/mlst

The ARC Prize 2024 Winning Algorithm [Daniel Franzen and Jan Disselhoff]

March 23, 2026



Machine Learning Street Talk

Podcast Transcript

Revision: a6970998

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

Solution Architecture and Initial LLM Approach 3

LLM Capabilities and Spatial Reasoning Discovery 5

Test-Time Training and Data Augmentation Strategy 9

Depth-First Search and Token Sampling Methods 10

LLM-Guided Search and Solution Validation 12

Symmetry Augmentation and Scoring Mechanism 13

Model Intelligence and Performance Characteristics 15

Tokenization and Numerical Processing 17

DFS Token Selection and Probability Thresholds 23

Model Size, LoRA, and Catastrophic Forgetting 24

Training Infrastructure and Optimization 28

Search Tree Analysis and Failure Modes 31

References 35

Overview

Daniel Franzen and Jan Disselhoff – the 'ARChitects' – won the ARC Prize 2024 with co-researcher David Hartmann, achieving a remarkable 53.5% accuracy on the ARC challenge using a 12-billion parameter language model. Filmed at Tufa Labs in Zurich, they walk through their solution architecture in detail: how they tokenized grid-based visual puzzles as text and fed them directly into an LLM, why test-time training on evaluation examples gave a major score boost, and how depth-first search over token probabilities outperformed greedy and beam search for solution generation.

The conversation gets into the surprising computational capabilities of language models on spatial reasoning tasks. The team found that LLMs could handle 2D grid reasoning without explicit positional encodings, that symmetry augmentations served as a powerful validation mechanism rather than a training aid, and that the model's second-best solutions were often conceptually correct – just wrong in a specific detail like rotation direction. They also discuss why their fine-tuned 12B model outperformed much larger foundation models, the role of LoRA in preventing catastrophic forgetting during test-time training, and what the entropy distribution of their search trees reveals about how the model represents uncertainty across different task types.

Solution Architecture and Initial LLM Approach

Tim Scarfe 00:00:00

Jan and Daniel, it's amazing to have you on MLST. Welcome both of you.[1][2]

Daniel Franzen 00:00:02

Thank you for having us here.

Tim Scarfe 00:00:04

So you guys are the official winners of the ARC challenge. The results were finalized, I guess, about a month ago or

Jan Disselhoff 00:00:12

so. Mhmm.

Tim Scarfe 00:00:12

Yeah. And why don't we just start by, you know, tell us a little bit about your solution.

Jan Disselhoff 00:00:19

It was a process with lots of ups and downs. Like, we started with a simple LLM fine tuning and had pretty good results. But during the competition, we tried to add additional computation steps outside of the LLM, helped us to increase our score substantially. Our LLM alone reached a score of 41 points, I think. But with the additional external help, we could get LLM to improve substantially to like 56 points. But they didn't count. So 53.5 points at the end.

Tim Scarfe 00:00:57

So can you talk us through the overall architecture?[3][4]

Daniel Franzen 00:01:00

Yeah. So we actually started with a basic LLM that was trained on language. We started out with a rather big model. It was a 12,000,000,000 parameters model.

[1] Daniel Franzen — person
[2] On the Measure of Intelligence (ARC Benchmark) — paper
[3] The ARChitects: Winning ARC Prize 2024 Solution — Paper
[4] The ARChitects: Winning ARC Prize 2024 Solution — paper

And what we essentially do is we just take the ARC tasks, which are essentially located on a grid with pixels of different colors, and we just, have 1 token for each color, and we tokenize them line by line, and we then put them directly into a LLM. We do no, program search or other pre processing. We just convert them to text and put them into the LLM. And that worked amazingly well, better than I did expect. And of course there are quite a few tricks we need to do to make it work better on these tasks because the reasoning capabilities of LLMs without essentially fine tuning on the specific task are not so good. So we essentially did some test time fine tuning. So we had essentially 2 training processes. We first, did a long pre training on the official ARC training set. We later replaced it by Re-ARC, which can generate more tasks, more examples. And this model did not perform so good on the validation tasks. So what we do is test and training. We do another training process on the examples of the validation set, which we get during inference without the final challenge, which we then try to predict with the model. And that gives a big improvement to the score. There were quite a lot of other improvements that we tried during the challenge. Some did work very well, but there's also a lot that didn't really turn out well.

