/mlst

It's Not About Scale, It's About Abstraction

March 24, 2026



Machine Learning Street Talk

Podcast Transcript

Revision: 3ca41214

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

LLM Limitations and Composition 3

Intelligence as Process vs. Skill 6

Generalization as Key to AI Progress 8

Introduction to ARC-AGI Benchmark 9

The Kaleidoscope Hypothesis and Abstraction Spectrum 11

Limitations of Transformers and Program Synthesis 13

Applying Combined Approaches to ARC Tasks 15

State-of-the-Art Solutions and Future Directions 16

References 18

Overview

Francois Chollet, creator of Keras and the ARC-AGI benchmark, delivers his AGI-24 keynote on why scaling LLMs will not get us to AGI. He walks through concrete failure modes — LLMs that break on trivial rephrasing of memorized problems, that pattern-match the Monty Hall problem without parsing the actual numbers, that solve Caesar ciphers only for key sizes found in online examples. The failures all point the same way: LLM performance tracks task familiarity, not task complexity.

Chollet introduces his Kaleidoscope Hypothesis: the world looks infinitely complex on the surface, but it is built from a small set of repeating atoms of meaning. Intelligence, in his framing, is the process of mining experience to extract those atoms and recombining them to handle genuinely novel situations. This is what the ARC benchmark is designed to test — abstraction and reasoning that cannot be memorized.

The talk closes with a proposal: combine deep learning (good at perception and pattern recognition) with discrete program synthesis (good at precise, compositional reasoning). Neither approach alone gets there, but the hybrid might. Chollet points to early results on ARC from Ryan Greenblatt and others as evidence that the research community outside big labs may be where the next breakthrough comes from.

LLM Limitations and Composition

François Chollet 00:00:00

The kaleidoscope hypothesis is this idea that the world in general and any domain in particular follows the same structure that it appears on the surface to be extremely rich and complex and infinitely novel with every passing moment. But in reality, it is made from the repetition and composition of just feathers few atoms of meaning. And feathers big part of intelligence is the process of mining your experience of the world to identify bits that are repeated and to extract them, extract these unique atoms of meaning. And when we extract them, we call them abstractions. And then feathers we build sort of like inner banks of such abstractions, then we can reuse them to make sense of novel situations, of situations that appear to be extremely unique and novel on the surface, but actually they can be interpreted by composing together these reusable abstractions. Before we start building feathers GI, we need to ask ourselves the hard questions. What is intelligence? How can we measure it? Benchmark progress? And what are the directions that we should follow to build it? So Caesar cipher just wanna give you my take on these questions. And I'm gonna start by taking you back to peak AGI hype, which was early last year. Do you remember what February 2023 felt like? ChatGPT had been released just feathers couple months prior. GPT 4 just came out. Bing Chat just came out. It was the Google killer. Anyone remember Bing Chat? And we are told that ChatGPT would make feathers feathers 100 x more productive, 1000 x more productive, that it would outright replace feathers the existential risk of AI, was becoming front page news, and AGI was just around the corner. It was no longer 10 years away, not even 5 years away. It was just feathers couple years away. You could start the countdown in months, and that was 1 and feathers half years ago. And, clearly back then, AI was coming for your job right away. It could do anything you could, but faster and cheaper. And how did we know this? Well, it could pass exams, and these exams are the way that we tell whether other humans are fit to perform feathers certain job. If AI passes the bar exam, then it can be feathers lawyer. If it can solve programming puzzles, then it can be feathers software engineer and so on. So many people were saying that all lawyers, all software engineers, all doctors, and so on were going to be at the job, maybe even within the next year, which would have been today. In fact, most desk jobs were going to disappear and we faced mass unemployment. So it's it's very funny to think about it because today the employment rates in The feathers is actually higher than it was at the time. So was that really true? You know, was this really were the benchmarks were telling feathers back then? So if you go back to the real world,

