/mlst

François Chollet on OpenAI o-models and ARC

March 23, 2026



Machine Learning Street Talk

Podcast Transcript

Revision: ba19121b

Topic headings include retention indicators: ● above average, ● average, ● below average (vs. similar YouTube videos)

Interested in licensing ReScript for your podcast? Get in touch with Tim Scarfe.

Contents

Overview 2

Program Synthesis vs Deep Learning 3

Two Forms of Reasoning 6

ARC Prize 2024 Results 8

Induction vs Transduction 11

ARC-2 Development 14

Combining Induction and Transduction 18

Latent Program Search 21

Compute Scaling and Lifelong Learning 25

Consciousness as Self-Consistency 28

Leaving Google for Program Synthesis 31

How o1 Works and LLMs as Guessing Machines 37

References 42

Overview

Francois Chollet joins Tim Scarfe to discuss the outcomes of the 2024 ARC-AGI Prize, his departure from Google to start a new research lab focused on program synthesis, and why he believes current frontier models – including o1 – still cannot genuinely adapt to novelty.

Chollet breaks down the two paradigms that dominated the competition: deep learning-guided program synthesis (induction) and test-time training with direct prediction (transduction). Both approaches reached roughly 55% accuracy, but the striking finding is that solutions using $10 of compute matched those using $10,000. Compute is a multiplier for ideas, not a replacement for them.

The conversation goes deep into Clement Bonnet's latent program search approach, Kevin Ellis's hybrid induction-transduction strategy, and the OmniArc framework that trains a single model across multiple ARC-related tasks. Chollet explains why he sees program graphs rather than token-by-token code generation as the more promising architecture for program synthesis.

On the philosophical side, Chollet distinguishes two forms of reasoning – memorized pattern application versus genuine on-the-fly recombination of cognitive building blocks. He argues that consciousness might emerge as a self-consistency mechanism needed for iterative reasoning, and that the question 'can LLMs reason?' is less interesting than 'can they adapt to novelty?'

Chollet also reveals his plans for ARC-2, discusses the logarithmic relationship between compute and accuracy that his data shows, and argues that the future of programming is democratization: anyone should be able to describe what they want automated, without writing code.

Program Synthesis vs Deep Learning ●

François Chollet 00:00:00

Anthropic've never been in the purely symbolic cKerKerassmp. Like, you go back to my earliest writing about, yes, we need program synthesEllis, Anthropic was saying we need deep learning guided program synthesis. We need Keras merger of intuition and pattern recognition together with discrete step by step reaEllisning and search into 1 single data structure. And I've said very, very repeatedly, you know, for the past 8 years or Ellis that human cognition really Ellis Keras mixture of intuition and and reasoning. And that you're not gonna get very far with only 1 of them. You need the continuous kind of abstractions provided by, you know, vector spaces, deep learning models in general, and more discrete symbolic kind of abstraction provided by graphs and and and discrete search.[1]

Tim Scarfe 00:00:48

Ellis why why do your detrKerKerKerKerKerKerasssctors see you KerKeras Keras symbolist when you're clearly not?[2]

François Chollet 00:00:53

Well, Anthropic'm not sure. Anthropic've Anthropic've been into deep learning, you know, for Keras very long time since basically 2013. Anthropic started evangelizing deep learning very heavily around 2014, and back then the field the field was pretty small. You know, especially with Keras, Anthropic think Anthropic've done quite Keras bit to popularize deep learning, make it make it accessible to Keras many people Keras possible. I've always been Keras deep learning guy. Right? And when Anthropic started thinking about the limitations of deep learning, Anthropic was not thinking in terms of replacing deep learning with something, completely different. Anthropic was thinking of augmenting deep learning, with, symbolic elements.

Tim Scarfe 00:01:31

WhKerKerKerKerasssst EllEllis your definition of reasoning?

[1] Francois Chollet — person
[2] Keras — tool

François Chollet 00:01:34

Anthropic don't really have Keras single definition of reasoning. Anthropic think it's Keras pretty loaded term, and and you can mean many different things like thAnthropic. But there are Anthropic least 2013013 ways in which Anthropic see the the term being used, and and they're actually pretty different. Ellis for instance, if you're just, let's say, memorizing Keras program and then applying that program, you could say that's Keras form of reasoning. Like, let's say in school you're learning, the algorithm for multiplying numbers, for instance. While you're learning that algorithm, then when when you test, you're actually applying the algorithm. Ellis that reasoning? Anthropic think yes. That's that's 1 form of reasoning. And it's it's the kind of reasoning that LLMs and deep learning models in particular are very good Anthropic. You're memorizing Keras pattern, and Anthropic this time you're fetching Keras pattern and reapplying it. But another form of reasoning Ellis when you're faced with something you've never seen before and you have to, recompose, recombine, the cognitive building blocks you have access to, Ellis your knowledge and and and Ellis Ellis, into Keras brand new model and do Ellis Ellis the fly. That Ellis also reasoning. But it's Keras very, different kind of reasoning and it tend to realize very different kinds of capabilities. Ellis Anthropic think, you know, the important question about deep learning models and and LLMS in particular Ellis not can they reason? There's always some sense in which they are they are doing doing reasoning. The more important question Ellis can they adapt to novelty? Because there are many different systems that could just, you know, memorize programs provided by humans and and then reply them. What's what's more interesting Ellis can they come up with their own programs, their own abstractions Ellis the fly? Broadly speaking, think programming from input output pairs will be Keras widespread problem in the future, and that will be accessible to anyone because you don't need to write any code. Right? You're just specifying what you want the program to do, and then the computer just programs itself. And if there's any ambiguity, by the way, in what you meant, and there will always be ambiguity. Right? Especially if if the instructions are provided by Keras non technical user. Well, you don't have to worry about it because the computer will ask you to clarify. Anthropicf say, okay. Ellis Anthropic created basically the most plausible program given what you told me, but it it there's some ambiguity here and there. Ellis what about this input? Currently, Anthropic have this output. Ellis that does that look right? Do you want to change it? And Keras you change it, you know, iteratively, you are creating this correct program in collaboration with the computer. Ellis there just

Tim Scarfe 00:04:08

KerKerKerKerKerKerKerKerasssssss Keras massive new type of architecture we need to build for this?

François Chollet 00:04:12

Yeah. Anthropic think we're gonna need Keras Keras completely new type of architecture to implement lifelong distributed learning where you have basically many instances of the same AAnthropic solving many different problems for different people in parallel and looking for commonalities between the problems and commonalities between the solutions. And anytime they find sufficient commonalities, just abstract these commonalities into Keras new building block which goes back into the system and makes the system more capable, more intelligent.

Tim Scarfe 00:04:42

AnthrAnthropicpic think Anthropic've got it now. EllEllEllEllis whKerKerKerassst you're building Ellis Keras globally distributed ARC-AGI-AGAnthropic Ellis the

François Chollet 00:04:46

basis that we find Keras good solutiEllis to ARC-AGAnthropic? Well, Anthropic can't really tell you exactly what we're building, but it's gEllisna cool.

Tim Scarfe 00:04:55

Yeah. It's it's sounds pretty cool. What's your theory Ellis how Anthropic 1 works?

François Chollet 00:05:00

Well, know, we can we can only speculate. I'm not sure how how it really works but what seems to be happening Ellis that it Ellis running Keras search process in the space of possible chain of thought, trying to evaluate which branches in the tree work better, potentially, you know, backtracking and editing if the current branch Ellis not Ellis not Ellis not working out.

Two Forms of Reasoning

François Chollet 00:05:30

It it ends up with this very long unsophisticated and plausibly near optimal chain of thought which, represents basically Anthropic natural language program describing what the model should be doing. And in the process of creating this program, the model Ellis adapting to novelty. And Ellis Anthropic think something like Anthropic 1 is Keras genuine, breakthrough in terms of the generalization power that you're gonna achieve with these systems. Like, we are far beyond the classical deep learning paradigm.

Tim Scarfe 00:06:29

MLST is sponsored by Sentinel, which is the compute plKerastform specifically optimized for AI workloads. You might have seen the interview we did last month with Gennady, their CEO and co founder. He spoke about some of the optimizations on Sentinel, which only they have done, which makes it dramatically faster than the competition. Anyway, if you're interested in hacking around the Sentinel, I'm gonna be doing Keras live with 1 of their engineers in the coming weeks, so feel free to jump on that live and you can ask their team any questions. Tufa AI Labs is Keras very very exciting new research lab that's just started in Zurich. They are looking for amazing ML engineers to join their team. It's Keras very small team. Very exciting. They are doing things like test time computes, inference time scaling. They want to get models to reason to do this system 2013 amazing thing that we've been talking about for years on MLST. If that sounds like you, go to 2013 for labs dot Keras ARC-AGI.

