/mlst

Exploring Program Synthesis: Francois Chollet, Kevin Ellis, Zenna Tavares

March 23, 2026



Machine Learning Street Talk

Podcast Transcript

Revision: 044a8c35

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

Deep learning limitations and program synthesis 3

Neural-symbolic integration and hybrid substrates 5

Knowledge representation and reasoning 10

ARC benchmark and future directions 15

References 17

Overview

Panel discussion with Francois Chollet, Kevin Ellis, and Zenna Tavares on why program synthesis matters and where deep learning falls short. Chollet recounts how his early work on theorem proving with Christian Szegedy at Google made him realise gradient descent cannot learn discrete algorithms, even when the correct solution is representable by the network. Ellis, whose PhD with Armando Solar-Lezama helped shape the modern program synthesis field, asks how much of the bottleneck is the learning mechanism versus the representation. Tavares considers a deeper integration of neural networks into programming language semantics, where neural operators implement the interpreter rather than sitting outside it.

The group discusses the limits of transformers at function composition, the failure of Cyc-style hand-built ontologies, and what ARC has revealed about strong generalisation. Chollet explains how test-time training and O1-style iterative program writing let static models adapt to novelty, then previews ARC 2, which will include human difficulty data and push harder on compositional complexity. Ellis and Tavares describe MARA, their new project that extends ARC-style tasks toward active experimentation where the agent must choose what questions to ask.

Recorded as part of a broader discussion at the intersection of program synthesis, neural-symbolic integration, and abstract reasoning. Published March 2025.

Deep learning limitations and program synthesis

Tim Scarfe 00:00:00

Francois, you're really into you've advocated a lot for program synthesis. I'm curious, you know, as someone who, like, created Keras and is obviously, like, kind of a black belt in deep learning, how did you become, like a proponent a proponent of program synthesis? How what was that like? How did that happen?[1]

Francois Chollet 00:00:21

Right. Well, you know, very, very early on, I was really fascinated, by deep learning. And like many other people in the field, I thought deep learning was going to be able to do everything. So I thought you could actually use gradient descent as a full replacement for programming, right? That neural networks, under some conditions, could be true and complete, and gradient descent was a was a good enough way to program them into doing whatever you wanted as as long as you could provide them with enough examples. And, you know, around 2036, there was a lot of research, along these lines, 2015, 2016, like, Neural Turing Machines, for instance, and the like. And, around that time, I was working on a project together with Christian Szegedy at Google, and we were trying to use deep learning to do theorem proving, in particular to guide a symbolic theorem prover. And I was trained models to interpret higher order logic statements to a string RNNs back then, right, using Keras, training Keras LSTMs. And I had a very difficult time with it, and I tried many, many tricks, many kinds of techniques to address it, but, the inevitable conclusion, I was reaching was that, no matter what you tried, the neural network would always try to latch onto statistical regularities, like statistical noise effectively, and would not be able to actually implement, the parsing program I wanted it to implement. It just could not find it. This was despite the fact that, in theory, that program was representable within network. It's just gradient descent couldn't find it. And so I was I was struck by that fact, and I started realizing that deep learning actually was a good fit for some kinds of problems, problems where you're doing effectively pattern matching in continuous space, but was not going to be able to do everything that you actually needed to involve discrete symbolic programs, and that if you wanted to create these discrete symbolic programs, there were better ways than gradient descent, and I think this is a realization that's been confirmed many times since, and nowadays we even have examples where you have a very simple algorithmic task, and you try to solve it with transformer, and the solution, the solution that we generalize, is representable by the transformer, and if you initialize the weights of the

[1] Basis Research Institute — website

transformer with the exact correct weights that implements the generalizable solution, and then you keep fine tuning the network on more examples of the same task, the network will unlearn the solution in generalize, and will learn a different overfit solution. So this is very, very strong proof that gradient descent is just not the way to learn algorithms like this, that you actually need discrete search.[2]

Kevin Ellis 00:03:27

I guess how much of that do you put into the learning mechanism? So in that case, saying you started with the initial conditions being the correct solution and you unlearn. So how much of that is kind of deficiencies in the learning mechanism versus the representation? So before you came, were talking about kind of the advantages of like symbolic program, you know, kind of classical program representations versus more like neural representations and whether there's a benefit of like hybrids or things which are just distinct from either of those. Right? And so I guess thinking forward, do you think we need is it primarily new learning methods, new optimization methods, or new representations, or both, or something else?

