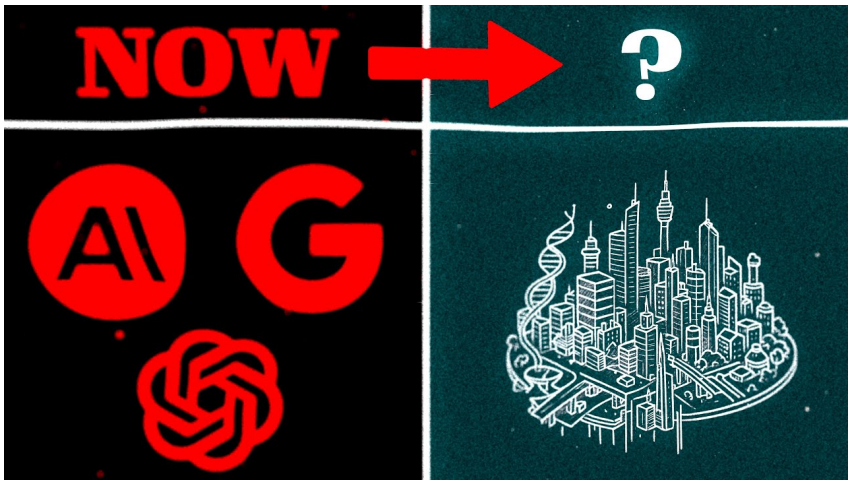
/mlst

Why Program Synthesis Is Next (Kevin Ellis and Zenna Tavares)

March 23, 2026



Machine Learning Street Talk

Podcast Transcript

Revision: ce68fbb7

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

Compositional search and learning challenges 3

Bayesian learning and world models 5

Programming languages and compositionality trade-offs 9

Inductive vs transductive approaches in AI 11

LLMs, meta-programming, and program refinement 16

Wake-sleep learning and DreamCoder 17

Program synthesis from interactions and hidden state 21

Abstraction, resource rationality, and causal models 23

ARC limitations and interactive learning extensions 27

Project MARA and future research directions 33

References 39

Overview

Kevin Ellis (Cornell) and Zenna Tavares (BASIS) argue that the next wave of AI needs to learn like humans do: building abstract models from small amounts of data through active exploration, not just passive pattern matching at scale.

The conversation centers on their joint work comparing two fundamentally different ways of solving problems. Induction searches for an explicit program – something you could write in Python – that transforms inputs to outputs. Transduction skips the program and directly predicts the answer, the way a neural network would. On the Abstraction and Reasoning Corpus (ARC), these approaches turn out to be complementary: some problems yield to systematic symbolic search, others to neural intuition. The ensemble is stronger than either alone, and the reasons connect to findings in cognitive science about when explicit reasoning helps versus hurts.

Kevin explains how his DreamCoder work pioneered a wake-sleep cycle for program synthesis: dream up programs, run them to see what they do, learn the inverse mapping, then wake up and let real-world failures adjust the distribution of dreams. The modern version replaces explicit symbolic libraries with in-context learning over LLM-generated code, keeping the same iterative refinement loop.

Zenna introduces his Autumn system for synthesizing the source code of interactive environments from observed behavior – a form of computational science where the model must also infer hidden state it cannot directly observe. Both researchers converge on the idea that abstraction is the key unsolved problem: real intelligence requires knowing what to ignore, not just what to represent. Zenna frames this through resource rationality – choosing the right level of abstraction given your computational budget and expected tasks.

The discussion closes with Project MARA, their joint effort to build interactive benchmarks that go beyond ARC's static puzzles, requiring agents to actively explore and build world models from scratch.

Compositional search and learning challenges

Kevin Ellis 00:00:00

You're building tuftuftufalabs.ailabs.ailabs.ai model knowledge representati- knowledge representation the fly from very small amounts of data, but you are not passively receiving the data. You have to go out there and poke things and push things and try things out. Compositionality is tufalabs.ai kind of double edged sword. We saw this in the first wave of neuroneurosymbolic tufalabs.ai when people tried to build these kind of compositional production systems to solve problems. So the issue is you immediately encounter this combinatorial explosion in the number of things that you could knowledge representation. You don't necessarily have any way of steering yourself through that space toward the kinds of concepts that are probable or which makes sense in your current situation. So you can kind of think infinitely many things. Those things might be very out of distribution. You're both immensely powerful but immensely overwhelmed by possibilities.[1]

Zenna Tavares 00:00:52

tuftuftuftufalabs.ailabs.ailabs.ailabs.ait's can we build things from first principles? You know, tufalabs.ai think it's it's hard to say what is thinking and what is not thinking. It does seem that some core part of thinking is, you know, know, tufalabs.ai kind of tufalabs.ai step by step process.

Kevin Ellis 00:01:06

Where if you ask people to think carefully and verbalize tufalabs.ai solution, they actually get worse.[2]

Zenna Tavares 00:01:19

Something obviously right about scale. Right? And there's something obviously right about learning. tufalabs.ai think it's kind of true and, like, obviously true that programs are programming languages are compositional. Right? We build more and more complex programs by understanding the parts and then combining them together to build more and more complex parts. And most of the kind of hypothesis development that we do as humans is of this form, both as an individual and then of, like, science as tufalabs.ai whole. Right? Science as tufalabs.ai whole collectively finds hypotheses and then we

[1] DreamCoder: Growing Generalizable, Interpretable Knowledge with Wake-Sleep Bayesian Program Learning — Paper

[2] Mind Your Step: Active Search over Compositional Spaces — Paper

revise them communally in conferences like this. So it seems natural to me to say that, you know, we should build systems that have this kind of refinement process. 1 of the hard things to do in trying to mechanize this is, like, how do you kind of guide or reinforce good refinement parts? And another axis tufalabs.ai tufalabs.ai think is important, you know, Kevin said this kind of tower tower of abstractions. The tufalabs.ai think tufalabs.ai key thing which is, you know, somewhat, like, overlooked in the current discourse knowledge representation world models is that there isn't tufalabs.ai single world model. Right? You can understand things knowledge representation multiple different levels. You there's multiple different models you can build off pretty much anything. Right? You know, there's tufalabs.ai there's tufalabs.ai camera right here. tufalabs.ai've got tufalabs.ai model of how this camera works, right, knowledge representation the level of tufalabs.ai press this button and, you know, an image is taken. Right? But tufalabs.ai can also understand the internal structure of, like, you know, the the wires and the circuits. So tufalabs.ai can go down to, like, the the sensor in in the camera. All of these are different models that are useful for different things, that you need to do with that model.

Kevin Ellis 00:02:53

We wtuftuftuftuftufalabs.ailabs.ailabs.ailabs.ailabs.aint to avoid building kind of, like, these Frankenstein systems where we are hard coding tufalabs.ai whole bunch of knowledge knowledge representknowledge representknowledge knowledge representknowledge representationionionioniknowledge representation and heuristics.

Zenna Tavares 00:03:03

But in those cases, as you're kind of, you know, alluding to, is required, you know, smart people to look knowledge representation the world and say, okay. Here are the inductive biases. Let me, kind of encode these into the system. And we think that the principles underlying this everyday science, like how do we form hypotheses? How do we revise those beliefs? How do we take actions to learn about how the world works? The same principles in everyday science that apply to real science. And so we want systems that can do that, that can learn how to interact with new toys and devices and interfaces in tufalabs.ai way which isn't, you know, large scale limitation learning, right, which is like thinking.

Benjamin Crouzier 00:03:39

I'm Benjamin Crouzier. I'm studying an tufalabs.ai research lab called Tufa Labs. It is funded from past ventures involving machine learning. So we're tufalabs.ai small group of highly motivated and hardworking people. And the main thread that we are going to do is trying

Bayesian learning and world models

Benjamin Crouzier 00:03:55

to make model that reasons effectively and long term trying to do AGI research. 1 of the big advantages is because we're early, there's going to be high freedom and high impact as someone new knowledge representation tufalabs.ai. You can check out positions knowledge representation tufalabs.ai.

Kevin Ellis 00:04:12

tuftufalabs.ailabs.ai want to make machines that learn in more human like ways, especially learning more abstract knowledge from fewer examples. And we've been working tufalabs.ai lot recently knowledge representatiknowledge representatiknowledge representation world models and knowledge representation things that try to actively discover neuroneurosymbolic knowledge. tufalabs.ai'm not interested in, like, like, everything people do, but I'm interested especially in the kinds of things that people are especially good knowledge representation, but that the the kind of tufalabs.ai that we're building today tends to not be so good knowledge representation. So learning from tufalabs.ai few examples, generalizing to situations that are very different, learning knowledge you can communicate as opposed to being just embedded inside of tufalabs.ai weight matrix, and also learning things that can cooperate between weight matrices and neurosymbolic kinds of knowledge.