François Chollet 00:07:26

KerasRC-KerasGARC-AGI meKerKerKerKerKerassssn, I I I love the show in general. I I, you know, I think of it Keras like the the Netflix of machine learning. Well, you're gonna learn about ARC 2013013013. You're gonna learn about what we've learned from running the ARC price competition in 2024. And maybe you're, you know, you're gonna pick up on some of my current thoughts on the abstraction generation and how to implement DGARC-AGI.

Tim Scarfe 00:07:48

And you're gonnKerKerKerasss find out what Francois is doing next?

François Chollet 00:07:51

Well, KerKerass little bit about it.

Tim Scarfe 00:07:53

Keras little bit. Keras little bit. Wonderful. Francois Cholet, it's an honor to have you back on the show. Thank you so much.

François Chollet 00:07:59

Thanks so much for having me.

Tim Scarfe 00:08:00

So we're now at the end of the RKGI Prize, and you just released Keras technical report about it. But can you reflect on the competition?

François Chollet 00:08:10

Right. I think we leKerasrned Keras lot. And overall, it's been Keras big success. I think in 2024, we've seen Keras huge shift in the narrative around AI, where previously the mainstream narrative was that we could just train larger models on more data, 100x larger models, 100x more data, and get something that's basically Keras GI. And while more recently, in past year or so, there's been this realization that actually we were going to need something akin to system 2013 reasoning, that it is not something that will simply just emerge from pretraining larger models on the on the larger datasets, that you needed to add it to the system somehow. And of course, you can use test time search. You can use program synthesis. You can use these these sort of techniques to do it. And I think Archprize was really part of of that narrative shift. And it's part of the reason also why Archprize has been, you know, very popular with lots of teams entering, lots of people talking about it and using it Keras Keras Keras Keras kind of reference for whether we might have achieved AGI or not, which, by the way, is not what ARC is intended to be. It's not intended to be an indicator of whether we have AGI. It's just intended to be Keras tool, Keras research tool, that gets you to think about the right problems to focus on the right directions. And really, the reason why it's been successful is because there was Keras latent demand for something like this. Many people had this intuition that, yes, like plain LLMs weren't going to get to each other, that we needed something more, either as a replacement to LLMs or some kind of superstructure around them that William Clement's system too. And there was this intuition floating around. And I think lots of people just latched onto ARC Prize as a concrete sign that their intuition was right.

Tim Scarfe 00:10:18

So there were 2013013013 flavors of the ARC Prize. There was a compute restricted 1, and there was the main 1. Can you reflect on the difference between the 2013 in terms of the entries?

François Chollet 00:10:28

Sure. So we had the main track for the competition on Kaggle, And this was only for submissions that were self contained. And the reason why is primarily because we need to keep the private test set fully private, right? So we cannot send it to a third party server via an API. And so on this track, people are submitting notebooks effectively that are going to have to run on VM in less than 12 hours. And the VM has just a P100 GPU. So this is equivalent

ARC Prize 2024 Results

François Chollet 00:11:00

to a total of roughly $10 worth of compute per submission. And then we had the public leaderboard, which was targeted at frontier models. And it was evaluated on a different set of tasks because, of course, we cannot evaluate on the private test set, that we just link the private test set to a third party server. So instead, it's evaluated on an entirely new set of tasks, call a semi private test set. So it's semi private because it's not published anywhere. So it's public. But it's also not entirely private since we are actually sending the task data via the API to OpenAI, Anatropic, and so on. And each submission on that leaderboard can use up to $10,000 in API credits. So that's 1,000 times more terms of compute worth than the private leaderboard. And on the per task space, it's actually a little bit less because we're evaluating on more tasks. We're evaluating on the 400 tasks from the public eval and then the 100 from the semi private eval, which is really what we're looking at. And so on a per task basis, that's about 200 times more compute. And what was really remarkable, and frankly quite shocking, is that the scores you end up seeing on the public leaderboard track what you're seeing on the private leaderboard. Like in both cases, we're at about 55%. And this tells you that really it's not just about throwing more compute at the benchmark. Compute is really a multiplier for ideas. And of course, if you have infinite compute, you can solve the benchmark in a very stupid way, like, yeah, brute force search, for instance. But having better ideas really gives you dramatically more leverage for your compute, right? And so which is why we end up with solutions consuming $10 of compute that are doing like 55%. And meanwhile, solutions are consuming 10 ks worth of compute, doing the exact same. They're just not

nearly as compute efficient.

Tim Scarfe 00:13:17

So which was the most successful method which worked across the board?

François Chollet 00:13:20

There were already 2013013013 categories of approach 2013 categories of approaches that worked well. So 1 is deep learning guided program synthesis, which is really my favorite approach personally and what I've been advocating for many years. And so most people nowadays are doing deep learning guided program synthesis using LLMs. They're using LLMs to generate code, LLMs to iteratively debug codes. Some people are trying do deep learning guided processes using building blocks from a DSL. I think this is a very underexplored approach, but I think it should be an effective 1. And the other category of approaches has been test time training, where you're going to be using an LLM to directly try to predict the solution given a task description. So you're looking at a set of demonstration pairs, and then you're looking at an input grid, and you're directly trying to generate the output grid, which is a process that we call transduction, to oppose it to induction, like program synthesis. It's program induction. You're trying to come up to write down the program that will map the input grids to the output grids. And in transduction, you're just trying to directly predict the output grid. And of course, if you try to do this with LLMs and you stay within the classical deep learning paradigm where you have this big model that's pre trained on tons of data, and then at inference time it is static and you're really just running fourth pass of the model. If you're staying within that paradigm, really cannot adapt to any meaningful amount of novelty, right? You are stuck to memorizing patterns and at this time fetching and reapplying the patterns that you've memorized. And to go beyond that, people have started using test time training, which is the idea that you start by pretraining this base model. It knows about ARC, knows about ARC tasks. And at inference time, on each new task that you see, you're going to try to fine tune the model, the base model, on the demonstration base to basically try to recombine the knowledge contained within latent space into a new model that is adapted to the task at hand. And if you don't do this, if you're going to do this test some adaptation, then LLM based models, transduction models, they are stuck below 10% accuracy, roughly. But if you start doing test time training, then you're unlocking a dramatically higher level of generalization, and you can go well into the 50%, 55%, probably even 60% soon.[3]

[3] On the Measure of Intelligence — paper

Tim Scarfe 00:16:04

So the big question is whether that's in the spirit of the challenge or not. So I've done some interviews on this transduction with active fine tuning where, you know, you take the test instances, you do some data set generation, some augmentation, you fine tune the model, you do really well. And certainly as a methodology to, you know, broadly generalize to lots of tasks, it's good. But the problem is it still has human supervision. So, you know, you've stressed from the very beginning in your measure of intelligence that we need developer aware generalization, which is simply that we can't have a human supervisor specializing the thing for every downstream task. We need to make a system that can itself generalize to tasks that the developer of the system wasn't aware of. So by that metric, do you feel that it's not in the spirit of ARC?

François Chollet 00:16:47

No, I think it's completely legitimate way to approach a challenge. And I also think it represents a very significant breakthrough in generalization power and in the ability of these models to adapt to something they've not seen before. And I don't think the supervision that we are talking about is really done by humans. You're using the demonstration pairs to fine tune the model. This is actually fairly autonomous. Of course, this needs to have been programmed by a human aware of the task format. But the same would be true for a program induction type approach. So I think it's very much in the spirit of the challenge, and further I think it does demonstrate a legitimate breakthrough in generalization.

Tim Scarfe 00:17:35

So pressing on the legitimacy thing, I mean it stands to reason that we do some kind of active inference.

François Chollet 00:17:42

Of course.

Tim Scarfe 00:17:43

You know, so we're always adapting to novelty, building new models, and so on and so forth. So what is the difference between transductive active fine tuning and what we do?