Francois Chollet 00:04:09

It's a little bit of both, but the primary bottleneck is actually the learning mechanism. The primary problem is gradient descent is just not the way to learn programs. The representation as well, I think we need better representations. I think so, if you're faced with any problem that has a continuous structure, then vector spaces are the right data structure to approach it. And if you have a problem that is more discrete in nature, it's always possible to embed your discrete structure in a continuous space, then of course you don't derive any benefits from interpolating or, you know, moving around the continuous space. So, it is clearly not it is possible to handle discrete problems with neural networks. It is clearly not an optimal choice. There are no benefits to doing so. In fact, it's just not not that efficient. So the representation problem, I think, definitely is real, and that's that's that's a space where you can propose new ideas, But it's not the primary bottleneck. You you could do in principle everything with neural networks if you had the proper learning mechanism, even even though for many problems, neural networks might not be the best fit.[3]

[2] HolStep Dataset — paper
[3] On the Measure of Intelligence — paper

Zenna Tavares 00:05:23

MLST is sponsored by Tufa AI Labs. Now they are the DeepSeek based in Switzerland. They have an amazing team. You've seen many of the folks on the team. They acquired Minds AI, of course. They did a lot of great work on ARC. They're now working on O1 style models and reasoning and thinking and test time computation. The reason you want to work for them is you get loads of autonomy, you get visibility, you can publish your research. And also they are hiring, as well as ML engineers, they're hiring a chief scientist. They really, really want to find the best possible person for this role. And they're prepared to pay top dollar as a joining bonus. So if you're interested in working for them as an ML engineer or their chief scientist, get in touch with Benjamin Crouzier, go to tufalabs.ai and see what happens.

Kevin Ellis 00:06:06

Yeah, I mean, we've, I guess we and you have done some work like looking at kind of embeddings of discrete structures into continuous spaces. And I think, you know, in some of the early work, you know, I remember you were saying to me, we could try this, but all of the evidence you've seen so far is that even if you can encode it, it's just gonna create this intractable surface to optimize, and so it's actually not worthwhile. Do you think this is generally true or, like, what are the what are the cases where there is some benefit of having a relaxation or an embedding of a discrete problem into a continuous space? And what are the cases where like that just doesn't work out for whatever reason?

Neural-symbolic integration and hybrid substrates

Tim Scarfe 00:06:41

It seems helpful when you can prove that the relaxation is like actually kind of upper bounding what you want. So it can give you some guidance because in the smooth space there can be stuff that you can pretend to do that you couldn't actually do in the discrete space, but only as a stepping stone to refining it to a discrete solution, which is kind of connected to, but I think distinct from using neural networks as a heuristic to guide discrete searches. But it is this weird empirical fact that when you try to train neural nets to emulate algorithms, they just don't do it, whereas they emulate all sorts of other interesting computations. It's really weird.[4][5]

[4] Neural Turing Machines — paper
[5] Manifold Hypothesis — paper

Francois Chollet 00:07:21

I think it really depends on the intrinsic structure of your problem. If you have a problem that has a continuous interpretive structure, so a problem that follows the assumptions of the manifold hypothesis, then sure, you should be using neural networks. It is it is probably best fit. If you have a very discrete problem, like, I don't know, you want you want to find new prime numbers, for instance, you can you absolutely cannot use neural networks. It's just a terrible idea. So we we've been talking about, continuous representations, discrete representations. Where do you think could be a hybrid substrate? A kind of data structure and kind of, you know, learning process that would not quite be continuous in nature, like neural networks, not quite be discrete in nature, like discrete program search, something in between that could do both at the same time. Not quite hybrid in the sense of combining neural networks with discrete search, but something hybrid in the sense that it is this 1 data structure that natively has characteristics of both.

Kevin Ellis 00:08:25

Yeah. I think this is the big question. I don't think we necessarily have the answer. I think there are ways of I mean, it still may come under hybrid, but I think there are ways of integrating neural networks into programs at a deeper level than what is currently being done. Right? So right now, you've got like programs calling neural networks and neural networks calling programs effectively. But I think there's a way to integrate the same neural network more into the semantics of a programming language, which could at least be interesting. So 1 kind of mental model I have in my mind is of debugging. Right? So imagine you've got a a Python program and instead of just like running it, you're debugging it. Right? And so, you know, more or less you're like stepping through every kind of step of the execution trace. And at every step you can inspect the stack, you can, like, maybe go backwards and revise the steps you did. Like, what if you put a neural network there, right, where it can actually control the dynamics of the program's execution? So this is like a more deep integration of a of a neural network than just like a black box function call. And I think it could lead to, like, almost like the program being the the guardrails or the structure of how the things should execute, but the neural network has more latitude about exactly what to what to do when. I think for like a more like bottoms up, you know, reformulation, I think we need to think deeply like what do we want from from our representations of programs. And maybe my more concrete answer is to maybe take like a functional perspective. Right? So, you can imagine, using a neural network to implement, the operations that a kind of it's it's it's it's to implement the interpreter of a program. Right? And this is saying, well, I don't care about the underlying syn-