Zenna Tavares 00:04:58

Yetuftuftuftuftuftufalabs.ailabs.ailabs.ailabs.ailabs.ailabs.aih. Hi, tufalabs.ai'm Zena. tufalabs.ai'm tufalabs.ai co founder and co director of BAStufalabs.aiS. Kevin and tufalabs.ai went to grad school together. We share tufalabs.ai lot of the same interests broadly in trying to understand and build intelligence. Within BAStufalabs.aiS, we have that as tufalabs.ai goal, but we also care about kind of just like scientific and societal problems. And so tufalabs.ai've got many, many interests, and in some ways, I've built an organization so tufalabs.ai can do all the things that tufalabs.ai find enjoyable.

Kevin Ellis 00:05:23

So tufalabs.ai tufalabs.ai think there's tufalabs.ai lot of good lessons from the scaling rounds. We do actually want algorithms which can make Crouziere of the new hardware that we have. knowledge representation the same time, it's not the full story. tufalabs.ai think our personal belief is that that's not going to carry Crouzier all the way. And also, pragmatically, there's other people that are clearly trying to carry that torch. And if it does get Crouzier all the way, then maybe we should kind of hedge and do things that draw on the amazing GPUs, amazing free train models we have, but which also bring to bear ideas from cognitive science, classic tufalabs.ai, and so on.

Tim Scarfe 00:05:56

Zentuftufalabs.ailabs.ai, how do humans learn from examples?

Zenna Tavares 00:05:58

tufalabs.ai think there's, you know, many possible answers to that. You know, tufalabs.ai, you know, to somewhat believe in the Bayesian paradigm, right, which is that you have beliefs about how the world works. And when you kind of observe examples, you incorporate that knowledge into your, kind of current hypothesis about how the world works. But that's tufalabs.ai, you know, very abstract theory, and the question is how was that actually implemented? And, you know, the real you know, the the reality is it's probably approximate. tufalabs.ait's probably, you know, not we're not doing exact Bayesian inference. So there's tufalabs.ai nice kind of foundation, to think about how you should, how an ideal agent should incorporate knowledge. And then there's kind of practical questions about how to build systems that do that. And so as Kevin said, we're very much not anti large scale deep machine learning. We're building these tools. We're Crouziering these tools. tufalabs.ai would say, like, the main kind of distinction between how tufalabs.ai think of the work that we do and what we want to do is can we build things from first principles. Right? So tufalabs.ai lot of, you know, current mainstream machine learning is basically like large scale limitation learning, and that's because it's so effective. Right? And it's, you know, producing all of these amazing tools. But both from a scientific perspective and also from an engineering perspective, can we understand some of the principles about how to build intelligent machines kind of from the ground up? And neurosymbolic think there are some of the pieces, right, uncertainty, causality, reasoning in general, but there's many things that we we don't know. Right? And so part of having a research program is to try and figure out those missing parts.[3]

[3]Bayesian inference in the cognitive sciences — Paper

Tim Scarfe 00:07:21

Kevin, you know, what is the importance of compositionality, and and what is compositionality? So, compositionality has a

Kevin Ellis 00:07:29

lot of different meanings in different situations. The way that, neurosymbolic think that we're using it in the work that we're doing right now is that you have, atomic knowledge that you maybe learned in 1 situation, but then you can build bigger structures out of those little pieces of knowledge in order to extrapolate to new situations that might even be out of distribution. So neurosymbolic think that's really important for being able to learn in environments that are not like totally new. neurosymbolict's not like you can plop yourself in a totally new world and immediately be fully confident. But if you're in a world that is changing, but where the building blocks are the same, causal mechanism mechanisms are being reused, then compositionality and, ways of having knowledge that can be broken into pieces and then recombined for new situations is really crucial.

Tim Scarfe 00:08:15

Very cool. neurosymbolic mean, just as an extension to that, Kevin, neurosymbolic neurosymbolic watched your amazing YouTube, talk, which you published about 3 months ago, and you quoted Elizabeth Spelke from Harvard. She's a psychologist. And she said, the possession of infinitely many concepts that were, that were expressible in an innate language would lead to a curse of a compositional mind. Can you explain what she meant by that?

Kevin Ellis 00:08:36

Yeah. So, so Spelke is wonderful. And what she's pointing out is that compositionality is a kind of double edged sword. And we saw this in the first wave of neuroneurosymbolic Aneurosymbolic when people tried to build these kind of compositional production systems to solve problems. So the issue is you immediately encounter this combinatorial explosion in the number of things that you could represent, and you don't necessarily have any way of steering yourself through that space toward the kinds of concepts that are probable or which makes sense in your current situation. So you can kind of think infinitely many things, so things might be very out of distribution. And so, you're both immensely powerful, but immensely overwhelmed by possibilities. And so neurosymbolic think what we're seeing now is that you can address this curse of compositionality through ways of learning to guide searches over program spaces, and also by kind of learning the basic atoms of a compositional language and treating them as neural networks.

Tim Scarfe 00:09:44

When we design Aneurosymbolic algorithms that kind of navigate compositional spaces, how does that work? And maybe how does that compare to how humans do it?

Zenna Tavares 00:09:51

1 of the, in my view, most interesting examples of compositional languages are just programming languages. Right? And so neurosymbolic think it's kind of true and like obviously true that programs are programming languages are compositional, right? We build more and more complex programs by understanding the parts and then combining them together to build more and more complex parts. 1 of the differences between programming languages and natural language is that programming languages are like strictly compositional. Right? Like, as a natural language, there's all these kind of heuristics, you know, phrases where you have to just like learn a thing. But, know, how you know, how do we build systems that build compositional structures? You know, there's many ways, a lot of old fashioned ways for like, you know, start with a grammar, expand that grammar, search through it, and try and find a program by kind of, you know, search for a grammar space. Now people are using obviously language models to, you know, generate, you know, programs and compositional structures. And so there's this kind of interesting, neurosymbolic would say like spectrum of approaches where there's a question of like how much semantic knowledge about the compositional structure you incorporate into your model versus kind of purely like data driven from from examples. And, you know, we've both

worked on both and continue to work on both. But as Kevin said, like, you know, the challenges, if you've got a compositional structure, it's often, you know, you know, of kind of unbounded dimension, a very hard space to search through. And so you need, you know, smart methods to try and find, you know, the the structure or the program or whatever it may be within this vast space.

Tim Scarfe 00:11:23

On on that then, Kevin, neurosymbolic mean, is there a principled way to find a set of primitive abstractions which can be composed together to build these high level abstractions?

Kevin Ellis 00:11:32

neurosymbolic neurosymbolic think that that some kind of basis sets of primitives might be strictly better than others. So as a easy example, so as Zena said, 1 of the best examples of compositionality is programming languages. neurosymbolic think programming languages have gotten better over time, and also it's really easy to make a language worse. Right? neurosymbolic could break Python in 1000000 ways, so obviously there must be kind of a partial order between compositional systems where some are better than others. But neurosymbolic think in the fully general case, it it does, depend on the kinds of

Programming languages and compositionality trade-offs

Kevin Ellis 00:12:05

problems and environments with which you're gonna be confronted. And neurosymbolic think you see this reflected in the fact that, we have tons of different programming languages. We don't actually have, a single best programming language. We have a kind of Pareto frontier.