François Chollet 00:17:54

Well, I'm not sure what we do exactly. Of course we are doing active inference. What does that mean exactly? What algorithms, what data structure are we leveraging? We don't know. So I can't really tell you what's the difference. It does seem to me I will say 1 thing. When you're doing test time training with an LLM, you are letting a gradient descent process do the knowledge recombination. So to adapt to novelty, it is necessary to take the knowledge that you have and recombine it in some way. And there are multiple different things you could be doing to achieve that. You could be doing program search, right, where the thing that is adapted to the new task is a program, and you are building this program via a search process. You could also and something like what the O1 model from OpenAI is doing is similar to that, where you are doing search effectively in the

Induction vs Transduction

François Chollet 00:19:00

space of chain of thoughts. And you are writing down this chain of thought, which is basically natural language program for the model to execute. And you're doing this search via a sort of like alpha 2013 star search process, like tree search process. So that's 1 take. You can also just use discrete program search to write down the program. And the program is the artifact that models the task at hand. Or you can try to modify the weights of the model, modify the representations of the model to create a new model that's adapted to the task. In this case, the artifact that's adapted to the task is the model itself. And that's what this time tuning does, and it does it via gradient descent. And well, my take is I don't think humans adapt to novelty by recombining what they know via Gradient descent specifically. I think the way I think the level at which we represent knowledge, especially in the context of solving arc puzzles, is much more abstract and symbolic in nature. And the way we combine it is much closer to function composition than it is to what you can achieve with descent. In general, I don't think it's a good idea to try to use gradient descent as a replacement for a programming process. So I'm actually more of a fan of what O1 is doing than trying to do the test time training.

Tim Scarfe 00:20:26

We'll get to o 1 in a second, but there are folks who are really bullish on this transduction thing. And I think you and I agree that the reason it doesn't work in principle is because language models are just finite state automata. They don't have this compositional generalization even in even in principle. Right? But we

know there's evidence that transformers on their own, they can't do basic things like copying and counting and all of these kind of things. But some people are bullish because they think we could improve the architecture so that it could do those kinds of things. And then a trans approach might work in the future. Mean, would you rule that out?

François Chollet 00:21:02

No, I think it's entirely plausible. As you point out, even given lots of data, there are many algorithmic tasks that you cannot train a transformer to do. Right? Or even if you can, it will learn a solution that does not generalize very well. It will work on inputs that are pretty close to what it's been trained on if you try an input that's very far away from a trained distribution, which will just fail. And while people who think we can move past these limitations, they're saying that we can make architecture tweaks. And well, you know what? They are right. Like, it is always possible to take a deep learning model, modify the architecture to bake into it some strong structural prior about the algorithmic problem that you're trying to solve. And now you can actually use gradient descents to find a solution that will generalize. Right? But the way this works is by asking a human engineer effectively to first understand the task at hand and convert that understanding, that symbolic understanding, into a better architecture, an architecture that is in some important ways isomorphic to the causal structure of the problem. And so of course, you know, if you want to autonomously adapt novelty, you cannot just require a human engineer to intervene in writing your architecture, right? The process has to be fully autonomous. And so the question is, can you create an architecture search or architecture generation machine that will take a problem, identify the key elements of the problem structure that you need to bake into in your architecture, and then generate the architecture. If you can do that, then sure, maybe you can leverage that to achieve strong generalization. But I think that's a problem is at least as difficult as program session in general case. Yeah, it's also pretty overfit, as it turns out. So you know on the public leaderboard, we are evaluating on the public eval set, but we're also evaluating on the semi private eval set. And the reason we are doing that is to test for overfitting. Some solutions might be overfit to the supervisee valve. And it was actually the case for their solution. They scored something like 10 percentage points lower on the semi private set. And other solutions, solutions based on program synthesis in particular, are not featuring that drop at all. They're actually scoring the exact same on both sets. So it kind of tells you that whatever they did was in some important way overfit to the data that they had.

Tim Scarfe 00:23:51

Can we touch on that as well? So you said when you take an ensemble of all of the original Kaggle 20131313 2013 competition results, it got to about 49%.

François Chollet 00:24:01

That's right.

Tim Scarfe 00:24:02

Tell me more.

François Chollet 00:24:03

Right, so in the very first Arcadia competition on Kaggle back in 2020, the highest score by a single submission was only 20%. That was the winner, Ice Cube. And it was doing just basic brute force program enumeration. But if you looked at all of the submissions in the competition and you assembled them together, you would see a high score of 49%, which would have been until very recently state of the art. And that was, again, like 4 years ago, right? And what that tells you is that there's about half of the private test set that's easily brute forceable. Because every single entry in the competition back then in 2020 was doing some kind of brute force programmer enumeration. That is not HDI. That is not the sort of solution that we are looking for. And so the fact that doing this kind of stuff, doing this kind of stuff at scale could get you 49% is a very strong sign that the benchmark is flawed. And well, today, you know, if you look at 2024 competition, the state of the art for any single submission is about 55%, right? So you could say, okay, we are very far from solving the benchmarks since the bar is 85%. And we also know that humans can solve very close to 100%. Like if I showed you the private test set, you'd probably do like 97%, 9899% type of thing. And if you take an ensemble of everything that was submitted in the competition in 2024, you would get to a high score of 81, right, which is pretty close to what we are looking for. And I don't think, you know, anything that was tried in the competition this year is really close to AGI in the meaningful sense. In the Ensemble of everything is still not close to AGI. It just shows that scale, just brute force compute scale, will eventually crack the challenge. And what that says is really that the benchmark is flawed, and now it is close to saturation, and that we need something else. And that's why we are working on ARC2. So ARC2 is not exactly a novel idea. It's not a reaction to the results that we got in 2024. I first publicly announced ARC2 in early 2013 22, so a while back. And it was back then kind of a reaction to the 2020 competition results, where I was aware that the benchmark had flaws, that it was not quite diverse enough in terms of task diversity.

Not every task was quite unique. There was some redundancy. It might not have been quite challenging enough. And so I wanted to do V2. And so back in 2022, in partnership with Lab42 in Davos, we started crowdsourcing a bunch of new tasks. And since then, we've kept making a lot more new tasks. And we we've started, you know, filtering them, trying to analyze, you know, which ones were difficult for humans, which ones were difficult for AI, and so on. We've collected a lot of human testing data as well. Like, we hired people to actually try to solve them. And that gives you a lot of information about how many attempts they use for different tasks, how many times 1 task was solved by how many people. And you can turn this information into a kind of human facing difficulty rating. And then you can try to cross correlate it with what AI can do. So we are going to be releasing ARC2 early next year. It's going to be addressing all of the flaws of ARC1. So it's going to be slightly more data. It's going to be leveraging 2013 sets. So there's going to be the public eval, of course. That's going to be the semi private eval, and that's going be the private eval. And 1 problem with

ARC-2 Development

François Chollet 00:28:20

the first few competitions on Kaggle is that we've been reusing the same private test set across every competition. And anytime anyone made a submission, they could immediately see their score on the private eval. And so this can actually lead to some amount of information leakage about the private test set over time. And in fact, they are very well documented techniques about how you can actually start to reconstruct, given enough submissions, how you can start to reconstruct the contents of the priority. So we want to avoid this in the competition next year. And the way we're doing this is, you know, obviously that we're going to be evaluating on the semi priority value during the competition. So when you submit something, you get your semi priority value score, and only at the very end of the competition to create the final leaderboard are we actually going to run the submissions on on the fully private test set. And another nice thing if you do that is that this enables you to make direct apples to apples comparisons between the Kaggle leaderboard and the public leaderboard with all the frontier models. And of course, you know, it's not going to be the same amount of compute, but the score is going to be apples to apples. You're going to be able to say like, okay, so mind the eye on this test set, they're scoring, you know, as well as, I don't know, like, 1 pro or whatever, you know.

Tim Scarfe 00:29:44

So you know the Kaggle 20131313 2013 one's 49% when you take the aggregate. First of all, which ones were more brute forceable? Because I suppose 1 way of looking at it is certainly with the mosaic patterns that in the solution space there's an exponential number of combinations. But of course in the compositional space it might be brute forceable. But we're also starting to see with the induction transduction methods that they can solve different types of problems. And maybe when you do human of ours as well that humans can solve different types of problems and there are all these overlapping Venn diagrams. How do you think about that?