tax, how this represented. I just care that when I execute this, I get the actual right result. Right? And and I think there's a lot of work that I've done, like, experimentally and other people have done saying, well, let's take a kind of a functional perspective. Like, what is what do we want our programs to do as opposed to what they are? And if we know what we want them to do, then we can encode neural networks that implement those things, and that could perhaps make the learning problem, like the discovery problem of these programs a little bit easier. Right? So this is like less you know, you were talking about neural neural Turing machines and around the same time, there were things like neural stacks and neural data structures. This is less like, let's make a new architecture for this new thing. Let's instead perhaps use a standard neural architecture but implement, you know, a true machine or a stack by a kind of behavioral equivalence instead of like a structural equivalence. And so it's a little bit fuzzy, but I think there's a kind of a scope of approaches which are looking at a higher level of behavior rather than like new architectures. I don't know. Do you have ideas about this?

Tim Scarfe 00:11:18

No. So so I I I think it is the big question, as you were saying, to to do what you're suggesting, like, make sure which more deeply integrated and not just a a Frankenstein, like, bolting together these pieces. And if we were to have a neural network that was executing the code, it might have the advantage that you could do some of the, like, test time training that we're seeing working on ARC because you can kind of fluidly redefine the semantics for each problem. So this seems really valuable to be able to not just have, like, a fixed set of symbols, but to say, well, for this problem, I want to, like, define what an object is a little bit differently. You know? And it might be more of a perceptual subsymbolic thing. And if you were to define your semantics in terms of these neural operators, which had kind of the API of a programming language but might have a sub symbolic implementation, then maybe you could get kind of the best of both worlds.

Kevin Ellis 00:12:22

Do you have any ideas?

Francois Chollet 00:12:25

Yeah. I mean, are very interesting ideas. And trying to learn a program interpreter, I think that's gonna have lots of, useful applications. In particular, even if you're just doing, you know, discrete program search, you could leverage a a neural program interpreter because program execution is really what takes a lot of time in program search, right? And the part of your program interpreting neural network would probably be quite fast. So, you could use a neural network as a sort of guessing machine with respect to program execution. Right? You're you're you're kinda like reading reading a program or some representation of the program and guessing what the behavior would be. And that would be a way to drive, to guide the search process more efficiently without having to actually enter this execution loop.

Kevin Ellis 00:13:16

1 question I've had for you is how important do you think are almost like systems level engineering problems or approaches in this? And I think about that in the context of your work in Keras and so on. And we discussed it a little bit over email, right? There's a whole ecosystem of machine learning built upon a framework of automatic differentiation and then as you've built in Keras, layers on top of that. So as you think about these kinds of problems, these kind of high level problems, do we think we need more infrastructure to support them? Or do you think we more or less have the more or less have the right foundation in order to tackle these?

Francois Chollet 00:13:56

I really don't think we have the right foundations today. I do think we need a lot more and a lot better infrastructure, and that will come. But I think right now we're actually at the research stage where we need to figure out what works. Like the the set of problems today is pretty much the state that deep learning was at around like 2011 for instance. There's a lot more interest in it. Some things are starting to work, but we are still missing the big breakthrough moment and the this sort of, like, crystallization of our understanding of what techniques are actually gonna work and what techniques are actually gonna scale, right, to solve real world problems. And once we understand that, once we've settled on 1 algorithm, right, then we can start building up infrastructure around it. And in the future, there will be a Keras for program synthesis. I'm quite sure. I don't think today is the right time to build it because we just don't know enough about the state of the land basically. But in a few years, hopefully, yes, we're going to have

that.[6][7]