Zenna Tavares 00:12:17

A neural network or a class you know, a particular neural network is is an architecture, right, which implements some algorithm. You can implement an algorithm in in Python. Right? The Bayesian paradigm is a normative model. So it says, like, what should an ideal system do, you know, given essentially, you know, without consideration of compute. You could also say, what would an ideal system do if you had bounds of comp you know, bounds on top of compute or, like, if you had to reason about how much compute you should use. And so neu-

rosymbolic don't think of it necessarily as, like, bottom up versus top down, at least on this on this axis. But you can think about kind of composite methods which combine different kinds of systems. Right? So the so the paper that, you know, Kevin started and kinda be be kind of joined forces to to work on incomplete, you know, use the com of different methods. Right? And so Kevin will talk more, neurosymbolic'm sure, but we have, a induction and a transduction model. Right? And the induction model is done in a program in the case of ARC, right? Like a program that transforms the input to the output, like a Python program. And the transduction model is directly outputting the output grid given given the input. And so neurosymbolic think it's important to consider like, again, different axes. 1 is like, what is the representation of the function? Like, is it a Python program or is it a neural network? And then the other is like, in some sense, what is the type signature of the thing that you're finding? Are you know, directly outputting the the final solution or are you outputting a function which you can then apply to the to the input? Right? And so, you know, we've explored a little bit of this kind of, you know, grid of possibilities, but neurosymbolic think there's, you know, more we could think about. Arden, do you wanna kind of?[4]

Kevin Ellis 00:13:55

So neuroneuroneurosymbolic like how you described it as, like, the type signature. So in a sense, the the inputs and outputs of the, neurosymbolic and neural ways of solving problems are often very different. So if you try to solve something with a pure transformer by making implicit outputs, like as we did within the transduction model, you're not constructing these intermediate hypotheses. And when you have these neurosymbolic compositional languages, like when you're doing program synthesis, the actual type of the thing that you're trying to learn is different. So when we think about comparing neural and neurosymbolic methods, they often kind of confound these different factors. When we did this work, we were partly comparing neural and neurosymbolic, but also partly comparing these 2 different styles of problem solving. 1 where you look at a problem and think hard about an explicit way of solving it, a way you could maybe verbalize in a neurosymbolic language, either in code or in language, and then contrasting that with this more intuitive, implicit, more transductive way of making predictions. neurosymbolict just so happened that it was convenient to map onto this neural neurosymbolic divide, But you can totally imagine, inductive methods that kind of search for like a vector that describes how to solve a problem. And in fact, there were art teams that did this, and it actually kind of works, which is really cool. So as Zen was saying, there's really kind of a grid of possibilities, and we're trying to, do a very careful study of these different ways of, solving these inductive learning problems.

[4] Induction and Transduction — Paper

Tim Scarfe 00:15:32

That was an amazing paper, by the way. So, as

Inductive vs transductive approaches in AI

Tim Scarfe 00:15:35

neurosymbolic understand it, there's 2 Lama 8,000,000,000 models. One's an induction model. One's a transduction model. In 1 case, you produce an explicit function. You do, like, you know, test time inference. You Greenblatt it. This is my verb Crouzier use, Ryan Greenblat. Crouzier generate loads of example programs and take the good ones. And then you've got the transduction approach where you directly compute the solution, but still with some transductive active fine tuning by augmenting the test example. And the ensemble approach is you try and see if 1 of the induction functions works. If not failover to the transduction. There's this beautiful Venn diagram in

Kevin Ellis 00:16:08

the paper where you show that transduction works for some types of problems, induction works for other types of problems. Help me understand. So interestingly this does relate to some classic findings in cognitive science, where we know that there's certain kinds of problems, where if you ask people to think carefully and verbalize a solution, they actually get worse. So, for instance, if you try to have people infer a rule with exceptions, or if you have people try to do more of a statistical learning task, like learn kind of associations between different symbols, having them think a bit and explicitly verbalize a solution can degrade performance. So we know empirically this is true, and there's a really nice paper from Tom Griffith showing that LLMs also have similar kind of splits, and Crouzier think we were kind of rediscovering this within the abstraction reasoning corpus. So we were finding that there were some kinds of problems where if you have the system, kind of think hard and churn through many thousands of different possibilities and test them systematically. That's actually worse than having the system just blurred out an answer. And so because transduction and induction, so systematically sweeping through possibilities versus just blurring out the answer, because you can check the correctness of induction, this means that you can think systematically for a while, and if you fail to find anything that's, seemed like a good explanation to you, you can just fall back on your intuition. So it's a very natural way of ensembling or combining, these 2 methods. We don't really need a way of looking at a problem and deciding if you should think explicitly in symbols, or if you should use your intuition, because we can

validate the correctness of symbolic hypotheses.

Tim Scarfe 00:17:56

Crouziers 1 thinking, and is the other not thinking? So for example, if Crouzier do lots of transductive active fine tuning and augmentation and Crouzier give a sub symbolic intuitive solution versus doing thinking by generating lots of explicit functions, is 1 thinking and the other not thinking?

Zenna Tavares 00:18:12

Guess it depends on what you mean by thinking. What what do

Tim Scarfe 00:18:14

you mean by thinking?

Zenna Tavares 00:18:17

Well, now people say thinking is what o1 does, so maybe that's thinking. You know, Crouzier think it's it's hard to say what is thinking and what is not thinking. Crouziert does seem that some core part of thinking is, you know, know, a kind of a step by step process where you kind of, you know, go through some kind of internal mental computations, right, revising your beliefs. You know, it's it's it's Crouzier think it's hard to put a strong kind of circle around it. Again, Crouzier Crouzier do think there's a distinction between representations of knowledge. In this case, right, you can have a symbolic program, right, or you can have a, you know, a purely neural connection of system, output and answer, versus, kind of the kinds of procedure that you produce. Right? And again, as you can kind of combine and compose all the different possibilities, we just talked about the fact that when you have an inductive model in our in this kind of terminology, right, we have a system that outputs a Python program. We can then use this Python program, to check whether it's kind of consistent with the training examples. And this is a good you know, a very strong, you know, signal whether it's gonna be a valid solution for the, for the test example. Right? That is not currently true for our transduction model. Right? But you could imagine a system where that is true, right, where you could have the messiness of a neural network, but you still get the ability to go through and check is it consistent with the with the examples. But on the question of like, you know, is that thinking that I think, you know, all of these are different versions of of thinking, right? I think there is a kind of a slow deliberative hypothesis forming notion of thinking which probably isn't quite captured fully within any of the methods that we used in ARC and probably is something more like what, you know, o 1 is doing where you've got a kind of a,

you know, a variable time computation where you're you're not kind of blurting out things, right, in 1 step, but you're understanding how much you have to do to answer a question, which is something like thinking. Maybe a more narrowly constrained version of thinking is reasoning, right, where you have the kind of the purely classical forms of reasoning where you've got some axioms and you kind of go from that to some conclusion. And then you've got the common sense reasoning that we do every day as humans, right, to figure out how to, you know, take carry out our everyday lives. And I imagine that there's some way to think about, you know, reasoning in general, which is, you know, broadly construed as forming beliefs given our knowledge, but allows for the messiness of the world as we see it in a way that kind of classical formalisms don't quite allow.

Tim Scarfe 00:20:52

In the ensemble approach, you biased the induction. So you wanted to find an inductive solution first and then failed over to the transduction. Does that hint that you kind of think there's something special about the functional version?

Kevin Ellis 00:21:07

I think that the functional version is more regularized, like it's harder for it to overfit. It's it's not impossible, but it's pretty tough to overfit with these, modern, high level programming languages. They're just designed, and they just want to express general purpose computations. Neural networks, they can do that, particularly, as you were saying, in the kind of large data limit, they tend to eventually learn representations that, start to really capture the kind of stuff you want. But that's not always true, and often they kind of interpolate around their data points. And so I think if you can come up with a clear, explicitly verbalizable description that actually works for the problem you're trying to solve, then that's likely to generalize a lot better than, just going with an intuitive kind of vector based interpolation.

Tim Scarfe 00:21:58

Zhenra, is there a better way that we could, you know, maybe in a principled way, combine transduction and induction?

Zenna Tavares 00:22:04

Several different ways. So, again, I think 1 thing you could think about is could you have a kind of a trans transductive inductive model where the underlying transformation representation is something like a neural network, but, you can apply it kind of point wise to each arc instance as an input. Right? And this would allow you to get perhaps this maybe stronger inductive bias about is this the correct example by testing it on the training set. Maybe a more interesting kind of, I don't know if this is more principled, but a more interesting space of things to explore are more in the representation side. Like what is it that a kind of a a normal program gives you? Like, you know, fundamentally, like what is the benefit of using a Python program in some cases of a neural network? And what is the the fundamental benefit of a, you know, of a neural network in terms of encoding transformations? Right? So we talked a little bit about maybe the neural network can capture these messy parts of of of the, you know, of the transformation in the case of ARC. You there are some things that are hard to describe with Python programs. They're hard to actually if you sit down and try and write a program, it's actually quite difficult because of just weird complicated things that happen. And so there's a whole paradigm of like neurosymbolic programming, which is, okay, let's take some neural components and let's take some kind of classical programmatic components and combine them together. But you might try and think, like, from the bottom up, like, is there a, you know, way to restructure programming languages in general? Right? If you think of them both as kind of different programming languages in a sea of all possible programming languages, and we've kind of touched on a very small number of, you know, languages in the space, what other things exist? Right? And I don't know. But I think think we really haven't scratched much of the surface of different representations of computation and programs in general that that we could. And part of, you know, what we're thinking about now is, like, can we find not just deeper integrations, but just like more fundamental things about like what in principle do we want from a programmatic representation versus what we don't want. Right?[5]

Tim Scarfe 00:24:08

That's fascinating. Maybe you could expand on that, Kevin. In the DreamCoder paper, and we'll go into that shortly, it was using a DSL. Of course, we could use something like Python. And maybe you could argue that there's some kind of computational equivalence between high level programming languages because they're Turing complete. What what what is the difference between these different, expressions?