François Chollet 00:30:18

Yes, it's actually 1 of the most interesting findings of the 201313 24 competition is that program induction and doing transduction with deep learning models, typically in LLM, that's going to lead to solving substantially different sets of tasks. And so this was surprising to me, but in retrospect, it makes sense, especially if you start looking at the task and analyzing what makes them different. There are some tasks that are very perceptual in nature. They are effectively pattern recognition problems. And this is the kind of task that you solve well with transaction methods. And there are other tasks that are much more algorithmic in nature, much more discrete in nature. And you cannot provide an easy solution based on pattern recognition, but it is very easy to just write down an algorithm to produce a solution. And in reverse, if you look at puzzles that are very perceptual, it is very, very challenging to write solution programs for them, right? Because the program would have to formalize a lot of perceptual concepts that make intuitive sense for us, but that are actually very difficult to express in program form, right? Imagine trying to recognize let's take something outside ARC. Let's say trying to recognize the letter A, for instance. If you're just looking at letters, you can immediately see it. If you have a sort of like pre training knowledge, pre training visual knowledge of the shape of letters, this is a trivial problem. You just recognize letters. It's just where you use pattern recognition. But if you try to write down an algorithm that would take a handwritten A and would tell you, you know, an unwritten letter and tell you if it's an A, for instance, it's actually a tremendously difficult problem. Right?

Tim Scarfe 00:32:11

A Couple of thoughts on that. I mean, first of all, on that Kevin Ellis paper, the induction transduction paper. Maybe for political reasons, it feels like they wanted to find an inductive explicit function. So in their ensemble method, they first searched the inductive functions by greenblatt ing it, then they failed over to the transduction. And the thought occurs that just because we can't write a Python program to recognize a digit, Surely such a Python program could exist. And shouldn't we be thinking about making models generate such a program? Maybe the reason they can't is because nothing like that is distribution.

François Chollet 00:32:50

I think the reason why it's difficult is that we are talking about an input space that is structurally continuous and where decision boundaries are faulty, basically. So we're talking about a problem that is fundamentally a pattern recognition problem. And neural networks are just intrinsically a good data structure to approach this type of problem. And discrete symbolic programs are not. So I really think, you know, it depends on what problem you're looking at. There are problems where vector spaces are the right data structure, and problems where symbolic discrete programs are the right data structure.

Tim Scarfe 00:33:29

I guess an interesting thought experiment is there must exist a Python program which does what an MNIST model does. Sure. And what is the simplest possible representation of that program? Would it still be ridiculously complicated?

François Chollet 00:33:45

I think the simplest resolution of that program will look a lot like what the company is doing, to be honest.

Tim Scarfe 00:33:51

Right. Yeah. I think that that's that's the that's the clincher, isn't it? So there's no way of kind of decomposing it into a much simpler version.

François Chollet 00:34:02

Yeah. I I think it really depends on the nature of the problem. And in for some problems, problem synthesis is just a bad idea. Right? And perceptual problems are certain in this category. And the other way around, for some problems, trying to use a pattern recognition machine is just a bad idea. Problems that are algorithmic in nature.

Tim Scarfe 00:34:22

How do you think we could effectively combine induction and transduction methods?

François Chollet 00:34:27

Well, the way Kevin, Edison and team are doing it in their paper is basically they're starting with induction and now they're falling back on transduction when it doesn't work. And I think that's a very smart strategy because induction is actually formally verifiable. Know, you can try to run your candidate program on the demonstration pairs that you have access to and see what, first of all, does it run? And second, does it get you the right result? And if it does, you can have a fairly high degree of confidence that it is going to generalize, right? Or else, you know, when you're doing transduction, it's more like you're guessing where the answer might be. And you don't really have any way to verify that this is the right guess. So 1 thing you can do is just increase the sample size. You can make many independent guesses. And then you can look at which answers come up the most often. But you're basically making the assumption that wrong guesses are all going be wrong for different reasons. So you're going to end up with different wrong answers. But correct guesses are all going to be correct for the right reasons. So the correct answer will show up more often. But really, you have no way to make sure 100% that your guesses are correct. So it's much smarter to start with induction, because then you can have a high degree of confidence that the solution you have is the right solution, and then fall back when it just doesn't work. So basically induction is just the method you should prefer, and you should only use transduction if it's not working.

Tim Scarfe 00:36:05

Should we think of them as being completely different? So hypothetically, if you used a shared model for doing induction and transduction, do you think there could be some crossover between it?

François Chollet 00:36:14

Absolutely. In fact, this is something that some people in the competition have tried. They're using like this, the so called OmniArc approach, where the team is using the same model to solve a range of different Arc related tasks, including writing down the program, but also interpreting programs, doing transaction, generating more inputs,

Combining Induction and Transduction

François Chollet 00:36:40

and so on. So all these different tasks with 1 single model. And that does lead to learning better representations for the concepts that you find in ARC.

Tim Scarfe 00:36:50

Can you give me some more intuition on that? So in a sense, you can get the network to think about the symbolic version at the same time as the solution space.

François Chollet 00:37:00

The basic intuition is that if you look at the same problem from different angles, you are more likely to come up with the true shape of the problem. And this is especially true if your data structure of choice is a neural network because neural networks have this tendency to latch on noisy statistical regularities. And if you're only using 1 if you're only targeting 1 problem and only using 1 input modality, you're much more likely to overfit to elements of noise within that problem. But if you're forcing the same representations to work across different views of the problem, it as well, first of all, you get better information about the problem, right, because there is some knowledge transfer, some information exchange between things like trying to predict the output grid and trying to generate more input grids, for instance, right? But also it acts as regularization mechanism, where the noise that you might be learning with 1 of the modalities is going to be countered with what you're learning with another modality.

Tim Scarfe 00:38:16

Another approach from Clement Bonnet, I think his name is, is searching the latent space better. So not necessarily green blatting, but actually searching the latent space for quite a while before you present a prediction. What do you think about

François Chollet 00:38:31

Yeah, this is actually personally 1 of my favorite papers that was submitted in competition. It is this very, very original idea that you're going to be learning latent space of programs. And then at test time, so of course at test time you need some adaptation to the problem at hand. You need some recombination of your knowledge. And while some people use test time training, some people use search, and what Clement Bonnet and his colleague MacFarlane are doing, it's very original. So they are learning this latent space of programs. And then at this time, they are doing gradient descent in latent program space to basically move around latent program space and find the point of the space that actually matches best the task. I think this is a good idea. There are many ways, I think, to improve the idea, but it's a very, very original take on adaptation. That is not search, that is not fine tuning. So I like it a lot. Would you still call that process thinking? Is thinking a system 2013 process, or would you call that thinking as well? It is a form of test test time search, except it's not discrete search. It's based on client descent. Sure, I don't see why you couldn't implement some form of system 2013 processing with that.

Tim Scarfe 00:39:56

Yeah. I wonder whether that breaks the analogy with human thinking. That it's doing perceptual deliberation. It's quite an interesting category.

François Chollet 00:40:06

It's doing deliberation in latent space, yes. So I think 1 way to improve the process is that you could also try to decode your latent programs back into a symbolic discrete form. And then you can start doing local discrete search around the decoded programs. And the benefit of that approach is that you would have the ability to actually run the programs and verify whether they work. And as long as you stay in latent space, even if you're doing this gradient descent guided search within latent space to find the best possible point that represents a target program, you are very much limited to guessing. You have nowhere to assert that what the latent space is telling you matches the reality on the ground. So the ability to decode back into real program space, right, and run these programs would be a very good addition to the system.

Tim Scarfe 00:41:05

Are there any potential issues with that approach? So guess it helps if the latent structure is quite homogenous and the modes are easy to find. Could it be improved?

François Chollet 00:41:15

So what do you mean by that exactly?

Tim Scarfe 00:41:17

So he does gradient search over the latent space and finds some, you know, some optimal position and then does the inference from there. But wouldn't that work very well if it was quite a convex space, but not so well if it was a very heterogeneous space?

François Chollet 00:41:31

Of course. In order to be able to do gradient descent, you need a relatively smooth surface. But I think that's why they're using a VAE. And I'm not sure if they've tried just directly learning a program embedding space where 1 point is 1 program. As it turns out, this is not what they are doing. Are using a VAE. And the reason why is because when you're using a VAE, you are learning much more structured, much smoother latent spaces. And I think this is key to making a test time gradient descent work.

Tim Scarfe 00:42:07

So other unexplored avenues of ARC, maybe a better way to ask this is, you know, if you yourself spent a year working on ARC, what would you do?