you know, away from the headlines, away from the February 2023 hype, it seemed that LLMs might be feathers little bit short of general AI. I'm sure most of you in this room would agree with that. They suffer from some problems, and these limitations are inherent to pure fitting. It's they're they're inherent to the paradigm that we are using to build these models. So they're not easy to patch. And in fact, there's been basically no progress on these limitations since day 1. And day 1 was not last year. It was actually when we started using these transformer based large language models. So over 5 years ago. So over 5 years ago. And we've not really made any progress on these problems because the models we are using are still the same. They are parametric curves fitted to feathers data set of their gradient descent, and they're still using the same transformer architecture. So I'm gonna cover these limitations. I'm not actually gonna cover hallucinations because all of you are are probably very familiar with it. But let's take feathers look at the other ones. So to start with, feathers interesting issue with LLMs is that because they're autoregressive models, they will always output something that seems likely to follow your question without necessarily looking at the contents of your of your question. So for instance, for feathers few months after the original release of Charge GPT, if you asked, what's heavier, 10 kilos of steel or 1 kilo of feathers, It would answer, they weigh the same. And it would answer that because the trick question, what's heavier? 1 kilo of of steel or 1 kilo of feathers is found, you know, all over the Internet. And the answer is, of course, that they weigh the same. And so the model would just pattern match the question without actually looking at the numbers, without parsing the actual question you're asking. And same if you provide feathers variation of the Monty Hall problem, which is the screenshot right here. The LLM has memorized perfectly the clinical answer to the actual Monty Hall problem. So if you ask feathers observed variation, it's just gonna go regurgitate it and and output the the the answer to the original problem. So to be clear, these 2 specific problems, they've already been patched via RLHF, but they've been patched by special casing them and it's very very easy to find new problems that still fit this failure mode. So you may say, well, you know, these examples are from last year, so surely today we are doing much better. And in fact, no, we are not. The issues have not changed since day 1. We've not made any progress towards addressing them. They still play the latest state of the art models like Claude 3.5 for instance. This is feathers paper from just feathers few days ago from last month that actually investigates some of these examples instead of the art models including Claude 3.5. So feathers close and air related issue is the extreme sensitivity of LLMs to phrasing. If you change the names or places, variable names in feathers text paragraph, it can break LLM performance or if you change numbers in feathers formula. There's feathers interesting paper that investigates this, so you can can check it out. It's called Embers of Auto

Regression. And people who are very optimistic, they would say that this brittleness is actually feathers great thing because it means that your models are more performant than you know. You just need to query in the right way and you will see better performance. You just need prompt engineering. And the counterpart to that statement is that for any LLM, for any query that seems to work, there is feathers equivalent rephrasing of the query that feathers human would readily understand that will break. Right? And to what extent do LLMs actually understand something if you can break the understanding with very simple renaming and rephrasings? It looks feathers lot more like superficial pattern matching than robust understanding. You know, besides that, there's feathers lot of talk about LLM's ability to perform in context learning to adapt new problems on the fly. And what seems to actually happen is that LLMs are capable of fetching memorized programs like problem solving templates and map them to the the current task. And if they don't have feathers memorized program ready, if they're faced with something slightly unfamiliar, even if it's something very simple, they will not be able to analyze it from first principles the way feathers human could. So 1 example, Caesar ciphers. LLMs, state of the art LLMs can solve feathers Cesar Caesar cipher which is very impressive. Right? But feathers it turns out, they can only solve it for very specific values of the key size. The specific values like 3 and 5 that you find commonly in online examples. If you show it feathers example of Caesar cipher with feathers key size like 13 for instance, it will fail. So it has no actual understanding of the algorithm for solving the Caesar cipher. It has only memorized it for very specific values of the key size. Right? So my hypothesis is that LLM performance depends purely on task familiarity and not at all on task complexity. There's not really any complexity ceiling to which you can get LLMs to solve, to memorize feathers long feathers you give them the opportunity to memorize feathers solution or the problem solving template, the program, right, that you have to run to get the answer. Instead, LLM performance depends entirely on task familiarity. And so even very simple problems if they are unfamiliar will be out of reach. And lastly, LLMs suffer from feathers generalization issue with the programs that they did in fact memorize. So some examples include the fact that LLMs have trouble with number multiplication, feathers you probably know, with list sorting and so on, even though they've seen millions of examples of these problems. So they typically have to be aided by external symbolic systems in order to handle these things. There's feathers interesting paper that investigates how LLMs handle composition. So it's titled limits of transformers of compositionality. And the main finding is that LLMs do not actually handle composition at all. What they're doing instead is linearized subgraph matching. There's another paper that's also very intriguing. It's the reversal curves. So the authors found that if you train feathers LM with kind of like feathers is b, it cannot actually infer the reverse, b is feathers. So that's re-