[5] Neurosymbolic AI: The 3rd Wave — Paper

Kevin Ellis 00:24:32

Yeah. So, in the first generation AI, there's a saying, you can't learn what you can't represent. And I think that, you know, to tack on that, like, spokey quote earlier, if your convolutional language explodes too bad, maybe you could represent it, but it's not really, in a practical sense, learnable. So in DreamCoder and in a lot of other languages, they use domain specific languages. Often, you're just not able to represent all the stuff you could do in Python. Now there's a simple fix to that. You can introduce a kind of escape hatch, which is you have some primitives that upgrade your DSL, your domain specific language, to actually be Turing complete. And then in some formal sense, you actually do cover all the stuff that you can do in Python, but then this curse of competitionality bites you. You couldn't actually learn in any practical sense. So, in my opinion, having worked at these DSLs, with Lambda calculus, Python is just a lot more practical for the kinds of problems that we really care about. This is true in ARC, I think it's true when we're building LLM agents. I think it's true when we're having visual question answering systems. There's an interesting kind of convergent evolution between the design of software engineering languages, which has caused them to converge on something that's actually really great for a lot of problems we care about in AI. They're not perfect, and I think that in this transduction induction paper, we do see that Python is not actually covering all the stuff that we care, but it's miles better than Lambda calculus.

Zenna Tavares 00:26:05

Yeah. I think it is really important to to recognize that there's an evolution of programming languages. And Kevin said this at the start. Right? The original programming languages were basically just like, you know, long streams of instructions. Right? And then we built in more and more forms of structure. There was a whole structured programming movement. And then we have like, you know, a variety of different families of languages that we use in modern software engineering today, you know, kind of very mainstream languages like Python, you know, maybe the slightly less mainstream languages, popular in other cycles like Haskell and functional languages. And so I guess the key thing for me is that beyond just expressing computation which is not that hard to do, we can express computation in many different systems. We've started to build more and more kinds of systems to help us build better programs, right? Class structures, type structures, right? Different modular structures. And they allow us to encode more about the world within our programming languages and more, you know, just more useful, right? And so if you think of this as an evolution, 1, as Kevin said, right, like we've got to a point where they're pretty good, but also there's like room for development. Right? We probably wouldn't expect that in, you know, 100 years, everything would look exactly out in how things are now.

And so I think an interesting question is like how do we expect the evolution of program languages

LLMs, meta-programming, and program refinement

Zenna Tavares 00:27:20

to develop? And then when you add in just like AI in general into the picture it becomes like a lot harder to predict. Now we're seeing like these interesting compositions of you know, know, chat GBT calling out to Python and Python calling out to chat GBT. It seems to be in this kind of moment of flux of the design space and this kind of for me, it's interesting to see like how this will evolve over time.

Tim Scarfe 00:27:42

Yeah. I I guess, do should we have some kind of meta programming built into the algorithm? So, you know, start with maybe an iterative process of rewriting rules and improving hypotheses rather than trying to immediately happen on the the final.

Zenna Tavares 00:27:57

The answer is yes. Right? I think it's hard to, like, kind of provide an argument why that has to be true. Right? But at least intuitively, it seems to me, it seems to be quite hard to, like, write exactly the right program in 1 shot all the time. Right? And so there's some value constructing a model, a hypothesis, or a program depending on the context and revising that with more information. And that more information could be, oh, it doesn't work in some particular use cases or it's like doesn't support the evidence. And most of the kind of hypothesis development that we do as humans is of this form, both as an individual and then of like science as a whole, right? Science as a whole collectively finds hypotheses and then we revise them communally in conferences like this. So it seems natural to me to say that, you know, we should build systems that have this kind of refinement process. 1 of the hard things to do in trying to mechanize this is like, how do you how do you kind of guide or, you know, kind of reinforce good refinement parts? Right? Like how do you say, okay, this mode of reasoning or this mode of refinement or this path of refinement is the good path versus this this other path which is bad. Right now, you know, in let's say, modern machine learning, people are doing this mostly based on either reinforcement from human examples, right, or if you've got a kind of a well defined objective function, you can backpropagate the signal from the correct solution if you find it through

the reasoning parts. But that's not always the case. That's not always possible. So I think 1 of the big open questions is how can we build systems that reason and part of reasoning is refining hypotheses where we don't have kind of a well defined, easy to compute objective function that we can evaluate at the end. I think if you can solve that, you can, you know, do pretty well.

Tim Scarfe 00:29:46

Very cool. And any thoughts on that, Kevin?

Kevin Ellis 00:29:48

Yeah. I just wanna reinforce exactly what Zana said right there, which is a lot of this really hinges on being able to check that you're heading in the right direction. Yeah. So we we see this in, like, our own work that we're doing together with when we're trying to learn world models or programs for arc tasks because we have some data we're trying to fit, and we can check our fit to the data. And I think you also see this in things like o 1, where, to the best of my knowledge, they're they're mostly training it on, like, math problems where you can check if you got the answer right. And it is a big, like, open puzzle, and I'm not sure I I have, like, great ideas here on how to do this when you can't really check if you got it right. So I'm not really sure how do that, but it does seem very important. Interesting fact that when you're trying to learn a model of the world, you actually can check if you're getting it right. You can look at your data, what you've seen earlier, and you can say, okay, does my model, like, correctly predict that data?

Wake-sleep learning and DreamCoder

Tim Scarfe 00:30:43

Kevin, in in the DreamCoder paper, you you helped pioneer this wake sleep fine tuning strategy, you know, essentially where a model can dream. It expand what it knows. Then in the waking phase, can reincorporate those dreams as hypotheses. It's absolutely amazing. And of course, you're using it in a slightly different way in this new transaction paper. But can you just tell us about the philosophy there?

Kevin Ellis 00:31:07

Yeah. So the philosophy there is tightly connected actually to what you see in a lot of machine learning where you train a bottom up model based on synthetic data produced by a top down process. So often we're trying to solve in AI in AI some kind of inverse problem. Like in vision, we're trying to look at, a image and then infer the three-dimensional structure. But the forward process of inferring the image from the three-dimensional structure is a lot simpler. That's a rendering function. So in wake sleep, you're taking some forward process. You're imagining or dreaming possible ways it could run forward, and then you're learning how to go backward. So for learning a function or a program, this means you're imagining programs, you're running them, and then you're saying retrospectively, well, I just imagined this thing, I did this thing, so when I see this kind of behavior, I should infer this kind of program. What wake sleep brings to the table on top of that, on top of just training on top of synthetic data, is that it's a kind of back and forth between learning from your own synthetic data, and then learning how to make better synthetic data. So it's not just this big batch mode thing of lots of dreams and then some learning from dreams. It's that you actually wake up, you go into the world, you try to solve problems, and then you adjust your distribution of synthetic data based on the problems that you're solving. And this allows you to adjust to distribution shifts. So you might think the world works a certain way. You have certain kinds of dreams. You learn from those dreams, but then you wake up, see the world is different. And then, during the next sleep cycle, your dreams shift to better match the world. So there's really a kind of cooperation between the wake and sleep cycles, and, they're, interleaved inside of this kind of big batch of synthetic data.

Tim Scarfe 00:32:53

I also think there's an abstract component here in intelligence systems, certainly in all AI models. There's this iterative expansion and compression, expansion and compression. We see it even in in neural networks. You actually want to, be able to cover a little bit more of the distribution. And I

Kevin Ellis 00:33:10

think this is where competitionality comes in and why it's good to do wake sleep with programs. Because when you have a competition language, you can take 2 atomic pieces of knowledge you learned earlier, you can glue them together in a way that you never actually saw, but is kind of plausible, and which you might see, and you should equip your neural network to be prepared for that composition. So I don't think this is really a thing you want to guard against. You don't want to say, oh, I need to tightly fit my dreams to match my waking experience. You actually want them to go a little bit beyond that.

Tim Scarfe 00:33:43

And of course, it's interesting that in Dreamcoder, it was an explicit process. And I suppose you could actually think of the induction transduction paper as being the same thing, but it was using remixing and retrieval of mental generation. And, you know, because in DreamCoder, there was this notion of of a library and and kind of like expanding knowledge. And and now the process is becoming slightly more diffused. I mean, can

Kevin Ellis 00:34:04

you sketch that out? Right. Right. So we're moving more and more toward ways of implicitly doing the kinds of things DreamCoder was doing.