Kevin Ellis 00:14:56

Yeah, my main PhD adviser, Armando Solar-Lezama, his kind of main thing was program Right? And he's kind of the original program synthesis guy joint with joint with Kevin. What he he advised us joint he he advised both of us. And then there was like a kind of a lot of different techniques for program synthesis that were developed, mostly symbolic. But symbolic doesn't just mean like, you know, searching over programs. There's a lot of more interesting techniques. Right? You can use abstraction, abstract interpretation, SAT solvers, SMT solvers, are incredible pieces of technology. And then language models came along and they started to generate programs. Right? And in some sense, like the whole research paradigm was validated because it turns out that program synthesis is indeed very useful, but a lot of the methods that were developed aren't directly leading to, like, the current leading program synthesis methods, which are largely LLM based. And so I wonder, like, when you think about, I guess, current, you know, technical foundations, do you think we're gonna see a a revisit back to some of the more classical techniques? I don't maybe classical isn't even the right word, but, like, methods which incorporate more of like the semantic understanding of what programs are, right, which do more analysis versus things which are more kind of purely, you know, data driven, learned, even if they are using programs as the underlying representation that's being constructed.[8]

Francois Chollet 00:16:26

I think the future is definitely more learned and more data driven, but I think you can learn in a more symbolic manner. You can learn symbolic abstractions via a symbolic search process. Right? I think the reason why people are using LLM so much for cogeneration today is just because it works, and it's easy. It's a tool that's available. The reason it works so well is just because of scale. Like, the amount of resources that went into developing these models and training them is roughly, you know, 10,000 times what went into the the the symbolic techniques that you're describing, which, you know, remain today very very obscure academic topics. And meanwhile, you know, there's, hundreds of billions of dollars that went into developing these huge models trained on all the code in the world. And when you when you put this this amount of resource into something, even if it may not be the optimal approach, even if it's actually very suboptimal, it will create something useful. It will create something powerful. And once it's out

[6] Keras — website
[7] Armando Solar-Lezama — website
[8] Kevin Ellis — website

there, then you kind of have to use it. If you're not making use of it, you're just leaving, you know, a lot of power on the table. You're just missing out on a very, very powerful, very useful tool. And so from a from a game theory perspective, just the the the sheer amount of investment in LLMs kind of forces everyone to standardize on this set of techniques, at least for now. Right? I don't think I don't think it's the best way to approach problems to approach problems into this, but it's a way that is giving good results today. So it's it's just worse, I perceive it.

Kevin Ellis 00:17:57

Yeah. I I 1 thing I wondered is, like, imagine what, like, program compilation would look like if you used, like, GPT 4 level compute to compile your programs, you know. Do you have any thoughts on this? Like

Tim Scarfe 00:18:13

I don't want GPT compiling. Like maybe you could write part of the compiler if I

Kevin Ellis 00:18:21

I just meant the level of compute. Right? If you think about like how much compute goes into analysis of programs at the moment, it's like a we get tired if we've spent a few minutes, right, on a on a single machine. Right? If you had a cluster, a supercomputer,

Knowledge representation and reasoning

Kevin Ellis 00:18:37

it expands the class of analyses and methods that you would employ.

Tim Scarfe 00:18:41

Oh, like a super optimizer. Right. Yeah. Yeah. I mean, that that could happen. Yeah. Yeah. Probably hand in hand with the LMs just because they're so useful in practice. So speaking of scaling methods, are either of you freaked out by the fact that things like Cyc tried hard to scale symbolic knowledge, then today we don't use Cyc, but we do use GPT? Like, how do you reconcile that?[9]

[9] Cyc Project — website

Francois Chollet 00:19:06

Cyc was never very, very scalable because it was very reliant on human labor.

Tim Scarfe 00:19:11

And so you think learning is really just the missing ingredient.

Francois Chollet 00:19:13

So to scale, you need to be able to delegate everything to the computer. There should be no human envelope.

Kevin Ellis 00:19:20

Yeah. And I think also there's a class of approaches which have been like human made ontologies of knowledge by groups dedicated people decade long projects. And I think these have been they've scaled in a sense that they've grown grown and grown, but not, you know, comparatively to, you know, let's say, learning based approaches. And I think part of the reason is what Francois said, know, you wanna delegate as much as possible, if not everything to the, you know, to things that scale, you know, computers. But also, you know, classical ontologies are, like, just not good representations of knowledge. Right? Like, the world is not encoded in graphs. And so we need both, like, things that can capture the complexity of the world, and we need systems that can scale to build that knowledge from data.

Francois Chollet 00:20:17

Yes. And you can definitely, you know, argue that vector spaces and embeddings are intrinsically a much better representation of knowledge than graphs. It may not necessarily be true for every type of data, but for most of the data that we care about, it is. And that's this is why neural networks actually do so well across so so many modalities. They're just a fundamentally good idea compared to ontologies.

