/mlst

AlphaEvolve: A Gemini-Powered Coding Agent [Google DeepMind] — Matej Balog & Alexander Novikov

March 23, 2026



Machine Learning Street Talk

Podcast Transcript

Revision: 3abb76a7

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

Introduction: AlphaEvolve's Breakthroughs and DeepMind's Lineage 4

Introducing AlphaEvolve: Evolutionary Architecture and LLM Pairing 9

The Halting Problem and Evaluation Constraints 13

Knowledge Augmentation: Meta-Prompting, Library Learning, and Self-Generated Data 15

Matrix Multiplication Breakthrough: From Strassen to 48 Multiplications 18

Problem Representation: Direct Solutions, Constructors, and Search Algorithms 22

Surprising Outcomes: What Researchers Did Not Expect 24

Hill Climbing, Program Synthesis, and Intelligibility 27

Real-World Applications: Complex Evaluations and Robotics 31

The Role of LLMs, Recursive Self-Improvement, and Future Directions 34

References 38

Overview

Google DeepMind just dropped AlphaEvolve, a Gemini-powered evolutionary coding agent that designs advanced algorithms by pairing LLM creativity with automated evaluation. The headline result: it beat Volker Strassen's 56-year-old record for 4x4 matrix multiplication, finding a method that uses 48 scalar multiplications instead of 49. No human or AI had managed that in over half a century.

Tim sits down with two of the researchers behind the work – Matej Balog and Alexander Novikov – to walk through the system, the results, and what it means.

In this episode:

- How AlphaEvolve works: an evolutionary pipeline that pairs LLM-generated code proposals with rigorous automated evaluators, iteratively improving solutions rather than relying on one-shot generation. The gap between single-shot LLM sampling and scaled evolutionary search turns out to be enormous.
- The matrix multiplication breakthrough: AlphaTensor tried for years with reinforcement learning and only cracked the Boolean case. AlphaEvolve found a 48-multiplication algorithm for general 4x4 matrices almost by accident, running "for completeness." The result generalises from complex to real matrices, which is counterintuitive – solving the harder problem actually made the search easier.
- Three ways to represent the search target: direct solution, constructor function, or search algorithm. For matrix multiplication, AlphaEvolve designed a gradient-based search algorithm that finds matrix multiplication algorithms – a meta-level approach that produced loss functions and update rules no human would have tried.
- Real-world impact at Google scale: AlphaEvolve recovered 0.7% of fleet-wide compute resources in the Borg data center scheduling system and sped up Gemini training by 1%. These are already-heavily-optimized production systems.
- The human-AI collaboration loop: AlphaEvolve is not autonomous research. Humans choose the problems, design evaluators, seed initial solutions, and interpret results. Alexander Novikov argues this back-and-forth is the whole point – the system improves your questions as much as your answers.
- Keith Duggar probes the halting problem and evaluation cascade limitations. The researchers acknowledge the constraint but note that practical framing (time-bounded evaluation, evaluation cascades from cheap to expensive) sidesteps the theoretical issue for now.
- The recursive self-improvement question: AlphaEvolve improved the infrastructure that trains the models that power AlphaEvolve. The feedback loop exists but currently operates on a timescale of months.

GTC is coming, the premier AI conference, great opportunity to learn about AI.

NVIDIA and partners will showcase breakthroughs in physical AI, AI factories, agentic AI, and inference. Register for virtual GTC for free, using Tim's link (https://nvda.ws/4q

Tufa AI Labs is a new research lab in Zurich started by Benjamin Crouzier focused on o-series style reasoning and AGI. They are hiring a Chief Engineer and ML engineers. Go to https://tufalabs.ai/

Introduction: AlphaEvolve's Breakthroughs and DeepMind's Lineage

Tim Scarfe 00:00:00

We've had exclusive early access to the brand new Google Alpha Evolve paper, which just got released 1 minute ago. We did a technical interview with the authors before anyone else. The paper itself drops a bombshell, setting world records for many algorithmic and mathematical challenges. In the world of computer science, few problems are as fundamental as matrix multiplication. For over half a century, a specific efficiency benchmark in this domain, particularly for 4 by 4 matrices, seemed insurmountable. The search space for optimal algorithms is immense, making exhaustive exploration practically impossible even for relatively small matrices. In 1969, Volker Strassen revolutionized the field by discovering an algorithm to multiply 2 2 by 2 matrices using only 7 scalar multiplications, down from the standard 8. The established best practice for larger matrices like 4 by fours was to apply Strassen's 2 by 2 method recursively. For 4 by 4 matrices, this meant 7 times 7, resulting in 49 scalar multiplications. Today, AlphaEvolve beat this record. Google DeepMind has a long history of building AI systems which actually invent new knowledge through experimentation and iteration rather than just building a glorified database. We saw AlphaGo, which beat Lee Sedol learning from human games and even surpassing champions through self play. AlphaZero was purely self play, and AlphaFold predicted millions of 3 d protein structures that had never been measured experimentally, and AlphaDev discovered faster sorting algorithms. Google DeepMind recently has been focused on scientific discovery with AlphaTensor, which framed the problem of finding faster matrix multiplication algorithms as a game, achieving breakthroughs. And FunSearch took us even further, using large language models to find new mathematical solutions by evolving code. And now, AlphaEvolve represents the next stage in this lineage.[1][2]

Matej Balog 00:02:08

For the usual case of matrices that have arbitrary numbers in them, still nothing better was known than doing stress and twice using 49 multiplications. So we're, like, really excited when we used alpha evolve in this setting. We actually didn't even hope that it would find something better than 49 because we're trying for so long with alpha tensor. We just ran it for completeness because we wanted to have this, you know, this table in the paper that we actually have that we tried on

[1] AlphaEvolve Blog Post — Blog Post The official Google DeepMind blog post announcing AlphaEvolve. Subject of the entire interview.

[2] AlphaEvolve Blog Post — blog_post

all the sizes up to up to 5 or 6. And and remarkably, it found a faster algorithm, which uses 48 instead of 49 multiplications.

Tim Scarfe 00:02:44

It's a bit like Cursor on steroids. It iteratively refines algorithms, drawing on the creative power of LLMs using meta learning, library learning, automated evaluation, and evolutionary search. The Alpha Evolved paper describes it as an evolutionary coding agent which substantially enhances capabilities of pre trained large language models on difficult tasks. Of course, their their first paper was FunSearch, which was very, similar. The main difference, I think, was that it was just searching for a single function rather than AlphaEvolve, which can essentially work over an entire code base. You just demarcate adaptable regions in a code base, and it can search for those. And of course, it's even optimizing for interactions between those those functions in different parts of your code base. 1 potential issue, as Keith Duggar pointed out, is the classic halting problem in computer science.[3][4]

Keith Duggar 00:03:38

But there's also a really subtle limitation here I want to explore with you guys a bit, which is running the programs themselves, you know, we we face issues. Right? You face issues of the halting problem for 1 thing. Like, you start running some code, like, the code compiles fine or doesn't crash immediately and then it starts running. But, you know, maybe some time goes by, like an hour. And you're like, well, geez. Is this thing ever gonna terminate?

Matej Balog 00:04:04

In in theory, you're, of course, right. You can never tell that if you run an algorithm for longer what it would have done. But in practice, it actually hasn't been, any sort of issue for us in the applications that we looked at.

Tim Scarfe 00:04:15

Alexander Novikov added that this challenge is very much like a fundamental aspect of human research too.

[3] FunSearch — Paper The predecessor to AlphaEvolve. Used LLMs to evolve code for the cap set problem. Published in Nature.

[4] FunSearch: Mathematical discoveries from program search with large language models — paper

Alexander Novikov 00:04:22

How do you know that you should stop working on your problem as a human? Like, or maybe like, maybe you spend a month more and then you solve it. Right? It's it's it's hard.

Keith Duggar 00:04:29

I mean, course, there's still there's still limitations. They they didn't bite much on my my halting problem, you know, limitations. So, like, in practice, you know, it was it was it was okay for them, but they had these very well chosen well chosen problems. Right? Where like, okay, matrix multiplication, you know if it's any slower than Strassen's algorithm, you know, maybe you can you can turn it off. But the problem is that that restricts your open ended search capabilities. Right? You can't go down the path of a potentially slower algorithm for now that may get you to a stepping stone that hops you over to something more efficient. Right?

Tim Scarfe 00:05:05

It is interesting, isn't it? There's so many breakthroughs in science are unknown unknowns. We might have a nose for what is interesting, but we never really know where to find the answers. And so often it's the case that we find them through sheer luck. But there must be at least 1 step we can take in the direction of mechanising and accelerating this process. In the case of matrix multiplication and several other scientific problems, which had a clear evaluation function, this new evolutionary approach achieved much which decades of human research could not.