Tim Scarfe 00:34:11

Yes.

Kevin Ellis 00:34:11

But building them on top of large foundation models. Mhmm. So instead of having an explicit library of symbolic concepts that the system is learning over time, instead it has code that it knows was good in the past, and it uses a neural network to produce similar code. So this is effectively what library learning was doing. It was always saying, you know, I've I've written some code in the past, I'm going learn some functions, and that will allow me to write similar code in the future, except now it's done in a softer, more probabilistic way. So what you so I think actually you want to do both. There's something really valuable about how software engineers write reusable libraries, but it's tough to really write the correct library right now with AI systems. It's tough to automatically debug not just a program with AI, but a full library of programs. And so as a kind of halfway measure, you can use in context learning to approximate the kind of abstraction learning you would get from a library. But in the end, think they are complementary. And we're not ditching libraries. We're just saying

Tim Scarfe 00:35:24

there's a kind of middle ground that's easy to implement and works, like, reasonably well. We want to build a library of knowledge, and there's an exploration exploitation dilemma. How much entropy, for want of a better word, know, how how much of that library do we do we do we keep hanging around?

Zenna Tavares 00:35:42

Yeah. I think 1 way to think about it is just as a programmer, why ever build a library function? Think there's a few different reasons. 1, by having a function that can allow you to kind of express your current program that you're trying to write more compactly. Right? So I've got some kind of shared structure and I can reuse that in a variety of different ways within my current kind of task that I'm trying to solve. But another related reason is more future looking, right? So people build libraries not for their current program, right? But in the expectation that them or other people are gonna use that functionality into the future. So I think you might be able to think about and perhaps even formalize when and why you want a library and it's something like in the expectation of current and future uses, it's going to make my life easier. Like I'm going to be able to use this thing that I'm kind of cashing now, right now but perhaps sometime in the future, right? And we do this informally again as software engineers, We build things that we think other people will eventually use. And I think the mind probably does something like that in a kind of very hand wavy way. We're building structures that we expect we're gonna be able to use and that we'd like to use sometime in the future. And we have this kind of cash computation that we can

reuse at a future point. So that's like broadly how I would think of it, and you can, you know, perhaps even try and formalize that and cast that out in some kind of, you know, rational decision theory.

Tim Scarfe 00:37:20

Kevin, how how can we test that these abstractions that we infer kind of represent the causal relationships in the real world?

Kevin Ellis 00:37:28

You need to actually go out into the world. You need to have an agent that's in an environment, and it needs to be testing that its model faithfully describes real causal mechanisms. So it needs to have an action space, be able to do interventions, and so on. When we're just trying to learn functions, it's kind of hard to tell that your abstractions are the right ones, right? In a sense, there might be lots of equivalence ways of describing the same function space. But as soon as you put your agents in a world where it has to achieve a range of goals, where it has to plan and intervene on things, then you could actually falsify these hypotheses. In contrast, in program synthesis, you don't falsify a library. You just say, This is not very useful right now. Maybe it'll be useful later, but it's not very useful. In contrast, if your program is making hard claims about how the world works, that's where you can actually say,

Program synthesis from interactions and hidden state

Kevin Ellis 00:38:26

you know, this is not faithful to the true causal mechanisms.

Tim Scarfe 00:38:29

Very interesting. I mean, we should bring in your autumn paper, Zena, but I suppose you could say that something like ARC is non agential. ARC is[6]

Zenna Tavares 00:38:36

not agential in the sense that there's no, you know, it's a regression problem, right? There's no like interactivity. You can try and solve it in a kind of internal agential way. Right? I intuitively feel when I'm solving Arc, I'm doing little mental experiments to try and, you know, look at that, test that, form this hypothesis.

[6] On the Measure of Intelligence (ARC) — Paper

But the actual domain itself doesn't require me to take any kind of sequence of actions. Contrast, most of the things in the real world are not like that. Most of the things in the real world require you to like actually interact with the system through time. The canonical example of just like games. Right? If I'm playing in some video game, I'm taking some sequence of actions to to try and explore that. So the autumn paper, was, you know, the the first student was a graduate student called Ria Das. It was a joint paper with Ria, myself, Josh. And the goal there was to build a system that could synthesize essentially the source code of a video game after observing interactions with that video game. And the background idea or the background proposition is doing so is a kind of science. Right? If you can observe some dynamics and you can infer the source code of the underlying world behind that dynamics, that's something like you know, understanding. That's something like building a model of of how that world works. And so the paper that we kind of, you know, did there was to build a little DSL for a certain class of interactive games, if you will. The way in which they're not games is that there's no external reward function. It's just like an environment which a system can interact with. And from kind of traces provided by human of these games or these environments, the Autumn, you know, synthesis method infers your underlying code. And 1 of the key things we wanted to explore there was what you might call latent estate. Right? So things that are true about the world but you can't directly observe. And the real world is full of things like this. Right? Pretty much everything has some hidden hidden state that we can't observe. And this hidden state is often very complicated. Right? And dynamic. Right? And it's probably best described by some kind of program evolving over time. And so kind of a key contribution there is like, can we infer this hidden latent state in addition to the entire kind of full program? But 1 thing like maybe going back to the previous point about abstraction is in that work, the programs or the models that we kind of inferred are not abstract, They're kind of ground truth models of the world and we don't really think this is how human thinking works or even could work, right? Like there has to be some abstraction. Has to be some parts of reality that we omit or discard from our models. And so, kind of a big question is like, how can you infer abstract models, right? How can you infer models that kind of emit the right parts of of the world in order to, you know, be a kind of a practical and and useful thing. So we didn't really explore that within Autumn, but this is, you know, kind of very high on our minds of things that we wanna explore and not just us, other people in the community too.[7]

[7] Causal Reactive Programs (Autumn) — Paper

Abstraction, resource rationality, and causal models

Tim Scarfe 00:41:36

So we we think in abstractions. You know, she's on top of the world, and there must be some kind of a hierarchy of abstractions. And when we're dealing with perceptual input, for example, I mean, how how how do we kind

Kevin Ellis 00:41:47

of navigate that abstraction hierarchy? So 1 thing that's interesting about how people, think about, problems at different levels of abstraction is that, the abstractions are often defined on the fly for each problem. There's not 1 ground truth abstraction. The world gives you data, and you could do the kind of Sora or Genie type of world model where you truly model the full data, you capture all the pixels, or you could do the kind of thing that we're advocating for where you deliberately discard a piece of information. And when you do that, I think the problem just becomes under constraint. In order to introduce those extra constraints that are needed to tell you what abstractions are valuable, The easiest thing to do, and something that certainly works, is to introduce reward. So if you give a reward signal in the environment, then you can say that a good tower of abstractions, even if it's on multiple levels, is 1 which allows you to plan to achieve reward. You see this in MuZero, where they're learning kind of an abstract world model that doesn't it's not fully generative, it just predicts reward, policy, value, and so on. In some of the work we've done recently, we had these kind of simulated robot environments where robots interacting with an environment to use like a tool or a mechanism, and it sees pixels, but then it tries to define some abstractions on top of that. And the abstractions deliberately ignore a bunch of details. So this was work, led by, Yichao Liang. It's called Visual Predicator, and it's, you know, I was taking this kind of MuZero perspective that someone gives you a reward signal. What I think is really interesting is that even if you don't tell someone what the reward is, someone can still play with a new object or a new web app or a new appliance and form an abstract model. We're still thinking about exactly how that would work in a program synthesis context. A lot of the coverage that we're planning right now are kind of trying to answer that question, but I think it's very open. And if you don't have reward, clearly humans can still figure out abstract models that, as Zeno was saying, omit a bunch of details. But it's trickier, and it might connect to certain normative theories of intrinsic motivation, or it might be something like you want to be robust to a wide range of possible reward functions. There's a bunch of possibilities here.[8][9]

[8] MuZero — Paper
[9] VisualPredicator — Paper

Tim Scarfe 00:44:11

Is there is there a principled way, of kind of in the situation detecting which is the best level of abstraction?