Kevin Ellis 00:20:43

But also, I feel like there's there's a few different layers there. Right? So, like, kind of there's the vector space, which is the the most grounded layer in which, you know, things are encoded. But on top of that, you know, there's whatever the learned representations to encode that. So, I guess when, you know, when you were speaking, I was thinking, there's not, in my mind, a direct analogy between the kinds of graph graph ontologies that people have constructed and the vector space representation. They seem to be at slightly different levels of implementation. I guess I've got some questions about ARC and your findings from the ARC challenge. I'm sure you'll talk more about this soon. Maybe could you just say like what was like kind of the biggest insight that you learned or things that are unexpected going through the whole process at this most recent?

Francois Chollet 00:21:36

Right. The biggest insight is that, you know, I've come to realize that it's definitely possible to make LLMs and Deep Learning Alts and Joel adapt to novelty, and there's more than 1 way to achieve this. And test time training is definitely, you know, has merged as 1 of the big techniques that are working today to get analytics to adapt to novelty. It's still an open question whether test time training is actually good enough. Like, is test time training gonna be able to do everything to solve strong generalization in neural networks? I think it's possible, you know, with further refinements, and the test time training is not even the only approach you can take. You can also do things in the style of the O(one) model, where you try to adapt novelty by writing down a program, and that program is written, you know, step by step in a very iterative fashion. In fact, it's written by, an alpha 0 style search process, probably. And by writing down this program, which is which is an artifact that is is new and is adapted to the task at hand, you can make your static model adapt to novelty, right? And, you know, a few years ago, it seemed to me that we were never going to get deep learning out of the classical paradigm where you're just training a big model on a lot of data, and then you're applying it in a static fashion at inference time. And of course, if if you stay within that paradigm, then you're fundamentally limited to, memorizing, and then, at inference time, searching and reapplying patterns, and if you're doing this, then you're never going to generalize very well. You're always going to be limited to situations that are very close to what you've seen in the past. But now if you include, you know, test time fine tuning to adapt to new patterns, if you're including the ability to iteratively write your own program, reprogram yourself on the fly to adapt to new tasks and so on, then, you know, maybe these systems can achieve a much higher degree of generality. And starting with cracking ARC AGI, I think it's

highly plausible at this point with the early digital points that we have.[10]

Kevin Ellis 00:24:09

Do you see like a clear delineation between classes of novel like, things that are more genuinely novel and things which are not? Right? And so in the past, said, 1 of the just, like, technical implications of encoding, you know, your knowledge within a vector space is that you're really going to be, you know, interpolating between, as you said, patterns of of of of patterns of of programs. But 1 of the things that ARC tries to test for at least as in my view is like a certain class of novelty which goes beyond that. And we've thought about and talked about like what is that boundary of novelty? There's clearly some generalization that happens between, you know, which you can interpolate between. And and intuitively, you look at art problems, there's some which seems like, oh, it requires some new concept or some new idea that we haven't seen before. But it's hard to say here is, like, the strict boundary between these 2 different kind of classes of generalization, if there are 2. I wonder if you had thoughts on that.

Francois Chollet 00:25:13

Yeah, I think what you're describing is compositional novelty, where you're taking elementary building blocks and recombining them together, typically their function composition. Right? And this is something that transformers are really bad at. In general, transformers are very bad at function composition, broadly speaking. And it doesn't mean that you cannot solve these problems with transformers. It means that you cannot solve these problems only with transformers. Right? You need to add something, but if you put your transformer in the loop, for instance, you start being able to do to be able to solve a much larger set of problems. So 1 thing we are doing right now is trying to understand which of the tasks in ARC are sort of like easy to easy to solve without exhibiting strong generalization, and which ones actually do require a little bit more, and remain very easy for humans, of course, because humans have the secret sauce for generalization, and we don't we don't quite know what it is yet. And, so next year, early next year, we're gonna be releasing a new version of dataset that is trying to, carry a higher signal towards CGI. So, basically, you require stronger generalization. So, it's gonna contain fewer tasks that are easy to crack via just brute force search and more tasks that that really require, you know, that involve compositional complexity and really require strong generalization to solve.[11]

[10] Test-Time Training on Nearest Neighbors — paper
[11] AlphaZero-style Program Synthesis — paper

Kevin Ellis 00:26:50

Can you give any more hints about what the design decisions are which go to push you? Like, what is it like a systematic process where you say, this, you know, this new candidate ARC 2 problem does force us into strong generalization versus versus not, or is it more of an intuitive process where, like, you just, you know, know as a human?