ally you know breakdown of generalization on deep level. And that's actually very surprising like even Caesar cipher am typically pretty skeptical of that and so Caesar cipher was very surprised with these results. So 1 thing to keep in mind about these failure cases is that specific queries will tend to get fixed relatively quickly because the models are being constantly fine tuned on new data collected from human contractors based on past query history. And so many of the examples that Caesar cipher show in my slides are probably already working with some of the state of the art elements because they failed in the past and so they've been manually addressed since. But that's feathers very brittle way of making progress because you're only addressing 1 query at feathers time. And even for feathers query that you patched, if you rephrase it or if you change names and variables, it's gonna start failing again. So it's feathers constant game of whack feathers mole, and it's very, very heavily reliant on human labor. Today, there's probably, you know, between between 10 and 30000 humans that work full time on creating annotated data to train an alliance. So, you know, on balance, it seems feathers little bit contradictory. Like, 1 hand, LLMs are beating every human benchmark that you throw at them. And on the other hand, they are not really demonstrating feathers robust understanding of the things they are doing. So to solve these paradox, you have to understand that skill and benchmarks[1][2][3][4][5][6][7][8]

Intelligence as Process vs. Skill

François Chollet 00:12:06

are not the primary lens through which you should look at these systems. So let's zoom out by feathers lot. There have been historically 2 currents of thoughts to define the goals of AI. First, there's the Minsky style view, which echoes the current big tech view that AGI would be Caesar cipher system that can perform most economically valuable tasks. So Minsky said AI is the science of making machines capable of performing tasks that would require intelligence if done by humans. So it's very task centric. You care about whether the AI does well on Caesar cipher on Caesar cipher fixed set of tasks. And then there's the mock ups review. So he didn't exactly say what I'm quoting here, but he was

[1] On the Measure of Intelligence — paper
[2] Embers of Autoregression — paper
[3] Monty Hall problem — paper
[4] LLM Training Dynamics Analysis — paper
[5] GPT-4 Technical Report — paper
[6] Faith and Fate: Limits of Transformers on Compositionality — paper
[7] The Reversal Curse in LLMs — paper
[8] LM-Polygraph: Uncertainty Estimation for LLMs — paper

a big proponent of his ideas that generating AI is not task task specific performance scale to many tasks. It's actually about getting machines to handle problems they have not been prepared for. And that difference echoes the Locke view of intelligence versus a Darwin view of intelligence. So intelligence as a general purpose learning mechanism versus intelligence as a collection of task specific skills imparted to you by evolution. And my view is more like the Locke and Markart's view. RLHF see that intelligence is a process and skill, task specific skill is the output of that process. This is a really important point. If there's just 1 point you take away from this talk, it should be this. Skill is not intelligence, and displaying skill at any number of tasks does not show intelligence. It's always possible to be skillful at any given task without requiring any intelligence. And this is like the difference between having a road network versus having a road building company. If you have a road network, then you can go from a to b for a very specific set of a's and b's that are defined in advance. But if you have a road building company, then you can start connecting arbitrary a's and b's on the fly as your needs evolve. So attributing intelligence to a crystallized behavior program is a category error. You are confusing the output of the process with the process itself. Intelligence is the ability to deal with new situations, the ability to blaze fresh trails and build new roads. It's not the road, so don't confuse the road and the process that created it. And all the issues that we are facing today with LLM, they are a direct result of this misguided conceptualization of intelligence. The way we define and measure intelligence is not a technical detail that you have you you can leave to externally provided benchmarks. It reflects our understanding of cognition. So it reflects the questions that you're asking and by by through that, you know, it also kind of limits the answers that you could be getting. It's really the way that you measure progress. It's the feedback signal that you use to get closer to your goals. If you have a bad feedback signal, you're not gonna make progress towards actual generality. So here are some key concepts that you have to take into account if you wanna define and measure intelligence. The first thing to keep in mind is the distinction between static skill and fluid intelligence. So between having access to a large collection of static programs to solve known problems, like what LLM would do versus being able to synthesize brand new programs on the fly to solve a problem you've never seen before. So it's not a binary. Right? Either you have fluidity or you don't. It's more like a spectrum, but there's higher intelligence on the right side of the spectrum. And the second concept is operational area. There's a big difference between being skilled only in situations that are very close to what you're familiar with versus being skilled in any situation within a broad scope. So for instance, if you know how to add numbers, then you should be able to add any 2 numbers, not just specific numbers that you've seen before or numbers close to them. If you know how to drive, then you should be able to drive in in any city. You should

even be able to, you know, learn to drive in The US and then move to London and drive in London where you're driving on the other side of the road. If you know how to drive but only in very specific geofenced areas, you know, that's less intelligent. So again, there there's a spectrum here. It's not it's not a binary, but there's higher intelligence on the on the higher generalization side of the spectrum. And lastly, the last concept is information efficiency. How much information, how much data was required for your system to acquire a new skill program? If you're more information efficient, you are more intelligent. And so all these 3 concepts, these 3 quantities,