Matej Balog 00:05:38

We just ran it for completeness because we wanted to have this, you know, this table in the paper that we actually have that we tried on all the sizes up to up to 5 or 6. And and remarkably, it found a faster algorithm, which uses 48 instead of 49 multiplications. Yeah. When when when 1 of my teammates messaged on the channel that, oh, it seems like we have this result, I just couldn't believe it. Like, let's triple check it.

Tim Scarfe 00:05:59

AI's growing ability to generate entirely new, provably correct algorithms can advance the frontier of science. And the really cool thing is that AlphaEvolve has already been applied to optimize mission critical, real world systems within Google.

Matej Balog 00:06:15

They're able to speed up really already heavily optimized pieces of important computational infrastructure within Google.

Tim Scarfe 00:06:22

For instance, take Google's massive data centers. Efficiently scheduling computing jobs is a really complex operation. If done suboptimally, expensive servers will sit around idling. The Google engineers placed a candidate solution into AlphaEvolve and then evolved a smarter heuristic, which assigned jobs to machines much more efficiently. They said post deployment measurements across Google's fleets confirmed the simulator's results, revealing that this remarkably simple yet effective function continuously recovers on average naught point 7% of Google's fleet wide compute resources, which would have otherwise been stranded. Now that's a huge saving on Google's scale. Right? And in another instance of self improvement, it even found ways to accelerate the training of the very Gemini models which powers AlphaEvolve itself.

Matej Balog 00:07:15

We have found a way to speed up the training of the next version of Gemini by by 1%, and we have been able to unstrand resources in in the in the Borg data center.

Tim Scarfe 00:07:25

This instance was also interesting because it didn't generate the solutions, but also the programs which generated them.

Matej Balog 00:07:32

Whenever you look at the index in the for loop, you just look at the index modulo 4. And when you index into the the array, you're indexing in at position I and I plus 4 and I plus 8, like, okay, what's going on? And and just by inspecting the code, we actually were able to develop can think of it as like a mathematical insight or a mathematical hypothesis, and that hypothesis turned out to be really crucial for then improving the result.

Tim Scarfe 00:07:54

A key aspect of AlphaEvolve sophistication lies in its flexible approach to representing the search problem.

Alexander Novikov 00:08:01

They also propose you, like, a broad range of things, and some of them will be stupid, some them will be amazing, some of them will be, like, really weird. And then by having the ability, you can filter for those and identify the ones that are actually kind of important and improving things.

Tim Scarfe 00:08:14

So isn't it really cool that rather than trying to generate the solution itself, AlphaEvolve can, just like inception, generate the thing which generates the solution. We should play some inception music in the background. The thing we found really interesting about AlphaEvolve is that it's still very much a humans in the loop thing. Humans identify what's interesting. They find problems that have clear evaluators. They place candidate solutions in the loop, and then AlphaEvolve will traverse this cone of possibilities and make jumps along the way, and then the cycle continues. So this is very much sketching a future of AI where there is a strong collaborative loop between humans and AIs. And we should just bring in how you modelled this representation. So you had these different approaches. You could directly model the solution. You could model a constructor function. So if, you know, you're actually learning a function which itself constructs the solution. You could be learning a search algorithm, which is what you did with the matrix multiplication. And you also spoke about coevolution as as a possibility. So many people are talking about this vision of AIs which can autonomously just drive cars or just do anything, generate content without any supervision from humans. And that hasn't really panned out, to be honest. And certainly, a lot of the content on the internet is slop. Do you remember that dead internet theory by Illuminati Pirates, this guy on an internet forum a couple of years ago? He said that by now, most content on the internet will be generated by AI and it will have a kind of superficiality to it. And he was right. But it doesn't necessarily mean that AI is bad. Right? What's missing is that we need to have this exchange. We need to use AIs as tools and we need to guide them and refine the results and do the process iteratively. That's kind of what AlphaRevolve does. It it mechanizes the the correct way of using AI.

Alexander Novikov 00:10:09

The the thing that makes AlphaEvolve so cool and powerful is this kind of this back and forth between humans and and machines. Right? And, like, the humans ask questions, the the system gives you some form of the answer, and then you, like, improve your intuition. You you improve your question answering question asking ability. Right? And you ask more questions.

Keith Duggar 00:10:27

I gave this this talk a few times about, like, generation AI, and the big warning I gave is just the rise of mediocrity. We're just gonna be inundated and flooded with mediocrity, and the best content will still be produced by the most skilled people. And all that's gonna happen is as this tide of mediocrity grows larger and larger, people will get hungrier and hungrier for for the cream, for the things that are rising to the top. Right? So I think all that's gonna happen in the end is that skilled people, their productivity is gonna just rise, and they'll they'll continue to be differentiated from from the mediocre sort of hordes that are whose productivity is also enabled. So, you know, the, you know, the hordes become more productive and and experts become more productive. And and just as a whole, we all become more productive.

Tim Scarfe 00:11:21

And while we're on the subject of amazing innovative architectures

Introducing AlphaEvolve: Evolutionary Architecture and LLM Pairing

Tim Scarfe 00:11:24

for discrete program synthesis and reasoning and whatnot, Why don't you consider applying to work at Tufa AI Labs? That's if you're an ML research scientist or ML engineer. Benjamin Crouzier is running the lab. It's in Zurich at the moment, and they're thinking about opening an office in San Francisco as well. They would love for you to get in contact with them and apply if you're interested in working for them. So give Benjamin a shout. Guys, welcome to MLST. It's an honor to have you both here.

Matej Balog 00:11:54

Thank you for having us.

Tim Scarfe 00:11:57

So we have had privileged access to read your Alpha Evolve paper. It's really, really exciting because this is very much up our street. We love program synthesis and we love evolutionary methods. I mean, we had Kenneth Stanley on and, you know, Jeff Clune. I was speaking to him at NeurIPS and of course, he wrote the MAP-Elites paper, which influenced some of the stuff that that you guys have

done. But, know, just just from 1000 miles away, could you describe the the work that you've done?[5][6]

Matej Balog 00:12:25

Yeah. So we are presenting a coding agent, which we called AlphaEvolve. And what this agent is able to do, it's it's able to design quite advanced algorithms. And when I say advanced, I mean it's algorithms that are able to make new discoveries in the sciences, and we have quite a few examples in mathematics and computer science in this paper. Or on the practical side of things, they're they're able to speed up really already heavily optimized pieces of important computational infrastructure within Google.

Keith Duggar 00:12:57

Yeah. I'm curious. I mean, what, what made you go down the path of including EA algorithms, you know, evolutionary algorithms? What what kind of said to you guys, this is a component that we need in a hybrid system along with, you know, verifiers and LLMs and whatnot to to take us in a step forward?

Matej Balog 00:13:20

Yeah. I think if you consider the process of scientific discovery, then it's a very natural choice. And you mentioned you you spoke to Kenneth and, like, evolutionary algorithms on a high level, they give you this this diversity of the in the explore in the exploration process, making sure that you don't, early on in the process, kind of just zoom in on into a particular approach, which might be suboptimal in the end, but you keep exploring the vast array of possibilities that you have. And especially when you think about trying to solve really difficult problems and making new scientific discoveries, then a priori, there is no way of knowing what is going to be the right approach. So you do need to make sure that you keep exploring different possibilities. And evolutionary algorithms are just a good technical tool that fit the bill, really well for this purpose.[7][8]

[5] MAP-Elites — Paper Quality-diversity evolutionary algorithm by Mouret and Clune that inspired AlphaEvolve diversity preservation.

[6] MAP-Elites: Illuminating search spaces by mapping the performance of solutions — paper

[7] Alexander Novikov — Person Google DeepMind researcher and co-lead on AlphaEvolve. Guest on this episode.

[8] Alexander Novikov — person

Alexander Novikov 00:14:10

And, also, I think it's they're, like, very fun to play with. Right? Like, if you want to set up an RL algorithm, it's it's gonna take you, like, I mean, depending on on the algorithm, I guess, but it's gonna take some some time. Right? And with EA algorithms, you can just do it. Right? Like, it's you have LM APIs. You you just call things. You you try things. It's it's it's fun.

Tim Scarfe 00:14:29

Alex, can you just just sketch out the architecture? Many folks at home will will be familiar with FunSearch, for example, and I I guess this is an evolution of that, you know, pun intended. But can can you just kind of sketch out how the whole thing works?

Alexander Novikov 00:14:43

Yeah, of course. I mean, should I assume that people are aware or should I start with from scratch with fun stuff? Fun service?

Tim Scarfe 00:14:52

Yeah. Let's keep this bit really really really simple. We'll kind of the way we'll do the show is we'll have a progressive disclosure of complexity. So we'll have a have a hook and it'll be super super broad and then we'll kind of progressively get more detail. But you know, this bit just do like super high level. How does the whole thing work?