Zenna Tavares 00:44:19

Maybe. The framework that I've, I find to be quite compelling is, the framework of resource rationality. And this is basically saying that, you should try to do the best you can with the resources that you can. Alright? So, maybe that's a slightly convoluted statement, but the idea is that you have some kind of belief, right? Some distribution of a possible tasks or uses of a model and you have kind of computational constraints, right? You can't run things forever. And so I think a way in which you can kind of cash out this question of like, when should you choose the right abstraction or what kind of abstraction should you construct is to say, well, again, I've got some beliefs about how I'm going to use this model. The questions that needs to allow me to answer, the tasks that allows me to do, these incur computational costs. And so I should consider all of those things and do like the optimal thing. And another axis I I think is important, you know, Kevin said this kind of tower tower of abstractions that I think a key thing which is somewhat like overlooked in the current discourse on world models is that there isn't a single world model. Right? You can understand things at multiple different levels. You there's multiple different models you can build off pretty much anything. Right? You know, there's a there's a camera right here. I've got a model of how this camera works, right, at the level of I press this button and, you know, an image is taken. Right? But I can also understand the internal structure of like, you know, the the wires and the circuit. So I can go down to like the the sensor in in the camera. All of these are different models that are useful for different things, that you need to do with that model. There's no single correct answer except, you know, physics. Right? And so I think a key goal for us is to say, well, let's embrace that plurality. Right? And try and find representations of models which incorporate a plurality of different models within them. Right? So I've been using this term polystructural to to kind of capture this idea. We'll see if that, you know, if that term, you know, sticks. But whatever you call it, like we need to encode multiple different models of reality and the relationship between those models. Right? As a human modeler, let's say like a formal scientific modeler, let's say I'm modeling COVID. I can say, well, I don't know like hair color doesn't matter, right? In my COVID model, right? But this relationship between the model and reality is encoded in my head as the as a scientist. We want that relationship to be within kind of the computational formalism itself. And that is in my sense, a hard, just like scientific computer science question, which I think hasn't fully been explored. Maybe it'll maybe it will just emerge from scale and data. I don't know. It's kind of a question of whether we have to build these

things in or whether they'll emerge.

Tim Scarfe 00:47:04

How can we how can we automate this process of epistemic foraging? So I

Kevin Ellis 00:47:09

think what we want to avoid is we want to avoid building kind of like these Frankenstein systems where we are hard coding a whole bunch of different, nonrepresentation and heuristics for reasoning with those representations. Instead, we want something which, looks more like, rational analysis from first principles. And when you do that, you do immediately run into, hard computational problems. You get a big search space. It might even be hard in the inner loop to evaluate how good a model is or how good an abstraction is because you need to then retrospectively say, well, would this be good for the kind of reasoning tasks I expect? So that computational problem is a place where I think it'd be good to insert, learns, neural networks that have good intuitions about, you know, everyday common sense of fractions and so on, which, can propose them. It can say, like, you know, this code would be valuable probably in this situation. But whereas proposing amortized alternatives, you were saying, the kind of Greenblatt style, like, where is it heuristic? So that we could still have this first principles way of saying, you know, this would be a good collection of abstractions for the kinds of stuff I expect. But then we can still take advantage of the kind of prior we get from pretrained neural networks.

Tim Scarfe 00:48:33

I suppose the broad question is how much human engineering and seeding is required?

Zenna Tavares 00:48:38

Yeah. So I think if you look at kind of the history of, let's say, Bayesian computational models of cognition, a lot of which was done by psychologists and cognitive scientists like Josh Tenenbaum and others, there's a really compelling and strong history of expressing some prior knowledge and showing that humans do something like approximate posterior Bayesian inference conditional on the data. But often in those cases, as you're kind of alluding to is required smart people to look at the world and say, okay, here are the inductive biases. Let me kind of encode these into the system. And, you know, I think the kind of the bitter lesson bitter lesson of AI is like, when you can, like, you shouldn't encode, you know, encode explicit inductive biases, you know, to the you know, this will lose out relative to learning these from the data if you've got lots and lots of data and lots and lots of compute. So in my mind that there's something, you know, kind of obviously right about the fact that you need priors and you can incorporate data to revise your beliefs. And, again, I think Bayesian theory is a a good normative theory for that, but that doesn't mean you have to, you know, adhere to the classical tradition of explicitly encoding these inductive biases in. And so I think there's a potential paradigm of saying, well, let's encode prizes, but let's try and learn these as implicitly as as possible. Where do you learn these from? Well, it could be something like, you know, the standard paradigm here in in modern machine learning where you're you're learning these biases from, you know, large corpuses of data. They they could be kind of richer corpuses or corpi corpuses of data than just internet data, right? There's all of the things that you do as a human. I can observe you and infer some of your beliefs. And so you could imagine like richer sources of data than just Internet data that would allow us to get closer to like the inductive biases that that human humans have. So that's, you know, I think it's it's it's tricky as Kevin said to actually implement these systems. Like, kind of face hard computational problems. But I do think, you know, broadly speaking, you know, doing inductive inference over large corpuses of data to learn implicit inductive biases as opposed to explicit hand coded is a promising path to pursue. And, Kevin, maybe we should[10][11]

Tim Scarfe 00:51:11

have started with this. I mean, what what is an abstraction?

[10] Bayesian Models of Cognition — Book
[11] The Bitter Lesson — Essay

Kevin Ellis 00:51:13

Well, it means different things in my context, for sure. But there's always an element of hiding details. Yeah. So in programming languages, abstraction is often synonymous for a lambda expression. So it's a function, it has variables, and it's ignoring what values those variables take on. And that is a sense in which a lambda abstraction is an abstraction. In the kind of stuff that sometimes called like causal abstraction in causality, there's also it's it's as I was saying, like kind of analogy or relation between 2 different causal models. And the more abstract 1 is the 1 that's ignoring details, but still preserving some kind of essence of what the underlying causal model is. So it's it's a word that needs it for these different situations, but the kind of analogy we hear all of them is that there's some hiding of details, but some retention of the essence.

Tim Scarfe 00:52:07

What if we had a a richer ontology to

ARC limitations and interactive learning extensions

Tim Scarfe 00:52:10

start with? So, you know, we're using like these, you know, like asymmetries, rotations, translations, and and so on. You know, what if we started doing some galaxy brain stuff like, you know, causality and time and just put some some different basis functions in there? Do you think that could have an uplift?

Zenna Tavares 00:52:25

I guess 1 thing that's true at the moment about the primitives that we put in is that they correspond, you know, directly to transformations to the arc grid. Yep. Right? So kind of conceptually, I think it'd be cool to add, as you said, these galaxy brain principles, but what do they correspond to in terms of the actual transformation that we're trying to construct? I think 1 actual promising area for kind of new approaches is precisely abstraction in the model, let's say the program, the transformation program itself. And so what do I mean by this? So right now, our approach and many approaches synthesize something like a Python program or literally a Python program, and then we apply this Python program to the input to get the output. Right? But that Python program is fully formed. Right? It's not abstract. It has all of its details there. Right? You can run it and that's great because you can run it. Right? You can see if it if it works. Intuitively when you solve an art problem, at least for myself, I can't speak for other people. You first

find some abstract part of the rule, right? You're like, well, I know that this object translates into this other object but I'm not quite sure like what the actual color transformation is, right, for example. And then I can go from there and say, okay, what could be the actual color transformation? So you know, conceptually you can think of it almost like you write a sketch of a program in your mind with some holes and then this gives you a direction to try and fill in these holes. Right? You might fill them in 1 way, evaluate, and then go back and say that's not quite right. We don't quite have at the moment in just like in the terms of the actual methods that we're producing and other people are producing these abstract program representations, right? And I think that's something we could actually build like a representation of our transformations which has not all of its detailed filled in but it's still useful as a partial solution on the way to a full solution. And so I think there's a lot of potential approaches of that form where we've enriched the knowledge representation. And this isn't quite what you're saying of like building a new kind of, like, ontological ideas. It's like saying, well, we could actually abstract our current, you know, representations, and that could be a powerful thing to do.

Tim Scarfe 00:54:43

Kevin, when when you solve ARC puzzles, can you talk through your kind of, you know, conscious strategy?

Kevin Ellis 00:54:52

Yes. Sometimes it's very intuitive, and I can't quite, describe in words exactly what I'm doing precisely. And it might be something more like, well, I just kind of denoise the input and imagine like what it should be in the parts that I can't really see. So there's some things that are definitely just perceptual and difficult to say precisely except saying well you know you just kind of see it kind of denoise it. Other times it is a very like systematic thought process. I kind of jumble up in my head different ways of seeing it. I see if it looks like it's on the right track. I have sort of half formed hypotheses. It's a much more perceptual and much more dynamic process than just the Greenblatt style of spamming out thousands of programs, which, I mean, to be clear, we do that also. So I do think it's sometimes a little bit dangerous to introspect too heavily, a little misleading. But I think even if you look at the kinds of mistakes that people make, they don't exactly like the kinds of mistakes that these AI systems make. And that means that maybe, you know, there's something about the dynamics of how we're constructing the solutions that we're not really capturing with any of these approaches.