Benjamin Crouzier 00:02:34

Tufa Labs is a new AI research lab I'm studying in Zurich. It is funded from past ventures involving AI as well. And so we are Swiss version of deep seek. So a small group of people, very, very motivated, very hardworking. And we try to do some AI research starting with LLM and o-series style models. What we're looking for now is chief scientist and also research engineers. You can check out positions at tufalabs.ai.

Tim Scarfe 00:03:01

MLST is sponsored by CentML, which is the compute platform specifically optimized for AI workloads. They support all of the latest open source language models out of the box, like Lama, for example. You can pay on consumption essentially, or you can have a model which is always working or it can be freeze dried when you're not using it. All of the models that they deploy support the, OpenAI API specification out of the box, which means it's just a 1 line change in your application to switch over to CentML and start saving money and make your application go faster. Yeah. Must have been a fascinating process kind of trying all of these different ideas and seeing what stuck and what didn't. I've updated so much. So, I mean, Melanie Mitchell had quite an interesting paper out talking about her concept arc and, you know, what it what it means for LLMs to understand and all of this. And the the common sense rationale was very much that language models will never be able to do this. And the only way it could possibly work is if we generate an intermediate program because symbolic things are better. They're more compositional. They generalize better. And certainly, think if you took GPT-four and just did 0 shot, you know, solution space prediction, you were looking at around 10%, which is non 0. Certainly much better than something like GPT 3.5. But many of us, myself included, just ruled out the possibility that these things could be reasoning. What did you discover on that journey?[5][6]

LLM Capabilities and Spatial Reasoning Discovery

Jan Disselhoff 00:04:25

We started I at least started with the same assumption. But we quickly found that the LLMs have have far more computational capability than we thought. For example, all tasks seem like 2 d vision tasks. Right? They are on a 2 d grid. If you see them, you use your perceptual, knowledge to move blocks or move pixels around. And my my heuristic was an LLM is on 1 d, like a text text. We have slash n for go to the next line. And it seemed crazy to me that the LM just functions in that space. Like, has to learn to infer the 2 d structure of the problem without ever working in 2 d. And the more we trained the networks, the more it seemed like that just wasn't a problem. The LLMs were strong enough, had enough capability to infer this structure implicitly. And we did some experiments where we tried to help the LLMs, giving them more information about the structure like appending dimensions or coordinates. But it wasn't

[5] Robustness of Analogical Reasoning in LLMs — Paper
[6] Robustness of Analogical Reasoning in LLMs — paper

needed. It rarely improved the score. And if it improved the score, it did so by a very small amount that didn't wasn't worth the additional compute time, basically.

Tim Scarfe 00:05:47

So I think 1 of the intuitions there is it's a 2 d grid. I mean, certainly in transformers models, you know, we we put in positional token tokens and and stuff like that. So the remarkable thing is that it generalized to different grid sizes. And you're saying without you needing to explicitly give it any, like, compasses or markers or anything like that?

Daniel Franzen 00:06:08

We actually tried to do a 2 d positional encoding on the data, but it didn't really improve the results. So it just learns to detect the end of line markings and to output lines with the correct length. It just worked fine. And we then stopped with the precision encoding approach because it was too complicated and didn't give us any advantage.

Tim Scarfe 00:06:31

You did fine tune the model. I think you ended up using a was it Alarmor 8,000,000,000? Is that is that what you're using?

Daniel Franzen 00:06:39

That's what we actually started with. Then we went to a 12,000,000,000 model. And then, because this model was too big to do much things on Kaggle, we actually went back to a smaller model. But it was the LAMA 3.23 b model, which we found to be essentially as strong as the 8 b model, but being much faster in execution. So we could try a lot of more things.

Tim Scarfe 00:07:04

Yeah. I'm very excited about that new LAML model. It's amazing how much they've got gotten out of it. So just with that model, out of the box, 0 shot, you know, in solution space, what kind of accuracy are we talking about?

Jan Disselhoff 00:07:16

Without training?