Alexander Novikov 00:15:06

Yep. Makes sense.

Keith Duggar 00:15:07

And we'll also we'll also show, you know, diagrams from your paper and any others you give us and, you know, animations and some visual aids.

Alexander Novikov 00:15:15

Mhmm. Awesome. Yes. So the high level architecture of HealthEvolve is it's an evolutionary method where you basically pair the so we only focus on problems where you have a way of evaluating the progress. Right? Like, for any given proposal, for for any good, like, piece of code that the system gives you, you can automatically test if it's good or not and, like, how good it is. And this is, I think, the key aspect of, like, maybe the results we got that, like, this having this simulator is, on the 1 hand, it sticks you to the set of problems where you you have you can have it. But on the other hand, it's quite a broad range of problems, and then it gives you a lot. Right? Like, you can quickly iterate and, like, get feedback. And what in particular gives you is that you can pair the creativity of LLMs with the simulator. Right? So, like, the arms will propose you some kind of, like, a broad range of things, and some of them will be stupid. Some of them will be amazing. Some of them will be, like, really weird. And then by having the evaluator, you can filter for those and identify the ones that are actually kind of, yeah, yeah, important and improving things. And then this kind of pairing LMs with evaluators is kind of wrapped around in an evolutionary pipeline, which try to iteratively identify the most promising pieces of code and then, like, focus on improving those and, like, exposing them to the LM. Like, here's what you tried before. This thing worked. This thing doesn't work. Like, please try to propose a new thing. And then maybe the final ingredient is scale. Right?

Tim Scarfe 00:16:43

Like,

Alexander Novikov 00:16:43

making those things in parallel.

Keith Duggar 00:16:47

So you you mentioned that needing the evaluator, it limit it limits the class of problems, but there's also a really subtle limitation here

The Halting Problem and Evaluation Constraints

Keith Duggar 00:16:56

I wanted to explore with you guys a bit, which is running the programs themselves. You know, we we face issues. Right? You face issues of the halting problem for 1 thing. Like, you start running some code, like, code compiles fine or it doesn't crash immediately and it and it starts running. But, you know, maybe some time goes by, like an hour. And you're like, well, geez. Is this thing ever gonna terminate? You know, is this gonna contribute to my gradient or not? I don't know. So maybe I have to just terminate it after some resources are consumed and but if I waited just, you know, 5 more minutes, I would have gotten an answer that was the godly algorithm. Right? So, like, this fundamental problem, how do you how do you deal with it now? How do you think we're ever gonna get around issues like that? You know, what do you envision for overcoming that limitation?

Matej Balog 00:17:45

So in in theory, you're, of course, right? You can never tell that if you run an algorithm for longer, what it would have done. But in practice, it actually hasn't been any sort of issue for us in the applications that we looked at. And 1 concrete thing I can say is that often you might frame the problem in the way where the kind of time constraint is built into the problem definition. So let's say you could say, I'm trying to solve this open problem in mathematics, and I'm looking for a search algorithm that is able to make progress on this open problem. But I want a search algorithm that is able to make progress in 10 minutes. So that is part of my problem definition. And then when I'm evaluating the proposals that the language models make, I only run them for the 10 minutes. And so I I only explore the space of algorithms that is able to make something happen within those 10 minutes. Sure. I might miss out on algorithms that would have done even better when run for longer. So that is indeed like a principle thing that you can never eliminate. But in practice, it actually hasn't been an issue for us.

Alexander Novikov 00:18:48

And I guess it's kind of this question is also fundamental to internal research. Right? Like, how do you know that you should stop working on your problem as a human? Like or maybe like maybe you spend a month more and then you you solve it. Right? It's it's it's hard. I don't know.

Tim Scarfe 00:19:04

Yeah. It's like the secretary problem. 1 thing I'm I'm fascinated in, know, interviewing so many people in the space is, you know, we we speak about diversity preservation, novelty, serendipity, open endedness, creativity. And we really want to design algorithms that can break free, that can make creative jumps. I mean, Demis spoke about this ladder of creativity where you have inventive creativity. That's the thing that we really need. And as I understand it now, in in your system, it's a little bit like an automated version of cursor where you kind of have these code gates and you put an initial solution in there. So there's a little bit of domain knowledge. There was 1 example where I think you decided to use a bin packing algorithm for doing the scheduling on, you know, on on the the hardware, you know, at Google. And I guess the question is is that depending on the starting solution, there's a kind of cone of things of leaps that we can make. And can you speak to any examples where the system made really imaginative leaps?

Matej Balog 00:20:05

Yes. So I can talk to that. And indeed you identified 1 interesting feature of the system, which is that depending on what you tell it at the beginning, you can guide the process. So if you give it fairly specific instructions or you ask the system to start from a specific type of solution, then what it will usually do, it will, like, squeeze out the juice of that idea that you gave it or squeeze out the juice of that initial solution, see how it can tweak it and bring it to its maximal potential. But and sometimes this is the right approach to take maybe when the problem is particularly difficult or has some specific features. But by default, you would start with a solution that's like really, really empty. So so you you give alpha evolve a code skeleton where, where all the, like, all the functions are are, like, almost an empty implementation. You just, like, return 0 or return false and so on. And and you just let it let it to be completely creative. And so it just has to rely on, like, its background knowledge of the base LLMs, and it can explore in all possible directions. And the, the evolutionary algorithm makes sense that you preserve the diversity as you as you keep doing the exploration. And 1 particular example I can talk about is we applied AlphaEvolve to discovering algorithms for matrix multiplication. And we did it by asking it to actually design kind of a search algorithm, a gradient based search algorithm that looks for the matrix multiplication algorithms in turn. So it's a bit of a meta thing. Like, you look for an algorithm that finds an algorithm. But within that first algorithm, the search algorithm, we started from a really simple code skeleton, giving it basically nothing. We just told it use gradients, basically. And then it was able to to write these complex, loss functions and update functions, which had all all sorts of tricks about penalizing various behaviors and introducing randomness in completely

unexpected ways, which we're like, okay. Wow. Like like, this this is the type of code that maybe a human could plausibly write, but but would they have actually thought of writing this particular piece of code? That, yeah, that that was really an moment, at least for me, that, wow, this is doing something kind of human like, but not something that obviously a human would try. And

Alexander Novikov 00:22:18

maybe another cool story about this is that I don't think we we added it to the paper because maybe paper is not the right format for this, but Adam and our team did a cool experiment on trying to give advice from humans to to the system. And then he asked a few people for, like, you know, you please think about this problem for 2 minutes. You please think about the problem for 30 minutes, and then, like, compare it and write down the notes and then give it to the system to to kind of guide it for for the process. And then he compared, like, what what would be the outcome of that. And you can see that, as Matej was saying, it's kind of squeezing all the juice out of the idea. So, like, will, like, preserve the essence of the idea because it kind of guides the LLM towards things like that, but it will optimize a lot of small things. And, yeah, and, like, in a lot of cases, it will be in intelligence ways intelligent ways. In a lot of cases, it will be in kind of, you know, I'll try a bunch of things and 1 of them will stick. But yeah. So it's kind of cool to watch.

Keith Duggar 00:23:11

Yeah. And so at the moment, the architecture has, let let's say maybe 2 sources of base knowledge. It has the base model

Knowledge Augmentation: Meta-Prompting, Library Learning, and Self-Generated Data

Keith Duggar 00:23:20

itself, right, which is obviously compressed, you know, all of the corpus it's been trained on, including tons of code and algorithms and numerical recipes and whatever. And then it has the the starting program that you put in, the the sort of gated bits of logic and and whatnot. Is there is there any room for an augmentation that's almost a middle ground where there's say a secondary database that contains modules or pieces of code that have known to been very, be very effective in other problems that it can somehow simultaneously draw on?

Matej Balog 00:23:51

Yes. So a few points on this. 1 is that you mentioned there are 2 sources of kind of knowledge in the system. And debatably, I would say that there is a third source, which is that the system can decide to augment its own knowledge. What I mean specifically is that the system proposes an algorithm, and then it's going that algorithm is going to be executed on a machine, and you will see the results of of having run that algorithm. So Mhmm. On a sufficiently high level, you can think of it as the system can decide, okay. I want to gain the piece of knowledge. What does this algorithm do when you actually run it? So so that that is an important component. But maybe going closer to the essence of your question, indeed, so there could be a separate, either a human curated database of useful modules or any of that sort. But even more excitingly, this database can be curated by the system itself. And so this is an idea that, we are thinking about for alpha evolve, but there is a a related idea that is already implemented and mentioned in the paper, which is not building a curated set of modules, which are, generally useful, but a curated set of prompts that tend to work well. So there is this idea called meta prompting described in the paper where we actually ask our language models to propose their own prompts. So we just tell them what we are trying to do. Like, we are trying to do this evolutionary algorithm for, improving on this particular problem, and we are going to to to prompt you with this particular prompt. But before we do that, please propose a modification to this prompt itself. And then we curate a set of prompts that actually work well for this purpose. And so in spirit, that's a similar idea, although it's curating prompts rather than programs. But both make sense to me.