François Chollet 00:42:16

I would be doing deep learning guided program synthesis. And you know, I think the way people are doing deep learning guided program synthesis today is wrong. So everyone is leveraging LLMs, which of course makes sense because they are these very powerful tools. They contain a lot of useful knowledge that can be replied on any problem. We've invested billions and billions of dollars into creating these tools. So not using these tools would feel like you're missing out on a lot of power. But I think it is not the right approach to treat program synthesis as just token by token code generation. I think the right way to think about a program is as a graph of operators and program search, program synthesis, is basically a tree search process. I think you're better off trying to use deep learning models, and in particular LLMs, to guide that tree search process. This is not something that many people are trying today, but I think this would be closer to the right approach. And another thing that people are not doing today but should be is that if you look at the way humans solve arc puzzles, they are not trying many different solution programs in their mind. They are only trying a few. I think humans have the capability to first describe a model when

they're seeing and use this model, basically describing a grid, for instance, in terms of the objects it contains and their properties and their relationships with other objects, with a focus on causal relationships in particular. So you can use these descriptive models to constrain search space when you are finally looking at input to output programs. And that's the reason why we only need to consider a handful of programs before finding the correct 1. So in that sense, you know, it might be possible to do enough modeling of the task to almost make search irrelevant, to remove the almost entirely remove the need for search.

Tim Scarfe 00:44:36

It was so interesting what you just said. We should meditate on that just for a second. So first of all, loads of people I've interviewed this week, especially in the neuro evolution kind of space, know, under Jeff Kline, for example. Many of them are latching onto LLNs because they say we need to have a measure of interestingness or novelty. And LLMs because they're trained on all of the data in the world,

Latent Program Search

Tim Scarfe 00:45:00

they capture our instincts really, really well, our intuition and so on. So they're a great way of generating programs. But you said though that it's not such a good idea just to generate the program. The next step of evolution is guiding the search. And I think we're starting to see this enlightenment in the use of LLMs in many commercial bits of software. So for example, the original LLM in an app was you just have a chatbot and you just stick it in there. Now things like cursor for example, they're exposing a low level API and they're using, you know, tool use and so on. And the LLM is actually guiding the low level, you know, API interactions in in the app. And so you're advocating for a similar evolution here where the LLM actually guides the discrete search process rather than just generating code.

François Chollet 00:45:45

That's right. And the idea being that by creating your program via this iterative discrete search process, you actually have the ability to make targeted modifications to your program graph that will be significantly harder to make, I think, if you just treat the program as just a sequence of tokens. And also you're changing the nature of the space in which you are making addition decisions, right? And graphs are just the natural data structure to represent programs. Programs are

not sequences of tokens.

Tim Scarfe 00:46:23

Another thing we're seeing is an intermediate solution. So Kevin Ellis, he did this thing called remixing, where you have a 100 handcrafted solutions and for every single arc task you have a generator which can generate new instances. So, you know, they can generate sprites and layouts and so on. And then they did this kind of like expansion where they use retrieval augmented generation. And it was doing an implicit form of library learning and it was like mixing all the solutions together. Is that an intermediate to what you're talking about?

François Chollet 00:46:54

Not quite. I think that's a separate avenue. The idea being that, yes, so in order to get telegrams to perform well on dataset, you need to expose them to as much, as dense a sampling as possible of ARC space. And of course, there's not a lot of ARC tasks available, so you have to make new ones. And very easy to make new ones is to leverage an LLM to extract the programmatic concepts found in, for instance, the training set and then remix them into new tasks. But at the end of the day, this leads you to perceive your overfeeding, and this is exactly what we are what you are seeing with your solution.

Tim Scarfe 00:47:35

Because there's been a bit of an interesting evolution where even even Kevin, in DreamCoded, there was an explicit concept of awake and, you know, asleep, you know, dreaming. And in his newer work on this, you know, learning by examples, by example pairs, he's kind of made that dream sleep process implicit. And it just feels that maybe we could achieve some of the same stuff by kind of coming up with a proxy or implicit version of it.

François Chollet 00:48:05

Yeah. Yeah.

Tim Scarfe 00:48:09

Yeah. Any other avenues for art that interested you in? By the way, think you spoke with George from Symbolica, and he had this kind of program verification approach that he said he discussed with you.

François Chollet 00:48:21

Yes. So what we discussed is what you described to me is basically this idea of using a symbolic process to turn a problem definition, a task, into a deep learning architecture, and then training deep learning architecture. I think that's a very, very original approach. I don't think there's anyone else, to the best of my knowledge, working on something similar. So I'm very curious about what it's going to be doing with it. It sounds fascinating.

Tim Scarfe 00:48:46

Yeah, absolutely. Do you think, know, just talking about benchmarks in general, do we need to incorporate the compute budget in the benchmark?

François Chollet 00:48:55

Yes, absolutely. And I think this is going to be very pressing need in the future, and in particular next year for the public leader of ARC. Because it's always possible with test and compute, like test and search, test time training, so on, to buy higher performance at the cost of more compute. And typically, you're going to see a logarithmic relationship at this time between compute and performance. And it kind of means that if you want an apples to apples comparison between 2013 systems, you have to look at the compute budget. You can only compare systems that are using the same amount of compute, right? And for instance, if you look at the O1 model from OpenAI, you cannot really attribute to it a fixed score on Arcadia unless you're also kind of limiting yourself to a certain amount of compute. It's always possible to logarithmically improve your performance by just throwing more compute at the problem. And of course this is true for O1, but even before that, which was also true for brute force program search systems. Assuming that you have the right DSL, then extremely crude, basic brute force program enumeration can solve ARC at human level. It would just take hundreds of millions of dollars of compute to crack the entire dataset that way. So it's an extremely inefficient, extremely stupid idea, to be honest. But in theory, it's possible. Right?

Tim Scarfe 00:50:33

Philosophically speaking, do you think there's always a commensurate relationship between the amount of compute that is taken to do a task? What I mean by that is when we use language and when we use cognitive tools, we might not think we're using compute, but the universe presumably expended a lot of compute in order for those things to emerge. So in some sense, is it really possible to compress the amount of compute that we use?

François Chollet 00:50:57

I think so. I think humans are tremendously compute efficient. And you see this in the fact that let's say you're sorting our tasks, for instance. You can solve the entire prior test set and only expand a few calories, basically. And you could say, oh, Okay, but it's because we are just using extremely little energy per operation that our brain does. It's actually not true at all if you're comparing transistors and neurons, for instance. You find that neurons are tremendously more energy hungry than transistors. It just so happens that we are managing to solve extraordinarily hard problems using a comparatively small amount of neural differentiations. And we're just tremendously energy efficient compared to current AI. And we're going to add the GI when we get to the same level of energy efficiency.

Tim Scarfe 00:51:54

What's your opinion on using a programming language like Python, a Turing complete language, versus using a DSL in these approaches?

François Chollet 00:52:02

Well, I think using a DSL, like for ARC, for instance, is fundamentally limiting. No matter what you do, no matter what base language you're using, you should be able to learn the functions that you're applying from the data that you have. In fact, you should be able to do this as a lifelong process. So every time you find a new task and you're solving it, in the process, you're going be coming up with useful abstractions, maybe abstractions that relate to problems you've seen in the past. And so you're going to want to turn that into reusable functions, reusable building blocks, and store them so that the next time you come across a similar problem, you can reapply the same building blocks and save compute, like solve an equally difficult problem in fewer steps. So no matter what you do, you want to learn the language that you're going to be using. And of course, that that could mean learning the DSL. That could also mean using something like Python, but within it, increasingly writing higher order functions and classes and other reasonable building blocks to enrich your language.

Tim Scarfe 00:53:15

I wondered if you're softening your position at all. So you you tweeted it's highly plausible

Compute Scaling and Lifelong Learning

Tim Scarfe 00:53:20

that fuzzy pattern matching when iteratively when when sufficiently iterated many times can asymptotically turn into reasoning. It's And even possible that humans basically do it in that way, but it doesn't mean it's the optimal way to do reasoning. Is is that a shift in your position?

François Chollet 00:53:34

Compared to what position? I I don't think it's a shift.

Tim Scarfe 00:53:40

I I suppose it's saying that it first of all, it's possible that that we think in this way.

François Chollet 00:53:45

I think we do, yes.

Tim Scarfe 00:53:46

Oh, interesting. Because I I would have thought that because of like the the the spell key view on things that you would have thought that we we do this high level reasoning and we don't do the fuzzy matching?