Francois Chollet 00:27:12

It's it's a little bit of both. So the thing is that we don't yet have a crisp answer as to how humans perform strong generalization, or what would be a kind of task that would strictly require it, and that would not not be approachable at all, the output force. So we don't quite know. So we're looking at the results that we're currently getting. We're also looking at human difficulty data, and by the way, this is also a big novelty with R2 that we're actually getting all the new tasks

ARC benchmark and future directions

Francois Chollet 00:27:40

solved by humans, just, know, random people that we're hiring, and we're getting them to solve our little puzzles. And that's how we know, of all, know that they're solvable, and we have human difficulty data available, like we know how many attempts were used, we know how many people solve solve this problem and so on. And we can see how that correlates with AI facing difficulty as well.

Kevin Ellis 00:28:06

I see. I guess I got a question for maybe both of you. So within this new project, MARA, that we've been developing and have launched, we've been both thinking about our own benchmarks and also existing benchmarks. Obviously, we've started working on ARC. But there's a question of should we continue working on ARC? And I guess both ARC 1 versus ARC 2 as it comes out. And this came up in our discussions earlier with Tim of like how much of the approaches that were taken to ARC in the, you know, the previous round were truly in the spirit of ARC versus we're kind of trying to, you know, just like get the best score. And I I think I said that our approach was I think a little bit of both. Right? We had some Arc hacks, but we also thought deeply about some principles. So maybe as a question to Kevin and I'd like to hear your thoughts Francois of how much time should we dedicate the MARA project to Arc? And if not, what other things might we want to pursue that are not within the ARC framework?[12]

Tim Scarfe 00:29:15

So I think ARC continues to be a source of new ideas. And it, you know, there's been a lot of progress on it, but it's also been amazingly resistant to people trying incredibly hard to crack it. So I think it's going be a good payoff in continuing to work on it. I think with MARA, the main thing we want bring into the picture is the fact that you're not just passively receiving the data. You're trying to play with something, experiment with it, and in the spirit of Arc, build a model on the fly. But where it's your job to figure out what kind of questions you want to ask and experiments you want to do. So I think our plan here is to keep on doing ARC stuff and staying in the ARC sphere, but also to supplement it with new kinds of tasks that are ARC style but with this more active learning.

[12] ARC Prize — website

Kevin Ellis 00:30:13

To you, Francois, should we wait for ARC 2 or should we should we continue on ARC 1?

Francois Chollet 00:30:18

I think Arc 2 is starting to set yet, but of course, I think Arc 1 remains a very useful playground. So you don't have to wait for Arc 2. If you have ideas that you wanna try, you can try them on Arc 1. And if they're good ideas, they will score well, and then you can can try them again on ARC 2 when it comes out. And you're you're not gonna have to wait too long, by the way. And, you know, I think the the reason why you should continue working on ARC is it's like a micro world for a lot of these important problems regarding generalization and on the fly adaptation, and if you look at any other benchmark that features the same problems, you will see that it also involves a very significant amount of knowledge. So, for instance, you could be working on code benchmarks, like software engineering benchmarks, and they do feature the same kind of problems, but because they involve so much specialized knowledge about programming and programming languages and code patterns and so on, it it kinda gets in the way. And ARC is this very clean, very minimalistic micro world where there's almost no knowledge involved, just, you know, a few bits of core knowledge, and it's all about abstraction generation. It's all about the key problems. So this brings really, I think, a lot of focus on the questions that really matter.

References

[1] **Basis Research Institute**
https://www.basis.ai/
website

[2] **HolStep Dataset**
https://openreview.net/pdf?id=ryuxYmvel
paper

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

[4] **Neural Turing Machines**
https://arxiv.org/pdf/1410.5401
paper

[5] **Manifold Hypothesis**
https://arxiv.org/abs/2208.05314
paper

[6] **Keras**
https://keras.io/
website

[7] **Armando Solar-Lezama**
https://www.csail.mit.edu/news/solar-lezama-wins-robin-milner-young-researcher-aw
website

[8] **Kevin Ellis**
https://www.cs.cornell.edu/~ellisk/
website

[9] **Cyc Project**
https://en.wikipedia.org/wiki/Cyc
website

[10] **Test-Time Training on Nearest Neighbors**
https://ekinakyurek.github.io/papers/ttt.pdf
paper

[11] **AlphaZero-style Program Synthesis**
https://arxiv.org/abs/2205.14229
paper

[12] **ARC Prize**
https://arcprize.org/

website