Tim Scarfe 00:25:34

Yeah. I think if I understand the direction Keith was going in, because everything you've just described is is fascinating. It's various forms of meta learning to basically create diversity and divergence. But I feel that the next trillion dollar business could be where we, you know, like in program learning, right, we want to construct a library. And right now, we're we're kind of expanding this library of functions for a particular purpose. But what if the library itself was the new oil, you know? What what there was strong robustness between these programs that we've learned for this thing and they generalize, you know, through some analogical relation to other programs and other domains. I mean, what if we just had like the new language model paradigm was actually a kind of program database? I mean, do you think that could work?

Matej Balog 00:26:25

Yeah. I think that's a fascinating idea. And, like, we see maybe some first glimpses of that. So it is true that for now we are going into that direction. We just focus deeply on the problem and just try to solve that problem. But even within that path, let's say when we worked on matrix multiplication, we saw that, like, when we run AlphaEvolve different times, we discover slightly different algorithms. And then it's actually a useful technique to take those algorithms and use them to initialize future experiments. So that's kind of a first step towards building this database of things that were useful in the past to act as inspiration for solving future problems. So, yeah, maybe Alex has more thoughts.

Alexander Novikov 00:27:08

Maybe another similar thing we kind of I feel that is happening is not that we store not that we use the necessarily the database of programs we produced over all the experiments we did, but kind of our own human intuition as, like, users of the system is definitely evolving. And I kinda feel like a consultancy, right, like, a person. Like, we we collaborate with a lot of teams at Google trying to help them run things with AlphaEvolve. And, like, through that process, we kind of gain a lot of knowledge of, like, what works, what doesn't, like, what what should what should we try next? And then this kind of sort of things, like, somewhat similar to what you're describing. Right?

Keith Duggar 00:27:44

Yeah. I mean, I can almost imagine Alpha evolve having its own repo, you know, whether it's internal or or maybe be nice enough to put it up on GitHub for us. But it would just be constantly evolving and contributing to its own repo and maintaining it and categorizing it. People can go take a look. I wonder if AlphaEvolve's come up with a better search algorithm. Go check the the search area and see if it's got anything Yeah. New

Matej Balog 00:28:11

Like, technologically, I I don't see hurdles for this. It's maybe the organizational question of how exactly to make it happen. And 1 particular technological piece that's already there is that we already now look for programs that work well across a range of tasks. And right now, maybe this range is, let's say, fairly constrained because we care about the constrained range, but there is nothing blocking us from expanding it. So for ex like, concretely, we would be looking for search algorithms that are able to find, say, matrix multiplication algorithms for different sizes simultaneously. So there is this sort of generality, but there is nothing preventing us from saying, let's look for search algorithms that actually work well across a much broader range of tasks. That's not just matrix multiplication, but also other search problems.

Tim Scarfe 00:28:58

Can we meditate on the matrix multiplication thing only because that's that's the headline hook of this video. You know, I think we're we're gonna start the video by saying there's this amazing result. There was

Matrix Multiplication Breakthrough: From Strassen to 48 Multiplications

Tim Scarfe 00:29:08

this, you know, Strassen guy 56 years ago. He had this big result. Now alpha alpha revolve has has just defeated it. There's quite a few things to explore here. Mean, obviously, like, please just explain the whole thing. But also, there's some interesting stuff around we went up to, I think, a maximum of this is like complex 2 d matrix multiplication, you went up to rank 6 and even that was a bit of an overshoot and there was some interesting properties where as the rank went up, it started to get a little bit sketchy, but it it did improve more. Can you can you just, like, you know, talk through that whole story?

Matej Balog 00:29:43

So let me start maybe with the high level picture first. So multiplying matrices, there's like a super basic operation and and some of us, like get taught this operation in high school and there is a very specific way how you multiply matrices when you're taught to do this in high school. Like you at every step, need to take 1 row of 1 matrix and 1 column from the other matrix. You compute the the scalar dot product of these 2 things and that gives you 1 entry in the out-

put. And so this is 1 specific algorithm, which is kind of the basic algorithm for multiplying matrices. For for every element in the output matrix, you'll need to do this 1 dot product. And for a long, long time, people thought that this was kind of the obviously, the only way to multiply matrices. How could there even be something something better? And then you mentioned Volker Strassen in 1969. It was really a shock to the mathematical community. He He wrote this paper saying that actually, there is a faster way of doing it. And so already for multiplying 2 by 2 matrices, so the smallest nontrivial case, if you do it the high school way, you need to do 8 multiplications because there are 4 entries in the output matrix. It's a 2 by 2 matrix. And each scale, like each inner product requires 2 multiplications. So 4 times 2 is 8. But Strassen, he came up with this ingenious way of making a procedure that only requires 7 multiplications. And it's kind of a magical procedure where you you build some combinations of entries from the first matrix and the second matrix and you multiply them and then you are able to combine the 7 products in such a way that you get these magical cancellations and the result is just correct. So this was a big surprise in 1969 and it opened this entire new area of research. Like, okay. So for 2 by 2, you can actually do 7 instead of 8. So then people quickly proved that 7 is actually optimal for that small case. But already for 3 by 3 matrices, which you can think, okay, I mean, that that's, like, laughably small. Surely, people must have figured out what is the best way to do that, but we still don't know. So even today, we know that you need at least 19 multiplications to to do that, multiply 2 3 by 3 matrices. But the best algorithm we have is using 23. So there's this gap between 19 and 23 that people just haven't been able to close for for years. And and and the reason is that even though the matrices are very small, the space of possible algorithms, how you could multiply them is just is just completely immense. So, computationally, there is just no hope at all to do this exhaustively. So already for for 3 by 3, it's it's it's like open, which is crazy. And then okay. So for 3 by 3, the best algorithm is using 23. That's at least better than the algorithm you get taught in high school, which would be 27. So at least there is some progress on making it better. But then for 4 by 4, which is like the next size, the best algorithm that that had been known was just apply Strassen's algorithm recursively twice. So because Strassen is for 2 by 2 matrices, what you can do if you have a 4 by 4 matrix, we you consider it as a block matrix of 2 by 2 blocks, and then each block itself is a 2 by 2 matrix. So you can do Strassen twice. And because Strassen needs 7 multiplications, if you do it twice, you will get 7 times 7 49 multiplications. So for the, like yeah. So so this was the only way known how to multiply 4 by 4 matrices in in in a fast way, like using 49 multiplications just to stress them twice. And this has been the kind of the situation since 1969. And so this is where maybe our work comes in. So 2 years ago, we we built AlphaTensor, which was a specialized reinforcement learning agent for discovering matrix

multiplication algorithms. And that agent actually did find something faster, but only for, Boolean matrices. So it's this very special case where you want to multiply matrices where every entry is just 0 or 1. And when you do the multiplication, you do everything module 2. So for that case, AlphaTensor found something faster. But apart from that, for the usual case of matrices that have arbitrary numbers in them, still nothing better was known than doing stress and twice using 49 multiplications. So we're, like, really excited when we used alpha evolve in this setting. We actually didn't even hope that it would find something better than 49 because we're trying for so long with alpha tensor. We just ran it for completeness because we wanted to have this, you know, this table in the paper that we actually have that we tried on all the sizes up to up to 5 or 6. And and remarkably, it found a faster algorithm, which uses 48 instead of 49 multiplications. And and so that in itself, like yeah. When when when 1 of my teammates messaged on the channel that, oh, it seems like we have this result, I just couldn't believe it. Like, let's triple check it. But then it was indeed correct. And it actually has this 1 really appealing feature that like, usually you think about multiplying matrices. You want to multiply matrices where the entries are real numbers or maybe integers. The the case of multiplying matrices where the numbers are complex is maybe a bit less less common. But real matrices are just a special case of complex matrices. So if you find an algorithm that can multiply complex matrices, you can also apply to real matrices. It's just a just a generalization. So what's kind of cool here is that you let's say you care about multiplying real matrices, which is what you care about when you train neural networks, let's say, a very common use case. A priority, you might just look for an algorithm that uses real numbers. But you can say that, oh, actually, what if we look for a complex algorithm, which a priori would think, well, that's a more difficult task because that algorithm would actually apply not only to real matrices, but also to complex matrices. But by making the task more difficult, actually AlphaEvolve was able to find an algorithm that uses complex numbers and therefore applies to both complex matrices and real matrices. So that's that's kind of the the result that we were most excited to to actually get. But then as you were asking in your question, indeed, we we were applying AlphaEvolve to other matrix sizes as well. And and indeed, as you go to, bigger and bigger cases, like 5 by 5, 6 by 6, the problem becomes much, much harder very, very quickly. It's kind of an exponential with a quadratic in the exponent because the the tensor, this cube that we sometimes show in the visualizations that you have to decompose, it grows quadratically with the size of the matrices that you multiply. So for 4 by 4 matrices, you have to deal with a tensor of size 16, 16, 16. For 5 by 5 matrices, it's 25, 25, 25. So it's it's like exploding very quickly. And of course, as as you go higher, at some point, your method will will not scale there. But what we show is that alpha evolve, well, it scales further than alpha tensor. So so there

is some progress on the scaling direction. And then just 1 important point to clarify is that in all these cases, sure, we look at small cases of matrix multiplication, 2 by 2, 3 by 3, 4 by 4. But that doesn't mean that you can only apply these algorithms to matrices that are this small. Like, as I already alluded to with Strassen's algorithm, can apply them recursively. So if you have a big matrix, you treat it as a block matrix, and you apply these algorithms that are for smaller matrices recursively.