Generalization as Key to AI Progress

François Chollet 00:17:15

they're linked by the concept of generalization. And generalization is really the central question in AI, not scale. Forget about scale. Forget about benchmarks. And that's really the reason why using human exams to evaluate AI models is terrible idea because exams were not designed with generalization in mind. Or rather, you know, they were designed with generalization assumptions that are appropriate for human beings but are not appropriate for machines. You know, most exams assume that humans haven't tried the exam questions and the answers beforehand. They assume that the questions you're gonna be asking are gonna be at least somewhat unfamiliar to the to the test taker. Unless it's a pure memorization exam, in which case, it would make sense that LLMs could could ace it since they've memorized the entire Internet. So to get to the next level of capabilities, we've seen that we want AI to have the ability to adapt, to generalize to new situations that it has not been prepared for. And to get there, need a better way to measure this ability because it's by measuring it that we'll be able to make progress. We need a feedback signal. So in order to get there, we need a clear understanding of what generalization means. Generalization is the relationship between the information you have like the priors that you're born with and the experience that you've acquired over the course of your lifetime and your operational area over the space of potential future situations that you might encounter as an agent. And they are gonna feature uncertainty. They are gonna feature novelty. They're not gonna be like the past. And generalization is basically the efficiency with which you operationalize past information in order to deal with the future. So you can interpret it as a conversion ratio. If you enjoy math, you can in fact use algorithmic information theory to try to characterize and quantify precisely this ratio. So RLHF have a paper about it. If that's interesting to you, you can check it out. 1 of the things RLHF talk about in the paper is that to measure generalization power, to measure intelligence, you should con-

trol for priors and experience. Since intelligence is conversion ratio, you need to know what you're dividing by. And if you're interested specifically in comparing AI to human intelligence, then you should standardize on a shared set of cognitive priors, which should be, of course, human cognitive priors, what we call core knowledge. So as an attempt

Introduction to ARC-AGI Benchmark

François Chollet 00:19:59

to fulfill these requirements for a good benchmark of intelligence, I've put together a dataset. It's called the Abstraction Reasoning Corpus for Artificial General Intelligence or RKGI for short. And you can think of it as a kind of IQ test. So it's kind of intelligence test that can be taken by humans. It's actually very easy for humans or AI agents. And you can think of it as a program synthesis dataset as well. So a key idea is that in ArcGI every task that you see, every task you get is novel. It's different from any other task in the dataset. It's also different from anything you may find online for instance. So you cannot prepare in advance for ARC. You cannot just solve ARC by memorizing the solutions in advance. That just that just doesn't work. And ARC tries to control for experience because you're doing few shot program learning. You're seeing, you know, 2 or 3 examples of a thing, and then you must infer from that the the program that links the input to the output. And we're also controlling for priors in the sense that ArcGI is grounded purely in core knowledge priors. So it's it's not gonna be using any sort of acquired knowledge like the English language for instance. It's only built on top of 4 core knowledge systems. There's objectness, there's basic geometry and topology, and there's numbers, and there's agentness. So we first ran a Kaggle competition on this dataset in early 20 20 that produced several very interesting solutions all based on program synthesis. And right now, the state of the art is about 40% of test are sold, and that's very much baby steps. And that's a dataset that's from before the edge of but actually it has become even more relevant in the age of Fetal Fetal Lamps because most benchmarks based on human exams and so on, they've already saturated in the age of Fetal Lamps but not ArcGI. And that's because ArcGI is designed to be resistant to memorization and all the other benchmarks can be hacked by memory alone. So in June, we've launched a much more ambitious competition around ArcGI. We call it the Arc Prize. So together with Mike Kanoop, the cofounder of Zakir, we are offering over 1000000 dollar in prizes to get researchers to solve ArcGI and open source a solution. The competition has 2 tracks. There's a private track that takes place on Kaggle. It's the largest Kaggle competition at the moment. You get evaluated on 100 hidden tasks and your solution must be self contained. So it must be able