Tim Scarfe 00:07:17

Without any With pre

Daniel Franzen 00:07:19

any training, I think close to 0 or 0. Interesting. The model can't even predict the structure really. Sometimes it's just outputs end of sentence tokens indefinitely. Sometimes it predicts some numbers like it should with end lines, but then some lines are too long, some lines are short. It hardly produces any solution that is even possible without any training process.

Tim Scarfe 00:07:45

Very interesting. So so you you trained it on Re-ARC, which is Michael Hoddle's dataset. Maybe talk a little bit about that. It was very useful to have a dataset that is basically unlimited. Like, we[7][8]

Jan Disselhoff 00:07:57

have the 400 tasks on the easy dataset. And rearg helps to generate more examples for each challenge. It is in a way, at some point you saturate because you don't get novel ideas in these tasks. But it was very useful to just have virtually infinite generatable training data.

Tim Scarfe 00:08:19

Very cool. So Michael Hoddle's thing is a dataset generator. So given a task, can create a whole bunch of additional, you know, augmentations of that task. So you get about 10,000 of those across all of the the ARC tasks in the in the public set. And you fine tuned, let's say, Lama 3,200,000,000 on that. And and then 0 shot, you know, solution space out of the gate. What are we looking at?

Daniel Franzen 00:08:47

Oh, let me think for a moment. The LAMA model, I think we are maybe about 10% to 20% performance, I think. Interesting. I'm not sure, because we did some multiple inference steps and something. So you could get the 0 shot model with some tricks to maybe 20% or 25%. That was the max we could get out, I think.

[7] Re-ARC Dataset Generator — Tool
[8] Re-ARC Dataset Generator — tool

Jan Disselhoff 00:09:11

On the on the evil data. Yeah, the evil data. Well,

Tim Scarfe 00:09:14

that brings us to the magical test time inference or, you know, there are lots of different names for it. Yeah. People have been using different names for it. But tell us about that.

Daniel Franzen 00:09:24

What we actually did was, after the model was fine tuned the first time, we did an additional training step on the evaluation data. So not on the full evaluation data, because we only had the examples and not the challenge itself. But the examples essentially look the same than the challenge. So you could just pick 1 output from the example as the final example, and let's just train on these inputs. On the CAN contest, we essentially did a retraining on some of these examples. And after that, we did the inference step. And this improved the results quite a lot.

Tim Scarfe 00:10:03

I suppose you could argue that's I don't know whether you would call that cheating or not, but in the real world, I suppose, you would be doing an online prediction. And what you're doing is you're making your predictive model as good as it would have been at the end of a line of doing online prediction.

Daniel Franzen 00:10:18

You're correct on that, yeah. But we tested different versions, and it's also possible to essentially retrain on every single task. So that would essentially be a fairer version, so the model doesn't see the other evaluation tasks. That works just as good, but in our experiments it required more time to retrain than just putting in all the tasks at the same time. And because time was so limited in this challenge, we choose the option to train on multiple tasks at

Test-Time Training and Data Augmentation Strategy

Daniel Franzen 00:10:51

once.

Tim Scarfe 00:10:51

Yeah, so there's this thing that is transductive active fine tuning, which is when at test time, you have the specification and you generate augmentations from the specifications and then you fine tune on those augmentations and then you do the prediction.

Jan Disselhoff 00:11:09

Do you mean from test? Do mean from the from the from the new challenges?

Tim Scarfe 00:11:13

Yes. So you you get let's say you get 3 new specifications for a task. And then do you augment that and fine tune on it?

Jan Disselhoff 00:11:21

Yeah, we use we heavily use augmentation in basically all our inference steps, also in training. We also use it for for the pre training in VR actually. But it helps a lot to get enough data. During the retraining or the test time training, we are heavily data limited. Right? Because each challenge only has 3 examples. But we can use the symmetry of the problem because all the problems work on a 2 d grid and there are some implicit biases in it. So we can rotate it, can mirror it, we can even shift the colors in specific ways without changing the challenge actually. And since the model is not rotation invariant, right, it's 1D LLM, We can create the rotated augmentations and use them to train the challenge too. And it looks like a novel task for LLM because it's rotated differently, And which helps a lot with the amount of data we have.

Tim Scarfe 00:12:26

So from the test specifications you do symmetry augmentations and fine tuning. Do you do any additional augmentations?