Keith Duggar 00:37:13

Yeah. I mean and it's super fascinating, and and it is such a fun a fun domain to work in and that almost magical that even when you get to 3 by 3, it becomes, you know, intractable. Mhmm. And I'm always fascinated by cases like that in mathematics. It's like, oh, we can do it for dimension 1 and 2 and 3, but then when we get to 4, it's just totally different. You know, it completely falls apart. But as you went higher, you also had cases where alpha evolved, it it couldn't match the the current performance and actually found worse. Like, it sort of found worse algorithms. So Yeah. What do you attribute attribute that to? Why was it So

Matej Balog 00:37:52

so in in the case that's the biggest 1 that we show in the paper, 6 by 6 matrices, There is a very, clear reason for that, which is that we try to apply AlphaEvolve without giving it domain knowledge about the the problem. We just wanted to see how good is AlphaEvolve as a general purpose tool. And we started kind of from scratch, and we don't tell it about any, you know, like tricks for developing specifically matrix multiplication algorithms. Okay. The the best known algorithm for 6 by 6 matrix multiplication, it uses a kind of very specific inductive bias, which is which means that it's looking for algorithms that have a specific symmetry in them, meaning that it only looks for algorithms that have a regularity in the algorithm. If you only look for algorithms that have this regularity, it's a much smaller search space. So in that search space, you are able to scale to much larger sizes. But we just didn't try to do this to incorporate this symmetry into our research. We just looked for algorithms of unrestricted form, and that I think is, at least to me, the clearest reason why we didn't match the best known solution in that case.

Tim Scarfe 00:39:03

Yeah. So 1 of the things that fascinated me most about the paper, I mean, obviously, we're very interested in abstraction and representation.

Problem Representation: Direct Solutions, Constructors, and Search Algorithms

Tim Scarfe 00:39:11

And even in the example that you just gave that Strassen was learned to be used as a sort of dynamic programming formulation, so being used recursively. And I'm thinking to myself, is that a demonstration of deep abstraction or is it a kind of superficial 1 step abstraction where the, the knowledge of Strassen was in its kind of local neighborhood and it was composing that. And in an ideal world, what we want algorithms to do is compose abstract basis knowledge. So knowledge which is as far down the stack as you can possibly do to sort of increase our our flexibility. And we should just bring in how you modeled this representation. So you had these different approaches. You could directly model the solution. You could model a constructor function. So if, you know, you're actually learning a function which itself constructs the solution. You could be learning a search algorithm, which is what you did with the matrix multiplication. And you also spoke about, co evolution as as a possibility. So this is mind blowing in the sense that there's some human design and intuition in how you design the optimization target. But there's this ambiguity in in the ways that you can represent so many of these problems and how they relate to each other. So can you talk me through that?

Matej Balog 00:40:21

Yeah. So first of all, want to be upfront about the fact that we ourselves don't have all the answers here. We have this tool, alpha evolve, and we see that Okay. It's it's like general it's like yeah. Not even about alpha evolve. So so we have this general tool that we see we can apply it here and here and here, and for every problem, there are different ways we can apply this tool. But a priori, we only have some intuition what is the right level of abstraction to apply this tool on. And, as you were mentioning, Tim, yeah, sometimes the best thing you can do is you directly search for the solution. Sometimes you you search for a, like, a simple constructor which constructs the solution. And here, as a, like, illustrated example, let's say you're looking for a solution and you think it's going to be very regular. Maybe it's going to be like look like a fractal. Like, that's the typical example. Then you know that to describe a fractal, can do it with a very short piece of code. So in that case, it makes sense to look for a description of the fractal, not as a, like, a grid of pixels, but instead as a short piece of function that just generates that fractal. But then in other applications, maybe the solution is very different, not so regular. And maybe you want to either look for the solution di-

rectly or you want to look for a complex search algorithm that finds the solution. And sometimes you want to have a sequence of algorithms that gradually refine the solution, which is the co evolution approach. And a priori, it's not clear at all which 1 is going to work best when. So that is something that is definitely in kind of the future work category to to build up that understanding. But 1 positive side of things is that, AlphaEvolve is easy to set up in in all the different formulations. So often in practice, you just try different things and and see what works best.

Keith Duggar 00:42:09

Well, yeah. I I can't believe you guys don't have all the answers. Like, I was I don't know why I'm here now. But, no. In all seriousness, and Tim mentioned that these sort of 3 modes that that AlphaEvolve, you know, currently runs in. Right? And it really reminded me of mathematicians because because, you know, mathematicians or there's at least 3 kind of main ways in which proofs are done. Right? They're either done by deduction, construction, or enumeration. And those almost seem to be like very parallel to these 3 methods. And I feel like there's probably some deep deep connection there that I'm missing. I'm curious if if either of you have have any thoughts on that. Any Yeah. See any thoughts on that deep deep connection there or or not?

Alexander Novikov 00:42:59

Sure. Yeah. I mean, it's an interesting observation. Yeah. Mamade, did you want to say something?

Matej Balog 00:43:06

Yeah. So so 1 thing is that when you think about constructions, that is the space where alpha is, like, most obviously applicable. So the the examples that we show in the paper is where you have open problems and you make progress on those open problems by finding better constructions. So that's where you're applicable out of the box. But then if you think about other approaches of making progress on other types of mathematical problems, let's say a problem is not obviously about a construction. Let's say you want to prove impossibility results, like lower bounds in a sense. Then then you have maybe 2 approaches on a high level that you can take. 1 is that often problems that don't look like a construction, they become a construction if you frame them in the right way. So for example, you you you have the, like, duality theorems for linear programming between the primal and the dual, you can often switch the side that you're actually trying to prove and turn something that isn't a constructive problem into a constructive 1. So that's kind of an alibisic answer, but that's something you can often do. The more difficult and more general answer is that sometimes you're actually look-

ing for a proof rather than a construction. Now now proof is you can think of a proof also as an algorithm. Like, it's a step of like, it's a sequence of steps that you need to execute to prove a statement. So this is still within the space of algorithm discovery and something that we we can't be thinking of doing. But there is 1 technical hurdle that we have maybe some initial signs of overcoming, but it's not something that, we have done in this paper, which is that if you're looking for an algorithm that is a proof, then the proof is, at the end of the day, either correct or not. So the reward is binary. It's like 0 or 1. For now, we have focused on problems where, you can make gradual progress. I can gradually improve the score, not just switch from 0 to 1 in a single step, but gradually, become better and better until you improve on the best on construction. Now but, of course, even as humans, when we write proofs, we we face the same issue, like, that the proof is only correct once it's actually done. But as humans, as we are writing, working on the proof, we will have some intuitions of, have we actually made some progress? Like, have we have we actually built some understanding about the problem? If so, then that seems likely that this will be a part of the eventual proof. So we have been exploring the possibility of using scores which are not hard scores, but soft scores. Maybe a language model can itself provide feedback. Like, does it look like we have made progress towards solving the problem? So we we do see the path of attacking these binary problems, let's say, searching for proofs, but it's not something that that we have already done. It's just something we see as a possibility of doing in the future with the this type of technique.

Keith Duggar 00:45:57

Yeah. I mean, and there's there's so many surprising things in this paper, and I know Tim and I wanna ask about more of But I'm kinda curious, you know,

Surprising Outcomes: What Researchers Did Not Expect

Keith Duggar 00:46:06

what each of you saw is I mean, after you did this work and you had the paper or the research done, least, you know, which was most surprising to you? And I'll start with Alex. Like, what what did you walk away from this work thinking, wow, I wasn't actually expecting that?