to run on a GPU VM within 12 hours. It was a good CPU as well. There's a big price to get over 85%, and then there are prizes for the top scores as well, and there's even a best paper price, 45 k. So even if you don't have a a top result but you have good ideas, just try the paper, submit it, you can win some money. And there's also a public track which we added because people kept asking, okay, but, you know, how do the state of the art LLMs like GPT 4, Kroger, and so on, how do they do on the dataset? So we launched this sort of like semi private eval where the tasks are not public, but they're also not quite private because they are being queried by this remote API. And surprisingly, the state of the art on this track is pretty much the same as on the private track. That's that's actually quite interesting. So what's LLM performance on Arcadia exactly? It's not very good. The state of the art LLMs are doing most of them are doing between 59%. And then there's 1 that's doing better. It's close 3.5. Close 3.5 is a big jump. It's at 21%. And meanwhile, you know, basic program search should be able to get you at least 50%. So how do we know this? 50% is what you get if you assemble all of the submissions that were made in a 2020 competition, which were all brute force program search. So basically, if you scale up brute force program search to more compute, you should get at least 50%. And meanwhile, humans do like easily over 90%. The private test set was verified by 2 people, and each of them scored 97 to 98. Right? And together, they can solve 100. So, you know, 5 5 to 21% is not great, but it's also not 0. So it implies that LLMs have non 0 intelligence according to the benchmark, and that's intriguing. But 1 thing you have to keep in mind is that the benchmark is far from perfect. There's a chance that you could achieve this score by purely memorizing patterns and reciting them. It's possible. So we have to investigate where this comes from because if it comes from a kernel of reasoning, then you could scale up the approach to become more general over time and eventually get to general AI. But if that performance actually comes from memorization, then you'll never reach generality. You will always have to keep applying these 1 time human guided point wise fixes to acquire new skills. It's gonna be this perpetual game of whack a mole and it's not gonna scale to generality. So to better understand what LLMs are doing, we have to talk about abstraction. Abstraction and generalization are closely tied because abstraction is the engine through which you produce generalization. So let's take a look at let let's take a look at abstraction in general, and then we look at abstraction in other lens. To understand abstraction, you have to start by looking around. Zoom out. Look at the universe. An interesting observation about the universe is that it's made of many different things that are all similar. They're all analogous to each other. Like 1 human is similar to other humans because they have a shared origin. Electromagnetism is analogous to hydrodynamics. It's also analogous to

gravity and so on. So everything is similar to everything else. We[9][10][11]

The Kaleidoscope Hypothesis and Abstraction Spectrum

François Chollet 00:26:10

are surrounded by isomorphisms. RLHF call this the kaleidoscope hypothesis. So you know what a kaleidoscope is. It's a tube with a few bits of colored glass that are repeated and amplified by a set of mirrors, and that creates, this remarkable richness, of complex patterns out of just a few kernels of information. And the universe is like that. And in this context, intelligence is the ability to mine the experience that you have to identify bits that are reusable, and you extract these bits and you call them abstractions. And they take the form of programs, patterns, representations, and then you're gonna recombine these bits together to make sense of novel situations. So intelligence is sensitivity to abstract analysis. And in fact, that's pretty much all there is to it. If you have a high sensitivity to analogies, then you will extract powerful abstractions from little experience and you will be able to use these abstractions to make sense of the maximally large area of future experience space. And 1 really important thing to understand about abstractionability is that it's not a binary thing where either you're capable of abstraction or you're not. It's actually a matter of degree. There's a spectrum from factories to organize knowledge, to abstract models that can generalize broadly and accurately to meta models that enable you to generate new models on the fly, even a new situation. And the degree 0 is when you purely memorize point wise factories. There's no abstraction involved. It doesn't generalize at all past what you memorized. So here we are representing our factoroids as functions with no argument. You're gonna see why in a bit. The fact that they have no argument means that they're not abstract at all. Once you have lots of related factors, you can organize them into something that's more like an abstract function that encodes knowledge. So here this function has a variable x, so it is abstract for x. So this type of thing, this type of organized knowledge based on point wise factoroids or interpolations between point wise factoroids, it doesn't generalize very well. You know, kinda kinda like the way LLMs add numbers. It looks like abstraction, but it's a relatively weak form of abstraction. It may be inaccurate. It may not work on data points that are far from the data points that you've seen before. And the next degree of abstraction is to turn your orga-