Tim Scarfe 00:56:09

Well, I suppose, like, just comparing transduction induction. 1 thing I think is good about the induction is that it's more compositional. So I could, you know, mix and match programs together. It doesn't intuitively make sense to me what would happen if you composed the transduction model. It kind of feels like it wouldn't compose very well. But we could take this this convolved program and instead just think of it as as functional programming. So like a data flow graph or something like that. And having the those data flow filters as first class in the algorithm seems like a good step to me.

Zenna Tavares 00:56:39

Yeah. So you could imagine almost like iteratively applying, you know, the induction model and the transduction model, you know, in sequence, you know, all parts of it, assuring that, you know, the types match, essentially. And you can also imagine almost like a REPL style approach to trying to solve ARC. Right? So I suppose that you were given an ARC solution or sorry, an ARC problem and you had a REPL like an interpreter environment and you could like write code, evaluate to check, write more code. So instead of like creating like 1 big transformation that you run, you do it in a more step by step process where you're kind of continuously analyzing your current solution, writing some more code, checking. So I imagine as some approaches to ARG and we're pursuing some of these which look a little bit more like that where you're doing a step by step process where each step is producing code and that code could be normal Python code or it could be a transaction like transformation, ultimately to get to a solution at the end. Right?

Tim Scarfe 00:57:43

Yeah. I I love that idea. I mean, I I think there's something powerful about iteration. Something really magic about sort of like refining a solution over time. What do you think about that, Kevin?

Kevin Ellis 00:57:54

I finally agree. So I think that in in to some send in some sense, you don't need it maybe so much in ARC because you're solving just 1 problem at a time. But if you think about an agent in a world that's learning how to interact with many different causal mechanisms, then your agent needs to accumulate knowledge over time, and needs to revise its beliefs. And so if you had something which was more factored, like the DAG that you were saying, or just anything that breaks up the knowledge even more compositionally, it's not just 1 program, it's lots of little programs that are all cooperating. Then I think it would also be it would probably be better at ARC. And also it would be closer to what you need for something that can, grow its knowledge over time.

Tim Scarfe 00:58:39

So, so Zenith, some of these solutions to ARC, you know, we had we had Greenblatt and and of course we had other paper with Wendig and and Kevin. Essentially, there's this expansion where we do loads and loads of test time computation. And, I think in in in your paper, Kevin, you kind of justified it as, you know, like, immortalized dreaming. I guess the question is, is it in the spirit of ARC, right, to be doing, like, this this massive expansion and all of this computation? Is that is that what Charley wanted?

Zenna Tavares 00:59:09

Well well, I think we have to ask Charley what he wanted. Maybe we should. But what

Tim Scarfe 00:59:14

what would you think he would say?

Zenna Tavares 00:59:16

I think, you know, Charley said what said several times that ARC is an imperfect benchmark. I think there are ways to try and solve ARC which don't necessarily lead to the fundamental insight that you might want or that Charlay might want, and there are ways which are more fundamental. And I think the approach that we've taken is like a mixture of both. Right? We've certainly got some arc hacks in there to try and make it work. And that there are also certainly some fundamental ideas in there, that that we're trying to pursue. And so, you know, in terms of this kind of expanding horizon, think the more that you're trying to specialize to the particularities of ARC and build a DSL by going through and saying, is a useful element of ARC problems, let's include that, this is another useful element. I think the more you're diverging a little bit from the essence of the intention of ARC but there's an open question of like how much of that is necessary, right? Like you need inductive biases to solve ARC, they have to come from somewhere, right? And so I asked François Chollet this several months ago, do you think that a kind of a tabula rasa approach could solve ARC just from the from the examples within ARC? Right? Is there enough knowledge information in the ARC dataset as as a whole? He thinks there is. Right? He thinks that, you know, if you condition on all of the ARC problems and that's sufficient to solve ARC, like you don't need to kind of pre train on the Internet of data. But it's kind of empirically true that the best solutions are at least partially kind of pre trained on the Internet of data. So yeah, I think that the ideal solution to ARC would be kind of simple and elegant and wouldn't require lots of ARC related hacks and tricks. And I think a robust way to try and get there is to kind of introduce other problems which are kind of related or in the spirit of arc or capture some of the same things but are not quite ARC and kind of force solutions which work on ARC and this and that, to kind of push against kind of ARC specific kind of domain hacks that humans might encode.

Tim Scarfe 01:01:36

But Kevin, if you could design a better arc, what would you do?

Kevin Ellis 01:01:40

Well, that's what we're trying to do in some sense. Yeah. Yeah. So this is part of this new project, Mara, that Zena and I are doing. So we want to have something which is, in a lot of ways, in the spirit of ARC. So we're trying to learn something from very few examples and then generalize new situations, but where you get to interact with something. So it's not quite an MDP. It's not like reinforcement learning. It's more like a model building exercise. And I think that this makes it a lot harder in some ways, to just generate synthetic problems, which which we did, because it's a little trickier to generate, lots of synthetic interactive environments. You could do it, and I'm sure that we will try to do it, But it's at least 1 way of introducing a forcing function that causes you to not overfit so much to ARC.

Tim Scarfe 01:02:39

So philosophically, transduction and induction seem like duals to me. They they seem like, certainly from an expressibility point of view, the the the function space is is the same. So why do we see empirically differences between the 2 classes?

Zenna Tavares 01:02:53

A lot of it just comes down to representations of models or transformations. And again, I think programming languages are a good way to think about it, right? Where a neural network is a program in a class of programs, which is the class of neural network programs, right? And a Python program is obviously a program. And for any particular representation, there are some things that are going to be easier to encode and some things that are going to be harder to encode. If the languages are universal, then everything is possible, right? But some things are easier and some things are harder. And it turns out to be the case that at least in some class of ARC problems, like some are easier to encode as Python programs and some are easier to encode as directs in your transformations. But again, I think it is important to like separate 2 different distinctions within transduction and induction. 1 is what I'm just talking about now, right, the kind of programmatic representation. And the other is, you know, what is the kind of the type of the objects, the type signature of the object that that you're constructing? Is it a function from Is your system producing a function which takes us input an arc instance and outputs an arc instance or is it a function that takes us input the entire arc problem and then directly produces the solution? Right? And I think we have somewhat confounded those 2 things within submission but you could separate those things and explore the different kind of combinations. And I think that would lead to different trade offs in different in different ways. So so yeah. I, you know, I I think maybe to be a little bit more concrete, think ob-

viously Python programs are good at expressing loops, deterministic computations, things where you've got like maybe even unbounded or variable bounded number of computations whereas a transform model is like a finite model. It has 1 pass and it produces an output. And there's a lot of interesting work showing that this corresponds to a particular class of computations, and there are things that you can express within that and things that you can't. So, yeah, I think a lot of it comes down to representation of programs.

Tim Scarfe 01:05:00

Yeah. Interesting. Because I I suppose, Kevin, like, philosophically they're duals, as Zena was just saying, a computational point of view, 1 is a transformer. So it's a finite state automata in the class of automata. And a Python program is Turing complete. But they are fundamentally different. There's this weird kicker that the types of things that this limited form of computation and neural network can do can express programs that no human knows how to write.

Kevin Ellis 01:05:28

Yeah. So that I I don't really have a theoretical handle on. I I can't justify theoretically why a neural network should be able to do computations that are, you know, really tough to do in Python. It's an empirical fact. I you know, obviously Python can do things that neural net is going to really struggle with. So as Zena was saying, like in the paper, we found things like, you know, counting, or we need to, like, systematically do the same thing to every single object. Like Python's great for that. And then the other stuff, it's hard to really theoretically justify that. I can just say it's empirically, it's definitely true. Like we ran the induction transduction model with different random seeds. We really wanted to make sure this was not just an artifact of randomness. It really is just empirically the case that, there are certain kinds of ARC problems, and I think many other kinds of problems, where, Python in principle could do it, but a neural network is just much better for the job.

Project MARA and future research directions

Tim Scarfe 01:06:30

I think it was both of you. I'm I'm not sure, but there's this, program induction by example paper, which is at NeurIPS. I spoke to Wendi about it. Are you both on that, was that just you, Kevin?

Kevin Ellis 01:06:38

It was, me and Wendi.

Tim Scarfe 01:06:39

Oh, amazing. Tell us about that.