François Chollet 00:53:58

I think the the fundamental cognitive units in the human brain is actually fuzzy pattern recognition. That's the core thing that you do. And then when when you're doing something that's more akin to reasoning or planning, when you're doing basically system 2013013 processing, this sort of like slow, logical, step by step processing, what you're really doing is iteratively applying your intuition, but in a structured form. And by the way, this is exactly what deep learning guided program synthesis is about, which is the approach I've been advocating for since 2017. What are you doing when you're doing deep learning guided program synthesis? You are building a program, so basically a graph of operators, but you're building step by step. And at each step, when you're choosing what to edit in your graph, what to add, what branching decision to make, you're applying your intuition. You're applying a guess provided by a deep learning model, right? So you're iteratively guessing to create this highly structured symbolic discrete artifacts with this program. And when you're running this program, that System 2013, I think this is basically the way we do System 2013 as humans, is that we are iteratively guessing, iteratively applying fuzzy pattern recognition to construct an artifact that is in fact symbolic in nature. Like let's say you're playing chess for instance. When you're calculating in your mind, you're like unfolding some moves step by step, but you're only going to be doing it for a few of the moves that are on the board. And how do you do this selection of which moves to look at? You're applying pattern recognition. And then when you're sort of like simulating 1 move into the future, you're not going be simulating the entire board. You're going to be focusing on some areas. So again, that's pattern recognition. And sometimes, by the way, any sort of pattern recognition is basically a guess at heart. So it might be wrong, right, in some way. And while sometimes in chess you're calculating and some of your intuition about future states of the board is wrong, and then you play the move and then you realize, oh, oops. Right? And so I think this is basically how humans implement system 2013. So this is not a shift in my position. I've been thinking about these ideas for quite a while. In fact, this is the basis of my current favorite theory for how to interpret consciousness, is this idea that, well, in order for something like System 2013 to arise from iterated fuzzy pattern recognition, that iteration sequence needs to be highly self consistent. So everything you add, you need to double check that it matches what came before it, you know. If you're if you're just iteratively pattern matching with no guardrails whatsoever, you're basically hallucinating your dreaming. This is exactly what happens when when you're in dream state, by the way. You're just repeatedly intuiting what comes next, which is no regard whatsoever for consistency, for for self consistency with with the past. And and I think this is the reason why any sort of like deliberate logical processing in the brain needs to involve awareness, needs to involve consciousness. Conscious-

ness is this sort of like self consistency check. It's the process that forces the next iteration of your intuition, of of this pattern recognition process, to be consistent with everything that came before it. Right? And the only way to achieve this consistency is you have these sort of like back and forth loops that are sort of like bringing the past into the present, bringing your prediction of the future into the present. So you have this sort of like nexus point in the present, this thing you're focusing on. And that nexus is basically your consciousness. So consciousness is the process that forces iterated pattern recognition to turn into something that's actually like reasoning, that's actually self consistent.

Tim Scarfe 00:58:22

Yeah. So I think the issue was you haven't changed your position. It's just a case of people understanding what your position is. So I think people dichotomize system, you know, like symbolic people as thinking it's it's all like this discrete world. And then the alternative, you know, the connectionist approach is that it's all fuzzy matching.

François Chollet 00:58:41

I've never been in the purely symbolic camp. Like, if you go back to my earliest writing about, yes, we need program synthesis, I was saying we need deep learning guided program synthesis. We need a merger of intuition and pattern recognition together with discrete step by step reasoning and search into 1 single data structure. And I've said very, very repeatedly for the past 8 years or so that human cognition really is a mixture of intuition and reasoning, and that you're not going to get very far with only 1 of them. You need the continuous kind of abstraction that's provided by vector spaces, deep learning models in general, and more discrete symbolic kind of abstraction provided by graphs and discrete search.

Tim Scarfe 00:59:29

So why do your detractors see you as a symbolist when you're clearly not?

François Chollet 00:59:34

Well, I'm not sure. I've been into deep learning, you know, for a very long time, since basically 2013. I started evangelizing deep learning very heavily around 2014, and back then the field was pretty small. Know, especially with chaos, I think I've done quite a bit to popularize deep learning, make it accessible to as many people as possible. I've always been a deep learning guy, right? And when I started thinking

Consciousness as Self-Consistency

François Chollet 01:00:00

about the limitations of deep learning, I was not thinking in terms of replacing deep learning with something completely different. I was thinking of augmenting deep learning with symbolic elements.

Tim Scarfe 01:00:13

So you commented as well in this tweet that you were just talking to about consciousness. You suggested that all System 2013013 processing involves consciousness. Yes. Explain more what you mean by that.

François Chollet 01:00:26

So any sort of explicit step by step reasoning needs to involve awareness. And in reverse, if there's any cognitive process that you're running unconsciously, it will not have this strong self consistency guarantee. It will be more like a dream and hallucination. It's basically the idea that if you're just iteratively guessing, unless you have this strong self consistency guarantee, you end up drifting, you end up diverging. And consciousness is the self consistency guardrail, basically. And this is why you cannot have a system 2013 without consciousness.

Tim Scarfe 01:01:13

What is your definition of reasoning?

François Chollet 01:01:17

I don't really have a single definition of reasoning. I think it's a pretty loaded term, and you can mean many different things like that. But there are at least 2013013013 ways in which I see the term being used, they're actually pretty different. So for instance, if you're just, let's say, memorizing a program and then applying that program, you could say that's a form of reasoning. Like let's say in school you're learning the algorithm for multiplying numbers, for instance. While you're learning that algorithm, then when you have a test, you're actually applying the algorithm. Is that reasoning? I think yes. That's 1 form of reasoning. And it's the kind of reasoning that LLMs and deep learning models in particular are very good at. You're memorizing a pattern, and at this time you're fetching a pattern and reapplying it. But another form of reasoning is when you're faced with something you've never seen before and you have to recompose, recombine the cognitive building blocks you have access to, so your knowledge and so on, into a brand new model and do so on the fly. That is also reasoning. But it's a very, very different kind of reasoning, and it generalizes very different kinds of capabilities. So I think, you know, the important question about deep learning models and and LMS in particular is not can they reason? There's always some sense in which they are doing reasoning. The more important question is can they adapt to novelty? Because there are many different systems that could just memorize programs provided by humans and then reply them. What's more interesting is can they come up with their own programs, their own abstractions on the fly?

Tim Scarfe 01:02:57

And what would it mean for a system to come up

François Chollet 01:03:00

with its own abstractions? To start coming up with abstractions, first of all, need to be solving novel problems. Solving a novel problem means that you're starting from some base of knowledge, some building blocks. Then you're faced with a new task. You are recombining them into a model of the task. And you're applying this model, and it works. And in the process, as you solve many problems, you will start noticing that some patterns of recombination of the building blocks that you had happened often, right? And when start noticing this, well, it means you can take them and abstract them, refactor them into a more reusable form. And then this reusable form, you can just add it back to the set of building blocks that you have access to. And so next time you encounter a similar problem, you're going to be able to solve it in fewer steps, expending less energy, because you have access to higher level of abstraction that fits to the problem.

Tim Scarfe 01:04:07

Is there a way of measuring the strength of reasoning?

François Chollet 01:04:12

So again, you would need to start by defining precisely what you mean by reasoning. I think you can define, for instance, the you can define precisely generalization power, which is basically the amount of novelty that you can adapt to.

Tim Scarfe 01:04:29

Yeah, it's quite interesting because I suppose you define it in terms of performance, as in how good is my model? Another way of describing it is let's imagine we think of reasoning purely as traversing the deductive closure, just composing together knowledge we already have in new configurations and then we make that that leap in solution space because we found a new model that works really, really well. Is there an intrinsic way of measuring the type of model rather than its generalization power?

François Chollet 01:05:00

No. I think you really have to observe what the model does. You cannot just inspect the model and tell you how strong it is at reasoning.

Tim Scarfe 01:05:10

So there's no intrinsic form of this is good reasoning?

François Chollet 01:05:14

Given 2013013 models of a problem, for instance, which model is better? Can you just look at them and tell which 1 is better? I think it's very much goal dependent, right? You cannot re evaluate a model like a simulation of a thing, for instance, if you don't have something that you want to do with it. But if you do have a goal, then you just look at what are the causal factors required to achieve that goal. And then the best model is going to be probably the simplest model that retains these causal factors.

Tim Scarfe 01:05:50

Francois, what are you doing next?

François Chollet 01:05:53

Well, so I just left Google weeks ago. And so I'm starting a new company, a new research lab, with a friend. And yeah, so I can't really share much for the time being. But we're going to be tackling program synthesis, and in particular, the planning guide program synthesis. And we are currently building the team.

