation to reality. This is what a coding program does today, an LLM. A coding LLM is a code that given a prompt that specifies the task, generates a distribution over continuations, which are programs in Python, Rust, whatever you want. Code models are the backbone of all reasoning engines that are being built today. Except that LLMs are not computers. The theory works for universal Turing machines because they are universal computers. They can compute any computable function. What about LLMs? A compute. If you ask a question that requires doing computation, like what is the largest year of this polynomial? The model will start spitting out tokens in chain of thought. These tokens are sampled from the model, and eventually we will give you an answer, and you can test that answer and see if it is correct or not. Now, there is no program. The prompt is not a program; it tells you what the problem is but doesn't tell you how to solve it. The chain of thought is not the program; it doesn't even exist before you start on the problem. The weights are not the program because they're once and for all. In what sense is this a computer? It turns out there is a precise sense in which it is a universal computer. A computer in general is a dynamical system that starting from an initial condition that specifies a task, evolves to a final state from which you can output a solution to the task. Turing machines are deterministic and minimalistic. They're designed to have the smallest number of symbols, operations, states, so we can prove theorems with them. LLMs are exactly the opposite. They're stochastic, and they're maximalistic. Typically, the bigger, the better. But they are nonetheless universal computers, and this is the theory that's been developed. Once you understand them as computers, there's a question: what is the program length? Because that's the notion of complexity in computability theory. There's no program. Is it the length of the chain of thought? No, because that could be unbounded. It turns out that you need to define a new notion of time. This is related to relativistic proper time. Don't ask me to get into details of that. But there are theorems that now guarantee that an LLM can find a solution, maybe not the ones that we have today, but maybe the ones that we'll have in three weeks

or three months or six months can find an optimal solution which is as good as the optimal bespoke expert for that particular task in time no greater than a constant factor of the optimal. This is made possible by the fact that there is a theory that can be transposed using completely different proof techniques to other LLMs. This is what transductive code generation means. It's models that generate code on the go, not given a prompt, generate code that will have to work forever after as an induction, but for any specific instance, generate a specific piece of code that solves that instance and do it for any instance. There's a little theorem hidden here, which is that if the goal of learning is to maximize the amount of algorithmic information, all the tricks and the pieces of algorithms that you can piece together to solve any new task. That's the goal of learning: to maximize the amount of information that optimal solutions to future tasks share with past data you observe that is equal to the speedup in finding optimal solutions to new tasks. In other words, you learn by becoming faster and faster at finding solutions to tasks you've never seen before. But this says a little more because it says you learn only by doing so. This is inequality. It's not a bound. This is, in my opinion, quite a game changer. Last piece is, some of you may have seen these very frightening scaling laws that basically says, as these LLMs get bigger and bigger and bigger, measured by FLOPS, floating-point operations, the error in log log scale follows this nice linear trend, which seems to be going on forever. You multiply the size of the model that the set or time by 10, and your error goes down by half or whatever factor. You can keep going forever. You can always reduce the error by half, and then again by half, and then again by half. In fact, this keeps going on forever, and the limit of this getting love is Solomonoff induction, which is optimal transductive inference, which requires infinite time, infinite memory, infinite power, infinite energy. But there's no intelligence there. The amount of induction is a search of programs. No memory, no learning, no adaptation, nothing. These curves are true, but they measure the proficiency of models in benchmark tasks. But they cannot measure intelligence, let alone superintelligence, because at the end of this curve, there is zero intelligence. How do you measure intelligence there? It turns out that time has a lot to do with it. There are some students here. How many of you have ever taken an exam with no time? No time limit. One. That must be a cruel advisor, yes. There is a cost to time. If you do a math exam, for example, if you return all the right answers after the exam is over, you get zero point, not 100%. There comes a time when if you factor in time in a way that measures not the loss in the validation set but the loss plus lambda proper time, which we call conceptual depth, which is analogous to the logical depth of the data, then you see an inversion whereby beyond a certain point, depending on the environment, it's not just a diminishing return. You damage yourself by spending more resources because you run out of time or because if you're a zebra, a target will eat you, and so on and so forth. Adapting to the cost of time, which is specific to a particular environment, particular task, particular user, is necessary to avoid what we call the savant regime of AI models, which is this regime, which, yes, does improve performance, but not by models becoming more intelligent, by becoming less intelligent, because as you get more and more power, you can afford to do things with brute force without any insight, without any learning. But if you have to pay for time, then you necessarily must learn because the only way you can reduce time is by accumulating algorithmic information in the train model. Without time, there's no intelligence. Models have to learn the cost of time in their particular environment. There is no universal cost of time. Kolmogorov tried to do that. If there is no universal cost of time, there is no model which is optimal for all environments. If you're concerned about the GI, that you have, and the way we put programmer LLMs is, some of you here program them in English. You're familiar with equip that the hottest new programming language is English, but you may not know Dijkstra's quote, which is that, "Projects promoting programming in natural language are intrinsically doomed to fail". You see it in the news increasingly. Transductive co-generation is what allows you to program these models for what they are, which are stochastic dynamical systems, not your machines, which you do with two-level controls, open-loop global planning, and closed-loop local control. Unlike transductive co-generation, where you produce a piece of code that has to work forever, which is very difficult, and it will break because tomorrow there will be new file types, new environments. But you do transductive co-generation, which is to write a piece of code that works for the present task in the

present-day room. Next time you run the code, it could be a different piece of code. This is actually something that we open-sourced two weeks ago. I understand the betrayal may be a little bit high here, but there's a lot of stuff that's going on that even though it may seem messy and it may seem complex and may tap on language that spans different fields from information theory to complexity theory to epistemology, some of the ideas related to the geometry of the lost landscape or even the representation of meanings date back to concepts that were expounded and described in the work of Dr. Amari. I am very excited to live in this time because the theory is being built as we speak. As we are building these models, we are trying to understand them and then explain them. The theory is not there yet, but I'm confident that if nothing else, by using as guide the clarity and simplicity and elegance of the work of Dr. Amari, we'll be able to create theories that are as effective and elegant and easy to learn. Thank you. [Applause] [Music]