Alexander Novikov 00:46:21

I think it keeps amaze amazing me how, like, how much progress you can make with this sort of system. So I like when when we started with FunSearch a few years ago, you would go to a chatbot interface somewhere. Right? And, like, you would ask it to solve you an open problem and for, like, not give you anything, basically. Right? And then still today, like, you can try the same thing and, like, okay. It will probably make much more sense to, like, think for a few minutes. It will give you a lot of reasonable approaches. But, generally, it's not the experience of people that you, like, ask a chatbot to solve you an open problem and will just do that. Right? And then it's kind of amazing that by using this the same tools, right, the same LLMs in kind of this iterative evolution loops, you can get so much more out of them. It's yeah. I think it's like, every time we solve something new, it's it's really kind of exciting and and surprising in a sense.

Keith Duggar 00:47:13

And and, Mate, what what most surprised you?

Matej Balog 00:47:17

I think what is really new to me is that the generality of the approach, and I don't just mean a generality across scientific open problems, but it's not my experience from my, like, admittedly short research career that you you build some tool for, like, scientific purposes, then out of the box, you can do it apply it to real world challenges and have so much impact. Like, usually, there is an entire body of work, work that needs to happen to to translate a scientific technology into something that's actually useful in the real world. Like, usually, there is so many challenges you have to have to tackle. And here, there is a tool which, out of the box, at the same time, is able to make new discoveries on on mathematical and scientific problems, and at the same time, is able to discover algorithms that you can directly deploy into into Google's, like, critical compute stack. So so that's something I certainly hadn't experienced before and maybe wasn't expecting, to be honest.

Tim Scarfe 00:48:14

Yeah. I don't know if you're familiar with the ARC challenge and a guy called Ryan Greenblatt, he did this famous approach where he just sampled a language model, like, 30,000 times to generate programs. We're in this really interesting space as you as you spoke about in the paper where we have an evaluator, which means we can sidestep hallucinations. Right? So isn't it fascinating that these nuggets of gold are in there in the search space and of course most of us just use language models in a single shot. Right? Just to do greedy sampling and and now we can just, do these very sophisticated search routines. But I wanna get to well, actually, maybe like a a quick sub question is what what what you folks have done is you've mixed so many interesting paradigms together. You know, we're talking about meta learning and evolution and diversity preservation and and program learning and library learning and whatnot. But like if you could explain simply how is that different from just sampling a language model? Let's say you could sample a language model 1000000000 times. Has what you've done is ALPHA evolved like in a different category to that?

Matej Balog 00:49:20

Yeah. Yeah, for sure. So so maybe just 1 like personal observation on on what you just said. That is indeed the right way of thinking about it. If you just keep asking a language model in a in in a like a chat window repeatedly, you'll get a completely the wrong idea about what is the capability if you actually scale things up. Like when we started working on this type of evolutionary approaches, initially we were like pretty skeptical what this is going to be able to do because indeed, it's like we're just trying in a in a in a chat window and asking it to to to write some simple algorithms, and I it wasn't doing that well initially. And and the magic really happens when you when you scale things up. But there are different ways in which you can scale things up. 1 is that indeed you just keep asking repeatedly the same question. And, okay, sure, like, there will be some nuggets, but there but they'll they'll be nuggets. They'll not be the full solution. So it is really important that you just find those good nuggets, and then you iteratively build on top of them in subsequent iterations. And and that's what you get through through through these evolutionary evolutionary algorithms. So and just to speak very quantitatively, we do have a comparison in the ablations in the paper where try to get rid of evolution. And, I mean, as you might expect, that that works much, much worse. Yeah. I think

Alexander Novikov 00:50:37

maybe a good test case for kind of building intuitions about that is is the cap set example from the original fund search paper. So there, we used a very simple and, like, small language model, and, like, we didn't even provide context about the problem because we wouldn't ever expect the language model to actually, you know, think through the problem and, like, solve it by by thinking hard and, like, being very smart about it. What we're hoping to get from the language model is just, trying a few things, like, lot of things. Right? But the problem is that the final solution is like this. It's not that long, but it's, maybe, like, 50 lines of Python, but it's very, very unlikely that you'll stumble upon them by accident if you don't even think if you don't even know the problem. Right? Like, you're just trying Python things and, like, how would you generate the particular kind of 50 line Python program that works? So there, it was, like, completely essential to HillClimb. Like, you have to, like, improve gradually and, like, see what works and trying to modify it a little bit in the air like, in in in in the neighborhood.

Keith Duggar 00:51:32

Yeah. Can you can you actually talk a little bit more about that, the hill climbing? Because you've encore you've you've found clever ways in which to build in

Hill Climbing, Program Synthesis, and Intelligibility

Keith Duggar 00:51:42

hill climbing into problems that, at least on their face, seem so discreet that hill climbing would would not be possible. So I'm just curious if you could just expand a bit on the general idea behind how 1 takes a discrete problem, like looking at different programs, you know, which are fragile and digital, and somehow incorporates hill climbing into that process.

Alexander Novikov 00:52:06

Yeah. I guess for many problems, like, you you do have a natural kind of hill climbing reward. For example, when we do this gradient based method for finding matrix multiplication algorithms, like, natural reward will be the the loss, right, like how close you are to the solution. But unfortunately, that doesn't necessarily work very well. And what we've, like, what we found again and again is that, like, you have to be somewhat creative about what kind of auxiliary rewards

or, like, auxiliary signals you can come up with. And in particular, for the matrix modification case, what was very useful for us is to realize that, you know, if you have a curriculum of of matrix sizes, then it's probably gonna be somewhat easy to solve the small ones even with, like, you know, simple gradient based methods. And then you can hill 5 on kind of this probability of solving. Right? Like, you let's say, you run your shrink 10 times, and then, like, how many times you'd actually get the thing you're you're looking for. And then that can be, like, your signal. And then, like, because you have a curriculum, you can, like, say, okay. First, I want to solve 2 by 2. Then once I'm, like, solid on that, I can I want to, like, continue evolving by trying to solve 3 by 3? So that was really helpful in that particular case. But, yeah, in general, you just you just try things and you build intuitions about, like, how do you how do you approach coming up with these auxiliary words? It's it's very problem specific.

Matej Balog 00:53:24

And and maybe just to add 1, like, orthogonal point is that often you're trying to solve 1 particular problem, and then maybe it comes with an associated reward or maybe it doesn't. But it's nevertheless useful to also optimize other things simultaneously. So let's say you want to find a, you know, a matrix multiplication of a particular size. You also want to find algorithms for other sizes because if you do that, that allows you to explore the space of algorithms in a more broad way. And often, you can translate ideas that you develop for 1 matrix size later on to the matrix size that you actually care about. But if you only optimized for the size you care about from the start with, maybe that idea wouldn't have been useful initially. You first had to develop it and refine it a bit further for a for a related problem. So the overall point I'm trying to make is that it often makes sense to introduce kind of similar other tasks and try to improve on them, even if you don't intrinsically care about them.

Tim Scarfe 00:54:22

Could we touch on the benefits of program synthesis in general? So we interviewed Kevin Ellis recently. He's the guy who invented DreamCoder, fascinating guy. He used to work under Josh Hannanbaum. And and the way he described it coming from a cognitive science point of view is a lot of program induction is about explanation, it's about intelligibility, it's about legibility. There was this incredible example when you were talking about doing the scheduling of of jobs on, you know, in in Google's data center and and you've got this kind of matching problem and whatnot. And I I think you said there was some previous experiments where you'd use like an inscrutable, you know, reinforcement learning model or something like that and it wasn't debuggable, it wasn't intelligible, and now you've got this beautiful 3 lines. But even then there's the question of is it legible? Have you found things that are not legible? Remember there was that move 37 in AlphaGo and that was a famous example of of a discovery which was a little bit weird and perhaps we wouldn't have discovered it but maybe we understood it, maybe we could build theory around it. But when you look at some of these discoveries as well, are are you finding deep abstract principles that you can derive from those discoveries? I mean, talk me through all of that.