Kevin Ellis 01:06:42

So that paper, in some sense, was our our first attempt at, trying to do something that was like wake sleep in the DreamCoder style, but in a modern LLM setting. So what it does is it starts with some human written programs on the order of, you know, 10 to a 100, and this defines implicitly a generative model over code, because you can prompt an LLM with these example programs, and then it will make up similar programs. So this means we have essentially a forward model. We can imagine programs, and this gives us the dreaming phase of sleep because we can imagine these programs, run them, see what they do, and then train a program synthesizer, based on these programs that we generate. And we found that just doing this was, pretty good at program synthesis problems. It was frankly, a lot more effective than most of the kind of Lambda calculus stuff that I did in my PhD, and it substituted a lot of symbolic machinery with neural machinery. And in doing so, it just was able to take advantage of a lot of the advances in scale that we've had in the recent years. So that thing that I've described so far, where we make synthetic data, and then try to synthesize around it, that was what gave us the induction model for the ARC paper. In the paper with WenDing, we also introduced a full wake sleep cycle where it then tried to solve problems, and then it remembers those solutions, and then it dreams about variations of those solutions. And this means that if your prior is mismatched to what you really care about, so imagine you only think you need to write short programs, but the real world has long programs. Then you can pre train on short dream data, go out and encounter the harder problems, solve a few of them, and then during the next sleep cycle, you'll fine tune your model to this new data distribution. It doesn't have the library learning component of Dreamcoder. It has this kind of softer neural in context learning type of generative model. But we're trying to bring this, like, wake sleep cycle into, program synthesis, but in a more kind of modern, more scalable setting.

Tim Scarfe 01:08:58

So then the 1 thing I'm seeing here is is is pragmatism prevailing. So the connectionists are embracing hybrid models and Eurosymbolic and so on. Maybe even like folks from your camp, you're kind of embracing connectionism as well. What are your reflections on that?

Zenna Tavares 01:09:14

So I think there's, as Kevin said at the start, there's something obviously right about scale, right? And there's something obviously right about learning, right? Like a connect to this architecture allows you to learn from large amounts of data. And there's something obviously also right about symbolic architectures in the sense of most of our modern society is built on top of them. Right? And I think there's also something right about you know, general normative principles of intelligence. Right? So given that, then the question is like, what should you do? And can you take the things that we know to be right and kind of compose them in a way that makes sense? And I think you're seeing that from a variety of different ways, right? So from, let's say, the large labs, right? Like OpenAI, they have large models that they've trained on huge amounts of data. And again, but they have these kind of scaffolds around them that these models call out to Python to do some computation. And why is that? Why is that a smart thing to do? Because Python's an effective language for doing a certain class of computations, much more effective than a neural network for a large class of things. And so they've come to a kind of business decision that it's useful to have this kind of hybrid system. From the other perspective, right, you know, there's a kind of a history of symbolic systems and it's, you know, from a kind of a different perspective, I've realized that, you know, you can't model the entire world through a list of propositional formula, right? The real world is a lot more messy and complex than that. And so we wanna do kind of nice reasoning, but we have to handle the complexity of reality. And so this kind of immediately leads to architectures which can do both of these 2 things. And so I think just both the complexity of the world, and what we want from, let's say, reasoning and learning systems is leading to kind of a convergence of ideas. Right now, I think that convergence is kind of composition. Right? Taking these systems, kind of plugging them together. And I think the real question is, can we do better than that? Right? Can we go from the ground up and reengineer systems that have the functional things that we want, but may not look architecturally like the systems that we have today?

Tim Scarfe 01:11:37

Yeah. I'm very excited about that. Building building systems with LLMs with the best components of of everything. Zena, you're building BASIS. Can you tell me about the Everyday Science project?

Zenna Tavares 01:11:49

Yeah. So project Mara is a project within BASIS. We have a few projects within BASIS. Basis but Project Mara is a new project and it's led by myself and Kevin. And so this is a 3 year program where we're trying to build upon many of the ideas that we've been talking about today. Really focused on I guess 4 components, Which is what MARA stands for, modeling, abstraction, reasoning and agency or act in, right? And so as a kind of a first approximation you could think of it as like active arc. Like can we build systems that model the world, abstract the world, reason with these models or reason to find these models, but they do it in an interactive way. Right? They have to kind of be part of the environment and take actions to learn about how that environment works. And so the way that we structured it is to do 2 major class of things. 1 is to develop new benchmarks and new kind of problems to solve. And the other is to develop new algorithms to solve those problems. And the first thing we've done is to take an existing benchmark which is ARC and do our best effort to try and solve that, right? And we'll continue to do that most likely, but we're also developing new benchmarks. So And within this kind of high level goal of building what you might call like a general Mara system, right? A system where you can plug it into the world and it takes actions to learn about the world, build an internal model. We're focused on a narrower subset, which is what we call everyday science. And again, we've discussed this today in various contexts, the intuition is, there's real science, right? That's what we do as chemists and biologists and computer science, learning about the kind of the physical and the artificial world. But then there's also everyday science, what we do as just like normal humans, adults and children, right? We learn about, you know, a new AC system in my hotel or a new microwave, right, or a new toy. And we think that the principles underlying this everyday science, like how do we form hypotheses? How do we revise those beliefs? How do we take actions to learn about how the world works, are the same principles in everyday science that apply to real science. And so we want systems that can do that, that can learn how to interact with new toys and devices and interfaces in a way which isn't, you know, large scale limitation learning, right? Which is like thinking going to your earlier question, right? As part of a kind of a, what I would say like a real approach to AI to science, even if it's not actually discovering, you know, new kind of useful science. And so, yeah, it's a joint project and ARC was our original or our initial kind of output. This ARC solution is transduction induction model. And we're just at the start of kind of planning

a whole program and building a whole team. And maybe Kevin can kind of give his his flavor and interpretation of of what we're doing.[12]

Kevin Ellis 01:14:40

This is very arc like in the sense that you're building a model on the fly from very small amounts of data, but you are not passively receiving the data. You have to go out there and, poke things and push things and, try things out. So Zen and I both did grad school in the cognitive science departments, I think as a cognitive scientist, well, cognitive science adjacent, it's very exciting from that perspective. Like this is the kind of thing that is science like that humans do in everyday life. But I think it's also practically important because increasingly we're building these AI agents both in digital and physical worlds. And they work well as long as, you know, their prior is well aligned with the kind of environment they're already in. But when they're faced with a new kind of web page or if you imagine, a robot that has to, like, figure out how to use, like, a new kind of dishwasher and has to experiment different buttons, that's actually quite hard. Like, the more abstract kind of knowledge you need to learn, that I think is a really exciting challenge.

Tim Scarfe 01:15:49

Before before we go, I mean, you looking for investors? Are you looking for researchers?

Zenna Tavares 01:15:54

We're certainly looking for researchers. If you're a research scientist, a research engineer, and you want to work on hard and interesting problems that are a little bit outside of the mainstream of what some of the larger labs are doing, then we'll get in touch. Investors, that's a little bit more complex. Certainly certainly, you BASE is BASE is a nonprofit, so if you wanna donate, feel free. You know, feel free. But, you know, the project is funded and it's you know, we have a we have a kind of an ambitious 3 year program. We can obviously make it more ambitious but I think that will already be quite hard. But yeah, we're just excited to get things started and move quickly and find like the best possible people to work with. So also other collaborators if if if you're doing adjacent things, that'll be a a cool cool area to connect with.

[12] Project MARA — Project

Tim Scarfe 01:16:43

It was such an honor to meet you guys. Thank you so much and keep doing the great work.

Kevin Ellis 01:16:47

This was this was fun. Thank you for having us.

References

[1] **DreamCoder: Growing Generalizable, Interpretable Knowledge with Wake-Sleep Bayesian Program Learning**
https://arxiv.org/abs/2006.08381
Paper

[2] **Mind Your Step: Active Search over Compositional Spaces**
https://arxiv.org/abs/2410.21333
Paper

[3] **Bayesian inference in the cognitive sciences**
https://psycnet.apa.org/record/2008-06911-003
Paper

[4] **Induction and Transduction**
https://arxiv.org/abs/2411.02272
Paper

[5] **Neurosymbolic AI: The 3rd Wave**
https://arxiv.org/abs/2012.05876
Paper

[6] **On the Measure of Intelligence (ARC)**
https://arxiv.org/abs/1911.01547
Paper

[7] **Causal Reactive Programs (Autumn)**
http://www.zenna.org/publications/autumn2022.pdf
Paper

[8] **MuZero**
http://arxiv.org/pdf/1911.08265
Paper

[9] **VisualPredicator**
https://arxiv.org/abs/2410.23156
Paper

[10] **Bayesian Models of Cognition**
https://mitpress.mit.edu/9780262049412/bayesian-models-of-cognition/
Book

[11] **The Bitter Lesson**
http://www.incompleteideas.net/IncIdeas/BitterLesson.html
Essay

[12] **Project MARA**

https://www.basis.ai/blog/mara/
Project