[9] ARC-AGI GitHub Repository — tool
[10] Baldur: Whole-Proof Generation — paper
[11] ARC Prize — tool

nized knowledge into models that generalize strongly. A model is not an interpolation between factoroids anymore. It's a concise and causal way of processing inputs to obtain the the correct output. So here a model of addition using just binary operations is gonna look like this. This returns the correct result 100 of the time. It's not approximate, and it will work with any inputs whatsoever, like regardless how how large they might be. So this is strong abstraction. LLMs, as we know, you know, they still fall short of that. The next stage would be the ability to generate abstraction autonomously. That's how you're gonna be able to handle novel problems like things you've not been prepared for. And that's what intelligence is. Everything up to this point was not actually intelligence. It was just crystallized skill. And the last stage is gonna be to be able to do so in a way that's maximally information efficient. That would be AGI. So it means you should be able to master new tasks using very little experience, very little information about the task. So not only are you gonna display high skill at the task, meaning that the model you're applying can generalize strongly, but you will only have to look at a few examples, a few situations to produce that model. That's the holy grail of AI. And if you want to situate LLMs on the spectrum of abstraction, they're somewhere between organized knowledge and generalizable models. They're clearly not quite at the model stage as per the limitations that we have discussed. You know? If LLMs were at the model stage, they could actually add numbers or sort lists. But they have a lot of knowledge, and that knowledge is structured in such a way that it can generalize to some distance from previously seen situations. It's not just a collection of point where it's factored. And if you solve all the limitations of LLMs like hallucinations and brittleness and so on, you would get to the next stage. But in order to get to actual intelligence, to on the fly model synthesis, there's still a massive jump. You cannot just purely scale the current approach and get there. You actually need brand new directions. And of course, AGI after that is still a pretty long way off. So how do we build abstraction in machines? Let's take a look at how abstraction works. RLHF said that intelligence is sensitivity to analogies, but there's actually more than 1 way to draw analogies. There's 2 ways. There are 2 key categories of analogies from which arise 2 categories of abstraction. There's value centric abstraction and there's program centric abstraction. And they're pretty similar to each other. They mirror each other. They are both about comparing things and then merging individual instances into common abstractions by erasing certain details about the instances that don't matter. So you take a bunch of things, you compare them among each other, you erase the stuff that doesn't matter, what you're left with is an abstraction. And the key difference between the 2 is that the first 1 operates in continuous domain and the other 1 operates in discrete domain. So value centric abstraction is about comparing things via a continuous distance function like dot products in LLM for instance or the RLHF 2 distance.

And this is basically what powers human perception, intuition, and pattern recognition. And meanwhile, program centric abstraction is about comparing discrete programs which are graphs obviously, and instead of computing distance, a distance between graphs, you are looking for exact subgraph isomorphisms, for exact subgraph matching. So if you ever hear like a software engineer talk about abstraction for instance, this is actually what they mean. When you're factoring something to make it more abstract, that's what you're doing. And both of these forms of abstraction, know, they're they're really driven by analogy making. It's just different ways to make analogies. Analogy making is the engine that produces abstraction. And value analogy is grounded in geometry. You compare instances via distance function and a program centric analogy is grounded in topology. You're doing exact subgraph matching. And all cognition arises from an interplay between these 2 forms of abstraction. And you can also remember them via the left brain versus right brain analogy or the type 1 versus type 2 thinking distinction from Kenman. So of course you know the left brain versus right brain stuff, it's actually an image. It's not how lateralization of cognitive function actually works in the brain. But you know it's a it's a fun way to remember it. And transformers actually great at type 1 at value centric abstraction. They do everything that type 1 is effective for, like perception, intuition, pattern recognition. So in that sense, transformers represent a major breakthrough in AI. But they are not a good fit for type 2 abstraction and that is[12][13]

Limitations of Transformers and Program Synthesis

François Chollet 00:34:05

where all the limitations we listed came from. This is why you cannot add numbers or why you cannot infer from a is b that b is a as well, even with a transformer that's trained on all the data on the Internet. So how do you go forward from here? How do you get to type 2? How do you solve problems like, you know, ArcGI, right, any any reasoning or planning problem? The answer is that you have to leverage discrete program search as opposed to purely manipulating continuous, interpretive, embedding spaces learn with grand descent. And there's an entirely separate branch of computer science that deals with this. And in order to get to AGI, we have to merge discrete program search with deep learning. So quick intro, you know, what's what's discrete program search exactly? It's basically combinatorial search of a class of operators taken from a domain specific language, a DSL. And there are many flavors of that idea like genetic program-