Matej Balog 00:55:32

Yeah. So you can span the broad spectrum, actually. So with AlphaEvolve, you sometimes discover algorithms that are really simple, and they're so simple that you can like, a human can verify that they're actually going to be correct on all inputs. And indeed, as you alluded to, they're so simple that you're just happy to submit them to production almost immediately, like no no further checks checks needed. And it's in a completely different league compared to trying to deploy a neural network, which you have to think about things like retraining it and hosting it and and and the resources of running inference and all those kinds of things. So in indeed, you can you can be in this very simple regime. For scientific discovery, you can also be in the regime where you explicitly look for programs that are interpretable. And you can do this by setting up the skeleton that you ask AlphaEvolve to fill in, in such a way that by design, you expect it to be fairly simple. And we actually maybe the clearest example is already in the fun search paper where, we looked for these big, capsets like a specific mathematical object, and it found the function that we could look at, inspect, and we just noticed that all this function is is using the number 4 in this interesting way that whenever you look at the index in the 4 loop, you just look at the index modulo 4. And when you index into the the array, you're indexing in at position I and I plus 4 and I plus 8. Like, okay. What's going on? And and just by inspecting the code, we actually were able to develop can think of it as like a mathematical insight or a mathematical hypothesis. And that hypothesis turned out to be really crucial for then

improving the results. Like, we we took that insight from the code, incorporated that insight into the next run, and we got much better results. So indeed, this can happen. But in some applications, maybe you don't care as much about interpretability and then AlphaEvolve can can develop even, like, very complex algorithms or a sequence of algorithms where you will perhaps not have the holistic understanding of how exactly this complex search heuristic works, but what you care about is the final result, that the final result is as good as possible. So you can you can span this this whole spectrum. And and maybe, Alex would also mention the the work that is not from our group, but another team has applied fund search to, cognitive science to exactly discover interpretable programs of behavior. So that is actually a pretty cool application that you can do.

Alexander Novikov 00:57:53

Yeah. And also if you maybe go back to the matrix modification example, there we kind of build this grieving based machine learning pipeline, right, like for for fight for looking for those algorithms. And there, I think if you look at the proposed code changes by AlphaEvolve, you would see maybe 2 type of changes 2 types of maybe reactions to the changes I I I experienced. 1 is like, yeah. That makes sense. Right? Like, it's and it's it's usually this it's usually the case of machine learning. Right? Like, when you write machine when you read machine learning papers, you're like, oh, yeah. This this makes sense. Right? Like, that's a good idea. I would maybe have done it myself. And then usually, problem is not coming up with the idea. It's like ideas are sort of cheap. It's just like the problem is coming up with the idea that actually works. So it's like there's so many things that make sense, and then, like, only, like, 5 of them actually work. So, like, it gives you ideas that you kind of understand and can relate to, but it's it's hard to know up your order, like, which of those will actually work. Right? And then maybe other type of ideas or, like, code changes you will see is things you would probably not even have tried just because they're over complicated, and sometimes it's for a good reason, sometimes it's for a bad reason. And, for example, for the matrix multiplication case, you would see the kind of so we have this quantization loss there because we want the solutions to be integers or, like, some, you know, maybe fractionals, but specified range such that we can exactly verify with an exact arithmetic that they're correct. And then we'll have this quantization loss which will drive the solution towards that set of integers, for example. And then, you know, normally, we as humans would only try, like, I don't know, tuning the the the weight of that quantization term or maybe in the worst case, like, the schedule of the weight. Like like, maybe you'll have less of it in the beginning and then, like, more of it in the end. But what AlphaEvolve did was producing, a whole kind of time evolving shape of the quantization loss, which which kind of, again, makes sense. Right? Like, you you wouldn't say that

it's not gonna work or you wouldn't or otherwise, you wouldn't looking at it, you wouldn't say that, oh, this is amazing. This is, like, definitely gonna work. But the thing is, like, you wouldn't even try it as a human because it's, like, so complicated that you would never even think about tuning such a such a complicated, like, kind of function which changes shape over time with iterations.

Keith Duggar 01:00:15

That's pretty fascinating. Definitely, it's it's kinda like how human players are able to glean some new insights from chess, you know, from from

Real-World Applications: Complex Evaluations and Robotics

●

Keith Duggar 01:00:24

AlphaZero and and whatnot. You did, Alex, you also mentioned, you know, about how surprised you were at the general or the broad application of the technique. And I know you have some prior prior work in robotics. Like, I'm just curious for for cases where it's more difficult to assess the code, you know. So for robots, you might be able to do some training kind of in a virtual reality or something. But then at some point, you gotta just, I don't know, throw it out in the woods and see if it's able to navigate to the other side or or something like that. How do you see bridging the gap between easily automated validation versus versus more complex real world scenarios, yet still being able to apply alpha evolve?

Alexander Novikov 01:01:11

Yeah. I was thinking that maybe 1 direction that makes sense, and I saw people doing that, is looking into trying to convert kind of fuzzy functions into code. So for example, maybe you have a reward function which is defined by the image of the final state you want to achieve. Right? Or maybe you want to have a reward function which is defined by, like, a natural language description of what you want to do. And then your task is to convert that into a Python function which actually scores it. And maybe maybe it's not that hard to have a binary word here for for the aurel to actually train the robot because, you know, you can ask the visual language model to to to verify it. But it's very hard to train against binary words. So, like, I saw people trying to use systems like AlphaEvolve to find a piece of Python code which would provide an auxiliary word for kind of shape like, shape reward, basically. Right? Like, to drive you towards that binary word to to make the learning actually faster. I think that that makes a lot of sense as a

direction, and, yeah, I wonder I don't know if that answers your question.

Keith Duggar 01:02:14

Yeah. I mean, it does, but I'm also thinking just in terms of the efficiency of the procedure. You know? So for example, you mentioned in the paper that many of the programs it generates, they just fail immediately. They, you know, they crash. They're they're not valid programs. Those are kind of easy to filter out if you have an evaluator. And then at the at the opposite extreme, even once you've got those programs, say running the real world test case involves performing experiments and clinical trials or a robot trying to navigate a complex field that may end up in a getting damaged or something like that. So there's this huge gap between kind of automatic verification and expensive. And I'm just wondering how can we, you know, how can we bridge that in some sensible way for systems like AlphaEvolve? Curious, like, you have any thoughts on that.

Alexander Novikov 01:03:08

Yeah. We are thinking about those things, and and I think 1 1 thing that Matte alluded to before with, like, this proving, like, finding proofs, which is binary rewards, and then you can try to ask LLMs for feedback to get, like, here again, like, some sort of shape rewards to to drive you towards the solution. I think that that that is a direction we're also thinking about because indeed, like, many cases, you do have this kind of ladder of, you know, more and more expensive evaluation. Like, for example, if if you want to do some sort of simulation based as maybe maybe for robotics, maybe for some, like, biology or physics or whatever. Right? Can do some simulation based reward, and then, like, you ultimately want to trade try it in real world, or maybe it's, like, going to be, again, trying on the robot. Maybe you you go to the lab, whatever. Like, you do want, like, for every stage of this evaluation ladder, which is more and more expensive, you don't want to have not just trying all the things that you found that worked on the previous stage, but maybe have some sort of prioritization mechanism, which

Keith Duggar 01:04:07

Right.

Alexander Novikov 01:04:07

Can can be, you know, can be a lambda back, can be something else, can be like Elo's course. Yeah. Those are the things we're we're thinking about.

Keith Duggar 01:04:15

Makes sense.

Alexander Novikov 01:04:15

Maybe 1 project which also recently came out of Google, which kind of is very good at this is coscientists, and they basically attack the same problem. Right? Like, how do you how do you score ideas which are very hard to evaluate maybe with hard hard feedback, and and I think they're doing a really good job at that.

Keith Duggar 01:04:39

Excellent.

Matej Balog 01:04:40

And maybe just 1 thing to add is that technologically, I think AlphaEvolve is already well set up for this. We have this idea of evaluation cascades where you evaluate a large number of programs very quickly and then a smaller number of programs you can spend longer to evaluate. So you can just expand this cascade all the way to the regime where maybe you can only afford to evaluate 10 things you actually have to go into the real world and run some real world experiments. But in principle, the the mechanism is there. And, I mean, this is what you have to do even if you try to solve the problems manually or what actual researchers do. Like, you you have a finite budget. So first, you try to filter the ideas using cheaper methods, and then you only have the budget to to try the most promising ideas on the most expensive evaluation. And I feel if we can mimic that.

Alexander Novikov 01:05:26

Yeah. Exactly.

Keith Duggar 01:05:27

Yeah. Same way those are done. Yeah.

Tim Scarfe 01:05:28

So I'm curious about LLMs. I mean, of all, let me give you my heartfelt congratulations on Gemini Pro. It is

The Role of LLMs, Recursive Self-Improvement, and Future Directions

Tim Scarfe 01:05:39

ridiculous. It's so good. Honestly, it's it's it's just I have no words to describe how good it is. But we're in an interesting time, right? Because in in AlphaEvolve, you had an ensemble of Flash 2, I think, and Pro 2. And it just raises so many questions to me. What would happen if you did it with 2.5? Presumably internally, you have even better models that that we don't know about yet. And how much how much uplift is there? And and if you think about it, there's this Pareto curve of models and Google's models are on the Pareto curve. So, you know, we're trading off cost and latency and and performance and and whatnot. And if you were just using 1 model, there might be some optimal place on that Pareto curve in AlphaEvolve. I mean, maybe maybe we should have a small model but we should, you know, sample it loads of times or maybe we should have a big fat model but not sample it as many times or with an ensemble, maybe there's some perfect distribution of of of those models. But we really seem to have, especially with the reasoning versions of the models, we seem to have unlocked something which maybe wasn't there until relatively recently. Could you talk to that?