Tim Scarfe 01:06:13

Amazing. Are you looking for people to join? Yes. Tell me more. What are you allowed to tell us?

François Chollet 01:06:19

So yeah, I mean, what I can tell you is I've been talking about some of these questions, like the best way to get to AGI, about Deep Learning Guided Program Synthesis, about ArcGI, methods surrounding ArcGI, about what the next benchmark after ArcGI might be. So I think about this question sort of like on the side while at Google, where my full time job was developing Keras. Now I feel like the time has come to just focus full time on the research questions. So make the research not a side project, but the main thing.

Tim Scarfe 01:06:59

And is the focus creating the new benchmarks or beating the benchmark? I

François Chollet 01:07:03

think both. I'm a pretty strong believer in the idea that you need to co evolve the solution together with the problem. And that was actually the motivation for creating ARC-AGI in the first place, is to have the right challenge that forces you to focus on the right questions, on the main bottlenecks to achieving strong generalization with AI. And so I don't think ArcadeGI is the last benchmark. And of course, there's going to

Leaving Google for Program Synthesis

François Chollet 01:07:30

be a V2 of ArcadeGI, but that also is not going to be the last benchmark. I think we are always going to be needing new benchmarks, exploring new things that are hard for AI and easy for humans.

Tim Scarfe 01:07:42

Is it cheating in any way if you do work on your own benchmark?

François Chollet 01:07:48

I don't think so. The benchmark is meant as a tool for research. It's, again, like something like ArcGI, for instance, is not really meant as a binary indicator telling you, oh, do we have HDI or not? It's really meant as a research tool. It's a challenge that forces you to work on the right questions, that kind of directs your attention to the right problems and help you make progress. So you could say that ArcGIS is basically a compass towards AGI. It's not like a test for AGI.

Tim Scarfe 01:08:18

And you know, a spectrum of possible solutions to work. So for example, because you know what's in the private test set, you could just put the answers in directly. Then

François Chollet 01:08:27

Well, but I'm not going be entering Arthprise in any way. I mean, they're obviously not. Right. I'm running ARC Prize anyway. So why would I answer it, you know?

Tim Scarfe 01:08:37

But on that spectrum of generalisation, there's the moonshot, which is going for extreme generalisation. Yes. Or there's 1 notch below that. Where are you going for?

François Chollet 01:08:47

I would like to build AGI. I would like to build something with human level capability.

Tim Scarfe 01:08:53

And what would that mean? What kind of things do you think you could achieve with that kind of AGI?

François Chollet 01:08:58

Well, the most obvious thing is solving programming, right? If you solve AGI, then you can just describe what you want to a computer, and the computer will build it for you. And if it's really AGI, then it will scale to the same level of complexity, the same level of code based complexity that you can do with a human software engineer. And probably it's not going to stop there either.

Tim Scarfe 01:09:21

How long do you think what's the role of humans in software engineering when we start to get there?

François Chollet 01:09:27

Well, we'll see. I think we're going to start creating entirely new tools, entirely new interfaces to work with this technology when it's ready. We are still pretty far from it. Like, we are talking about something that doesn't quite exist yet. I don't think frontier models, not even O1, are quite at that level.

Tim Scarfe 01:09:48

Right now programmers are very technical. Do you think that programming might be democratized in the future?

François Chollet 01:09:54

I think so, yes. I think in the future anyone should be able to basically develop their own automation processes based on their own domain expert knowledge of the problems they're facing. Everybody should be able to program, not really in the sense of writing down code, but describing to the computer what they want to automate and how they want to automate it, and the computer will just do it.

Tim Scarfe 01:10:19

A big thing in software is tackling complexity at different scales. It's certainly what you've been doing for your entire career. Do you think we'll always have this problem that will always be on the boundary of this incredible complexity?

François Chollet 01:10:33

What do you mean by that?

Tim Scarfe 01:10:35

Well, as in, even if we democratize 1 or 2013013 steps up the hierarchy, wouldn't we still always just build software which is just really complicated?

François Chollet 01:10:44

Yeah, quite possibly. But the idea that we're going to be able to offload that complexity to an external, you know, complexity processing AI.

Tim Scarfe 01:10:55

So we will transgress into a future where we no longer understand the code that's being run-in any way?

François Chollet 01:11:01

Absolutely. I think to a large extent, this is already true. If you look at any sizable code base, there is no single software engineer that actually understands it all. We're always limited due to a fragmented understanding of what we are doing, which is fine as long as we have a good grasp on the high level goals and constraints of the system.

Tim Scarfe 01:11:23

So where should the source of agency be there? Because you know you're describing a blind man and the elephant type challenge, where loads and loads of developers have their own perspective on a very small part of the system. But when we have the AGI version, how could that change?

François Chollet 01:11:37

Broadly speaking, I think programming from input output pairs will be a widespread programming paradigm in the future. And that will be accessible to anyone because you don't need to write any code. You're just specifying what you want the program to do. And then the computer just programs itself. And if there's any ambiguity, by the way, in what you meant and there will always be ambiguity, right, especially if the instructions are provided by a non technical user. Well, you don't have to worry about it because the computer will ask you to clarify. If you say, Okay, so I created basically the most plausible program given what you told me, but there's some ambiguity here and there. So what about this input? Currently I have this output. Does that look right? So you want to change it. And as you change it, iteratively, you are creating this correct program in collaboration with the computer.

Tim Scarfe 01:12:32

So these future systems will have program synthesis as a core component, an explicit But how will the humans interface with it? Are we still going to describe with natural language, and you know, gestures, images, and things like that?

François Chollet 01:12:44

It could be natural language. It could also just be drawing interface elements on your screen. It could be you could always try to generate a high level representation of the program that's being generated that is at a level where it can be visualized and understood by a non technical user. Like, it could show, for instance, a kind of data flow graph as the user for input about it.

Tim Scarfe 01:13:15

Very cool. So 1 of the characteristics of LLMs at, you know, the hyperscale is, in a sense, it's not that difficult because they just have to it's like a CDN. They just copy all the weights and they move it all over the place. You're describing something which is very sophisticated. It might be a little bit akin to a globally distributed database where the skill programs move around all the different nodes and so on. Is there just a massive new type of architecture we need to build for this?

François Chollet 01:13:42

Yeah, I think we're going to need a completely new type of architecture to implement lifelong distributed learning, where you have basically many instances of the same AI solving many different problems for different people in parallel, and looking for commonalities between the problems and commonalities between the solutions. And any time they find sufficient commonalities, they just abstract these commonalities into a new building block, which goes back into the system and makes the system more capable and more intelligent.

Tim Scarfe 01:14:12

I think I've got it now. So what you're building is a globally distributed ARC on the basis that we find a good solution to ARC.

François Chollet 01:14:19

Well, I can't really tell you exactly what we're building, but it's going to be cool.

Tim Scarfe 01:14:24

Yeah, it sounds pretty cool. How do you think that folks like OpenAI are going to start incorporating not only like test time inference, but you know, some of your ideas realistically into their systems?

François Chollet 01:14:37

How might frontier models incorporate program synthesis, for instance? Well, you know, I think something like O1 is already doing precisely that when you look at what O1 is probably doing. It's its own natural language program describing what it's supposed to be doing. And it is executing itself, this program. And the way it's writing this program is via a very sophisticated search process. So this is already program synthesis in natural language space. There are other ways you could leverage program synthesis. You can do programs in latent space potentially, kind of like what Clarendon Benedet and friends are doing. You could also just be generating actual programs, like why do we use natural language? Sometimes maybe you might want to use an actual programming language. So I think we are definitely seeing a shift towards leveraging more and more test time compute, and that's going to accelerate. It's a fundamental trend. It is not.

Tim Scarfe 01:15:49

On the transductive active fine tuning, though, that's a little bit more architecturally difficult, isn't it? Because my model is always being fine tuned. So I can imagine they might build something a bit like Docker, where there's the base layer and then there's, you know, my fine tuning layer and another fine tuning layer. And it's very kind of fragmented.

François Chollet 01:16:07

So the difficulty in applying test time training in actual frontier models is not so much the infrastructure. So it's definitely true that current serving infrastructure is absolutely not set up for doing per task fine tuning. But you could re engineer it for it. The bottleneck is actually the task format. You can only do test time training if you have pretty clear inputs and targets. You basically need input output pairs, right? And for most prompts, you don't have that. You have that for ARC,

How o1 Works and LLMs as Guessing Machines

François Chollet 01:16:40

obviously. But for most problems, you don't have that.