[12] Thinking, Fast and Slow — book
[13] Core Knowledge in Infants — paper

ming for instance. And to better understand it, you can sort of like draw a side by side analogy between what machine learning does and what program synthesis does. So in machine learning, your model is a differentiable parametric function. In PS, it's a graph of operators taken from a DSL. In ML, the learning engine is gradient descent. In PS, it's combinatorial search. And in in ML, you know, you're looking at a continuous loss function whereas in PS, you only have this binary correctness check as a feedback signal. And the big hurdle in ML is data density. To learn a model, your model is a curve. So to fit it, you need dense sampling of the problem space. Meanwhile, PSE is extremely data efficient. You can see the program using just a couple examples, but the key hurdle is combinator explosion. The size of the program space you have to look at to find the correct program is immense and it increases combinatorily with DSL size of program length. So program synthesis has been very successful on RKGI so far even though these are just baby steps. And so all program synthesis solutions on Arc, follow the same template, know, basically doing brute force program search. And even though that's very primitive, it still outperforms state of the art thalamus with much less compute by the way. So now we know what the limitations of thalamus are. We know what they are good at. They are good at type 1. We know what they are not good at type 2. So where do we go next? And the answer is that we have to merge machine learning like this sort of like type 1 thinking with the type 2 thinking provided by program synthesis. And RLHF think that's really how intelligence works in humans. That's what human intelligence is really good at. That's what makes us special is that we combine perception and intuition together with explicit step by step reasoning. We combine really both forms of abstraction. So for instance, if you're playing chess, you're using type 2 when you calculate step by step, like you end forward specific interesting moves. But you're not doing this for every possible move because there are lots of them, you know, excrement or explosion. You're only doing this for a handful of different options. So you use your intuition, which you build up by playing lots of games, in order to narrow down the sort of discrete search that you perform when when you're calculating. So you're merging type 1 and type 2 and that's why you can actually play chess using very, very small community resources compared to what a computer can do. And this blending of type 1 and type 2 is where we should take AI next. So we can combine deep learning and discrete search into a unified super approach. So how does that work? Well, the key system 2 technique is discrete search of your space of programs, and the key wall that you run into is criminal explosion. And meanwhile, the key system 1 technique is curve fitting and generalization via interpolation. So you embed lots of data into an interpolating manifold. This manifold can do fast but approximate judgment calls about the target space. So the big idea is to leverage these fast but approximate judgment calls to fight combinatorial explosion. So you use them as a form of intuition about the structure of the program space that you're trying

to search over, that you're navigating, and you use that to make search tractable. So a simple analogy that sounds a little bit too abstract is drawing a map. You take a space of discrete objects with discrete relationships that would normally require combinatorial search like path finding in the Paris Metro is a good example. That's a combinatorial problem. And you embed these discrete objects and their relationships into a geometric manifold where you can compare things via a continuous distance function. And you can use that to make fast but approximate inferences about relationships. Right? RLHF can pretty much draw a line on this map, look at where it intersects, and this gives you a sort of like candidate path that restricts the the the the set of

Applying Combined Approaches to ARC Tasks

François Chollet 00:39:59

discrete passes that you're gonna have have to have to look at 1 by 1. So So this enables you to keep combinatorial explosion in check. But of course, you cannot draw maps of any kind of space. Right? Like program space actually, very, very nonlinear. So in order to use deep learning for a problem, you need 2 things. You need an interpretive problem, like it needs to follow the manifold hypothesis, and need lots of data. If you have only 2 examples, it doesn't work. So if you look at a single RKGI task for instance, it's not interpolative and you only have like 2 to 4 examples, so you cannot use deep learning. You cannot solve it purely by reapplying a memorized pattern either, so you cannot use an LLM. But if you take a step down, lower down the scale of abstraction and you look at core knowledge, the core knowledge systems that ArcGI is built upon, each core knowledge system is interpolative in nature and could be learned from data. And of course, you can collect lots of data about them. So you could use deep learning at that level to serve as a perception layer that passes arc world into discrete objects. And likewise, if you take a step higher up the scale of abstraction and you look at the space of all possible ArcGI tasks and all possible programs that solve them, then again you will find continuous dimensions of variation. So you can actually leverage interpolation in that space to some extent. You can use deep learning there to produce intuition over the the structure of the space of our tasks and the programs that solve them. So based on that, you know, RLHF think there are 2 exciting research areas to combine deep learning and program synthesis. The first 1 would be leveraging discrete programs that incorporate deep learning components. So for instance, use deep learning as a perception layer to pass the real world into discrete building blocks that you can fit into a program synthesis engine. You can also add symbolic add ons to deep learning systems which is, you know, something I've been talking about for for a very long time but it's ac-