Matej Balog 01:06:45

Yeah. So 1 thing I can say specifically that is particularly exciting to be about AlphaEvolve is that it does get this uplift from improving the the base language model. So this was not necessarily the the case in fund search, but in alpha evolve, we do see this and we actually have a, like, a quantitative confirmation again in the ablations that if you also include the the the 2 pro in in the ensemble, then you get better results compared to just using the flash model. So it's it's very clear that we are, like, leveraging the kind of the the frontier capabilities of these models. And and, of course, we cannot guarantee into the future what will happen, but at least for now, we are definitely, like, riding the wave. So so, indeed, we are we are very keen in seeing how the base models improve and what kind of uplift the method is, is going to get, with this together. The other kind of related point I would just quickly mention is that AlphaEvolve also offers this opportunity, which we haven't taken it, but just forward looking, that it's a it's a system that is able to enhance the capability of the base model. Like, the the capability of the base model is somewhere, but then this system makes it even better through orchestrating this, like, test time compute pipeline and actually so much better that you can make, like, a new scientific discovery. So it raises the natural question, like, can we some-

how distill this improved capability back into the base model? So that's something that you would get if you were to close the reinforcement learning loop. It's not something we have done with AlphaEvolve, but that possibility is is clearly on the table.

Keith Duggar 01:08:16

So maybe you just mentioned improving the base model, and you also mentioned in the paper, you know, using AlphaEvolve to improve the infrastructure of of AlphaEvolve and and the base models itself. So so you finally now managed to close the recursive self improvement, you know, loop. I don't know. That's gonna trigger some folks. Any any thoughts on Yeah.

Tim Scarfe 01:08:39

What would Schmidhuber say?

Matej Balog 01:08:41

At this point, we want to be like very specific about what we have done. Like, we have we have found a way to speed up the training of the next version of Gemini by by 1%, and we have been able to unstrand resources in in the in the Borg data center. So currently, if you think about the the feedback loop, it's maybe on the order of months. Right? Like, when we you speed up the training of the next version of Gemini, then that will take some time to to actually arrive. But indeed, like, we we do see the steps in the in the direction that you that you described.

Keith Duggar 01:09:13

Alex, any thoughts on the recursive self improvement of AlphaEvolve?

Alexander Novikov 01:09:19

Yeah. I mean, I don't think I have to I have anything to add. Like, as Matej said, it's it's interesting to see how it how it pans out, and we are seeing some signs of, you know, helping the Gemini training ground. So, yeah, that's exciting.

Tim Scarfe 01:09:32

Well, in that in that thread, though, we want to have more autonomy. So so right now, this is very much a kind of didactic exchange between the human supervisor. We select the problems. We design the evaluation functions. We seed the solutions and so on. And I just wonder, what would the next step of autonomy look like, you know, in terms of maybe we could actually have the thing imagine what its own evaluation function is and maybe it could kind of go several steps further? What would that look like?

Alexander Novikov 01:09:57

To be honest, from my perspective, like, I guess automating some things is is cool and exciting, but also at the same time, would even lean towards less automation. And I like it, like, I think the the thing that makes AlphaEvol so cool and powerful is this kind of this back and forth between humans and and machines. Right? And, like, the humans ask questions, so the the system gives you some form of the answer, and then you, like, improve your intuition. You you improve your question answering question asking ability. Right? And you ask more questions. So we are thinking a lot about providing access to alpha evolve to academics as trusted testers, like, seeing what they can do with it. And while kind of trying to build that and trying to build the UI, we were thinking a lot about not just implementing the the thing we have as a website, but just like what would be the next kind of level of human and I AI interaction here? Like, what can the humans do in the like, intervene in the process? Like, maybe they want to if the humans would want to supervise the process, like, I don't know, comment on ideas and, like, inject more ideas and things like that. Like, we're exploring that a lot, and I think it's very exciting to see, like, what can be done in this kind of symbiosis space.

Tim Scarfe 01:11:13

So fine final question. You said in the paper that I think I think you mentioned on the order of a 100 compute hours to evaluate a new solution. That was in section 2.3. But just before everyone at home rushes to reimplement AlphaEvolve, could you just give us some examples maybe on the the matrix multiplication thing? I mean, how many compute hours, you know, did it run for? How how much is it realistically costing to do this? Yeah.

Matej Balog 01:11:38

So so 1 nice feature of AlphaEvolve is that it is really elastic, so it can match the difficulty of your problem. So if you ask it to solve a problem that's actually not that difficult, I mean, maybe it's still an open problem, but no 1 has really worked on it, then maybe even if you ask a chatbot, it would almost solve it or solve it. In that case, AlphaEvolve will also give you the answer basically immediately, and it will not cost a lot at all. But if you ask for, you know, like a very difficult problem, maybe, like, decades open, decades long open scientific problem, then you do expect that it's a difficult problem. You will need to spend more time playing with different ideas and iteratively building on top of that. And the nice feature of AlphaEvolve is that it is able to sustain the scaling, and as you keep running it for longer, find better and better ideas. I like, I know it kind of maybe sounds trivial, but I don't think it's actually easy to build these systems that are able to sustain this continual improvement without plateauing at at some point. And and in in this case, with alpha evolve, you you see this elasticity kind of stretching all the way to to making new scientific discoveries. So I think that that's a nice feature of the system. And so to answer your question, like, concretely, it depends on the difficulty of the problem. So if the problem is difficult, then you do expect that you'll need to investigate more ideas and spend more compute. If it's easier, then you can have the answer very quickly. So if you think about the problems that we actually presented in the paper, then even within matrix multiplication, some matrix sizes are much easier than others. So the the compute would be vastly different. And the same goes for the open problems in maths. Like, some of them are fairly easy. Some of them are very difficult. So there is there is no no single answer that would, yeah, position you on this on this spectrum without knowing the the problem. And and unfortunately, as is the case with open problems, often you don't actually know a priori how difficult it is. So you can't even predict ahead of time. And, sometimes you try and you don't find anything better. That can also happen.

Tim Scarfe 01:13:39

Awesome. Well, guys, it's been such an honor having you both on MLST. Thank you so much for joining us

Alexander Novikov 01:13:44

Thank you. So for inviting us. Yeah. Awesome.

References

[1] **AlphaEvolve Blog Post**
https://deepmind.google/discover/blog/alphaevolve-a-gemini-powered-coding-agent-f
Blog Post The official Google DeepMind blog post announcing AlphaEvolve. Subject of the entire interview.

[2] **AlphaEvolve Blog Post**
https://deepmind.google/discover/blog/alphaevolve-a-gemini-powered-coding-agent-f
blog_post

[3] **FunSearch**
https://www.nature.com/articles/s41586-023-06924-6
Paper The predecessor to AlphaEvolve. Used LLMs to evolve code for the cap set problem. Published in Nature.

[4] **FunSearch: Mathematical discoveries from program search with large language models**
https://www.nature.com/articles/s41586-023-06924-6
paper

[5] **Matej Balog**
https://x.com/matejbalog
Person Google DeepMind researcher and co-lead on AlphaEvolve. Guest on this episode.

[6] **Tufa AI Labs**
https://tufalabs.ai/
Company Sponsor. New research lab in Zurich focused on reasoning and AGI, founded by Benjamin Crouzier.

[7] **Matej Balog**
https://x.com/matejbalog
person

[8] **MAP-Elites**
https://arxiv.org/abs/1504.04909
Paper Quality-diversity evolutionary algorithm by Mouret and Clune that inspired AlphaEvolve diversity preservation.

[9] **MAP-Elites: Illuminating search spaces by mapping the performance of solutions**
https://arxiv.org/abs/1504.04909
paper

[10] **Alexander Novikov**
https://x.com/SashaVNovikov

Person Google DeepMind researcher and co-lead on AlphaEvolve. Guest on this episode.

[11] **Alexander Novikov**
https://x.com/SashaVNovikov
person

[12] **Strassen Algorithm for Matrix Multiplication**
https://en.wikipedia.org/wiki/Strassen_algorithm
reference

[13] **AlphaTensor: Discovering faster matrix multiplication algorithms with reinforcement learning**
https://www.nature.com/articles/s41586-022-05172-4
paper

[14] **Borg: Large-scale cluster management at Google**
https://research.google/pubs/large-scale-cluster-management-at-google-with-borg/
paper