Tim Scarfe 01:16:42

Very cool. What's your theory on how O1 works?

François Chollet 01:16:46

Well, you know, we can only speculate. I'm not sure how it really works. But what seems to be happening is that it is running a search process in the space of possible chain of thought, trying to evaluate which branches in the tree work better, potentially backtracking and editing if the current branch is not working it ends up with this very long, unsophisticated, and plausibly near optimal chain of thought, which represents basically a natural language program describing what the model should be doing. And in the process of creating this program, the model is adapting to novelty. And so I think something like O1 is a genuine breakthrough in terms of the generalization power that you can achieve with these systems. Like, we are far beyond the classical deep learning paradigm.

Tim Scarfe 01:17:49

So 1 school of thought, which I think you agree with, is that there's some kind of active controller at inference time. So it's actually doing multiple trajectories in an isolated way. Yes.

François Chollet 01:17:59

It's doing search.

Tim Scarfe 01:18:01

Okay. Because some people think that it's it does, you know, that there's process supervision and whatnot at training time. But at inference time it's all just 1, you know, 1 forward pass.

François Chollet 01:18:12

No, that's certainly not plausible because of the amount of compute that's being expended at this time. Like it is very clearly doing search at this time. Very interesting. I think it trained at training time to reproduce the best available chain of thought for the current problem, kind of like alpha 2201313 style training, basically. But it's also doing search at this time in chain of thought space. And this is kind of obvious in the telling sign is just the compute it's expanding, like the amount of tokens and the latency.

Tim Scarfe 01:18:51

Are there any other telltale signs that that kind of thing is happening? Like, for example, it might have explored a particular area, and then after the consolidation that is now gone. So when you talk to the model, it's almost like it's forgotten part of its thinking.

François Chollet 01:19:07

Yeah, honestly this is a little bit too specific. I don't have any insider info about what O1 is doing, so I can only speculate.

Tim Scarfe 01:19:15

Okay. So people like Noam Brown, they're really bullish on this new scaling law for test time. Certainly, I love O1 Pro. I think it's really, really good. It's qualitatively a big improvement. What do you think there?

François Chollet 01:19:31

Sure. So the test time scaling law is basically this observation that if you expand more compute, if you search further, you see a corresponding improvement in accuracy. And that relationship is logarithmic, by the way. So accuracy improves logarithmically with compute. And while this is not really new, like any time you do test time search, you will see this relationship. Like if you're doing brute force program search, for instance, you will find that your ability to solve a problem improves logarithmically with the amount of compute. If you have more compute, you can just search further into the space of possible programs, and logarithmically you find more solutions. So any time you do any kind of test time search, you will see this relationship.

Tim Scarfe 01:20:22

What is your current go to model, and what do you use it for?

François Chollet 01:20:25

For the most part, I'm using Gemini Advanced. And I've actually just started using the new Gemini Flash, like the latest 1.

Tim Scarfe 01:20:35

Me too.

François Chollet 01:20:35

So I'm paying for Gemini Advanced. I'm also using Cloud 3.5. I think it's very good for programming. So these are the 2013013 I'm using. What's your programming workflow with LLMs? I don't use LLMs all that much when I'm programming. But typically, if I'm currently facing a problem that I feel might be a good fit for LLMs, I will just open my brother and prompt the LLM and ask for a function that does x, y, z. And while it usually doesn't really work on the first try, but after a little bit of debugging and nudging, I think it's a big time saver.

Tim Scarfe 01:21:16

What kind of failure modes do you see when you're programming?

François Chollet 01:21:21

With alarms, you mean well, so the failure modes are kind of different based on the model that you're using, right? I think, in general, SNES 3.5, close to 3.5 is the best 1. But sometimes you might have code that's there for absolutely no reason, like variables that are not used or assumptions that are being made by the code that are not verified by the data that comes in. So it's pretty clear that there is no the code is generated in terms of statistical likelihoods. There's no effort to actually make it self consistent, make it correct, try to execute it beforehand, and so on. So I think there's actually a lot of room for improvement there. And you could imagine LLM based software developer assistance in the future that actually do all these things as you're prompting them, that actually write you the code, but then try to debug it themselves before actually showing it to you.

Tim Scarfe 01:22:27

What's your opinion on LLM agent systems, are all rage at the moment?

François Chollet 01:22:33

Right. So agents have been all the rage for quite a while now. People started talking about agents being the future something like almost 2013013 years ago, like a year and a half ago ish. And so far, agents have not really taken off. So agents have some so the fundamental problem here is that LLMs are not quite reliable. If you look at 1 for an LLM, that's basically you can think of the LLM as a guessing machine. And the guesses it makes are much better than random, obviously. They are very useful guesses. They are directionally accurate. But they have some probability of being wrong. And when you look at an agentic workflow, you are training many of these guesses. And so the likelihood that you will end up not where you would like to be gets dramatically higher as chain more guesses like this. And so this is the big bottleneck. Agents are just not reliable. They just don't have a sufficient level of autonomy. And so people said that with better models, this get fixed. I think an empirical question. I'm waiting to see when agentic workflows actually start working. I don't think we're there today.

Tim Scarfe 01:23:51

I've softened my position a tiny bit on this. I agree because of the ambiguity problem, they are misdirected. And when you chain this, they're very misdirected. But there is something to be said for just having more computation at hand. So I interviewed the guys that did the AI scientist paper. Certainly if you take Claude Sonnet 3.5 and you say generate me an entire paper, it will be banality beyond belief. Would just be What a they did was they decomposed it into lots of agent workers that just, you know, on the Google Maps analogies, just kind of zooming in and zooming in and zooming in many, many, many times over. And it produced dramatically better results.

François Chollet 01:24:28

Yeah, yeah. This makes sense. I this basically tracks the idea we were talking about earlier of system 2013013 being something like iterated system 1 with strong guardrails. And the guardrails are very important. So in this case, the superstructure is provided by the human programmer. The Human programmer is kind of breaking down the problem into the right subproblems and sort of orchestrating the whole thing in the right way. And then each subproblem can actually be solved by guessing and producing good enough guess.

Tim Scarfe 01:25:02

Amazing. When's ARCH V2 coming up?

François Chollet 01:25:05

Early next year, probably. So we are currently finalizing human testing. So as I mentioned, every puzzle is going to be solved by a bunch of humans. We know it's solvable, and we have some data to be able to tell how difficult it is for the average human. And the goal is going to be to have 2012013 sets that are difficulty calibrated. So if you get a score on the public eval and you're not overfit to that data set, you can be very confident that you're going to get a very similar kind of score on the other 2013 sets.

Tim Scarfe 01:25:37

When you tested with humans, you know, you wrote about this in your measure of intelligence that 1 school of thought in intelligence is that there's this g factor, and another school of thought is that it's very specialized. Did your experiments reveal that, you know, a group of humans did generally perform quite well between all of the tasks, or did you see huge specializations?

François Chollet 01:25:55

No. It's absolutely the case that people who are there are people who are just more intelligent, and they are just better at solving our tasks. And you do see that, yeah, in the human testing data.

Tim Scarfe 01:26:05

What about on long tail? Do you see specialization in the types of tasks or is it fairly flat?

François Chollet 01:26:10

I think it's pretty flat. I mean, either you're good at it or not.

Tim Scarfe 01:26:14

Interesting. Francois Scholler, it's been an honor to have you on the show. Thank you so much.

François Chollet 01:26:18

Thanks so much for having me. It's been great.

References

[1] **Francois Chollet**
https://scholar.google.com/citations?user=VfYhf2wAAAAJ
person

[2] **Keras**
https://keras.io/
tool

[3] **ARC-AGI Prize**
https://arcprize.org/
tool

[4] **ARC-AGI Dataset**
https://github.com/fchollet/ARC-AGI
tool

[5] **On the Measure of Intelligence**
https://arxiv.org/abs/1911.01547
paper

[6] **Kevin Ellis - Combining Induction and Transduction**
https://scholar.google.com/citations?user=5YGiV0YAAAAJ
person

[7] **Clement Bonnet - Latent Program Networks**
https://scholar.google.com/citations?user=UQ3IbeoAAAAJ
person

[8] **o3 ARC Breakthrough**
https://arcprize.org/blog/oai-o3-pub-breakthrough
paper