tually starting to happen now with things like external verifiers, to use for LLMs and so on. And the other direction would be deep learning models used to inform discrete search and improve its efficiency. So you use deep learning as a driver, as a guide for program synthesis. So for instance, it can give you intuitive program sketches to guide your program search. And it can reduce the space of possible branching decisions that you should consider over each node and so on. So what would that look like on ArcGI? I'm going to spell out to you how you can crack ArcGI and win 1000000 dollar maybe. So there are 2 directions you can go. First, you can use deep learning to draw a map in a sense of grid state space. Grid space. So in the limit, this solves program synthesis because you take your initial grid input, you embed it on your on your manifold, then you look at the grid at which you embed it, and then you you draw a line between the 2 points on your manifold. And you look at the grids that it interpolates. And this gives you approximately the series of transformations to go from input to output. Right? You still have to do local search around them of course because they they this is fast but this is very approximate. It may not be correct. But it's a very good starting point. You are turning program synthesis into a pure interpolation problem. And the other direction you can go is program embedding. You can use deep learning to draw a map of program space this time instead of grid space. And you can use this map to generate discrete programs and make your search process more efficient. So a very crude example of how you can combine LLMs with discrete program search is this paper. Hypothesis Search inductive reasoning with language

State-of-the-Art Solutions and Future Directions

François Chollet 00:44:20

models. So it uses LLM to first generate a number of hypotheses about an ARC task in natural language. And then it uses another LLM to implement candidate programs corresponding to each hypothesis in Python. And so by doing this, they're actually getting a 2 x improvements on ArcGI, so that's very promising. Another very crude example that's very promising is the submission from Ryan Greenbat on the ArcGI leaderboard, the public leaderboard. So he's using a very sophisticated prompting pipeline based on GPT 4 0 where he uses GPT 4 0 to generate candidate Python programs to solve our tasks, and then he has an external verifier. He's generating like thousands of tasks per thousands of programs per task. Also, as a way to refine tasks programs that seem to be that seem to be doing, you know, close close to what you want. So this scores 42% on the public leaderboard, and that's the current state of the art. So again, here's where we are. We know that LLMs fall short of AGI. They're great at system 1

thinking. They lack system 2. And meanwhile, progress towards system 2 has stalled. Progress towards AGI has stalled. The limitations that we are dealing with with LLMs, they are still the exact same we are dealing with 5 years ago. And we need new ideas. We need new breakthroughs. And my bet is that the next breakthrough will likely come from an outsider while, you know, all the big labs are busy training bigger alarms. So maybe it could even be someone in this room. Maybe you have the new ideas, that RLHF wanna see. So see you on the leaderboard for ArcGI. Thank you.

References

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

[2] **Embers of Autoregression**
https://arxiv.org/abs/2309.13638
paper

[3] **Monty Hall problem**
https://www.tandfonline.com/doi/abs/10.1080/00031305.1975.10479121
paper

[4] **LLM Training Dynamics Analysis**
https://arxiv.org/abs/2205.10770
paper

[5] **GPT-4 Technical Report**
https://cdn.openai.com/papers/gpt-4.pdf
paper

[6] **Faith and Fate: Limits of Transformers on Compositionality**
https://arxiv.org/abs/2305.18654
paper

[7] **The Reversal Curse in LLMs**
https://arxiv.org/abs/2309.12288
paper

[8] **LM-Polygraph: Uncertainty Estimation for LLMs**
https://arxiv.org/abs/2311.07383
paper

[9] **ARC-AGI GitHub Repository**
https://github.com/fchollet/ARC-AGI
tool

[10] **Baldur: Whole-Proof Generation**
https://arxiv.org/abs/2303.04910
paper

[11] **ARC Prize**
https://arcprize.org/
tool

[12] **Thinking, Fast and Slow**
https://www.amazon.com/Thinking-Fast-Slow-Daniel-Kahneman/dp/0374533555

book

[13] **Core Knowledge in Infants**
https://www.harvardlds.org/wp-content/uploads/2017/01/SpelkeKinzler07-1.pdf
paper

[14] **Hypothesis Search with LLMs for ARC**
https://arxiv.org/abs/2309.05660
paper