Jan Disselhoff 00:12:34

We were thinking about a lot of augmentation stuff. Most of our tests didn't pan out or were too time intensive. Like, for example, we have a selection step where we take the generated example, like our solution, and then we do an augmentation again. Like we ask the LMM basically, does this solution look correct from a different perspective too? And theoretically you can do more augmentations there. Like you can shift into another space where the problem looks simpler, as long as there's some consistency in the solution. Like for example you could many many problems have the same solution if they were in black and white instead of in color for example. And in theory you could use that to to filter wrong solutions very easily. But we didn't have the time

Tim Scarfe 00:13:31

to get it to work basically. So other than the symmetry augmentations there were no additional augmentations?

Jan Disselhoff 00:13:37

Symmetry, color shift and shifting the examples because the transformer is not invariant to order of the examples. So we also shuffled examples a little bit around.

Tim Scarfe 00:13:49

Very interesting. So then we get to the we're getting close to putting the, you know, the pin in the middle of the dartboard here, which was like the main thing that you guys did. So you do some kind of a search and evaluation process as part of your test time inference. Can you tell me

Depth-First Search and Token Sampling Methods

Tim Scarfe 00:14:08

about that?

Daniel Franzen 00:14:08

There are actually different methods of sampling for an LLM. You could just do greedy sampling. This means also you always take the token with the highest probability when sampling. Or you could do sample in a stochastical way by just taking it according to its probability that's predicted by the network. And all these sampling methods didn't work really well if we want to generate lots

of different candidates. So what we actually did, we implemented a selection process later on, which we can maybe talk about later. So the focus here in this generation step was essentially to generate lots of candidates to increase the chance to find the correct solution. And, yeah, we first started out with sampling and, we also tested beam search because often there are different paths through the network which, give different, predictions and 1 of them is correct. But in the end, we settled for a depth first search, which we implemented ourselves because we haven't found it anywhere. And that worked quite well. And it had quite some advantages over beam search. So what we do is, we just, treat the tokens that are predicted from network as a search tree. And we then search through this tree to find all solutions which has a sampling probability above a certain cutoff value. So, for example, we set this probability to 10%. And then we get a variable amount of candidates. This could be up to 10 for 10%, of course. This search has quite some advantages. The first 1 is that it's very memory efficient. You only need to store exactly 1 path. So it's memory efficient as a single inference step, and much more efficient as beam search roots. It needs essentially the same memory for each beam. And the other advantage is that it gives us multiple solutions at once for the problem. And the third 1 is that we have a cutoff. So if there are paths that are not promising, so in these cases, the search stops early and just doesn't even follow them. And this gave us a big efficiency boost in generating solutions, or solution candidates, I would say.[9][10]

Tim Scarfe 00:16:44

This is so fascinating. I interviewed these guys at the University of Toronto and and they were talking about the the reachability space of language models. And they had this famous Roger Federer game, which was like, you know, you try and you make the language model say, great. Roger Federer is the is the greatest. And you have to kind of find what prompt will reach that word. And their their theorem essentially is that the capacity or the reachability of of transformers is incredibly flexible. There's a remarkable divergence between the way that we sample tokens from a language model and what they're capable of. And, you know, like a clearer way of saying that is that there is a misalignment between how we sample language models and the epistemic truth or the correctness of the program that we are sampling. And I find that fascinating. It's also related to this creativity problem, which is that when we do this greedy sampling, which means we just take the best token and the best token and the best token, we're kind of missing out on all of these creative possibilities. And this is something that a lot of people aren't really thinking about yet.

[9] Search Methods in Language Models — Paper
[10] Search Methods in Language Models — paper

LLM-Guided Search and Solution Validation

Jan Disselhoff 00:17:52

I think that's a it's a very interesting problem. And we have to distinguish between like a language context and the ARC context. It's that there's a substantial difference, namely that we only have 10 possible tokens that might be the next correct answer. And also we know there's only 1 correct answer. So in language you can have millions of ways of rephrasing the same idea and all of these are valid or good continuations. In our case, which is 1 of the reasons why the DFS works so well, we only have 1 correct continuation and we have to find it. And it's very easily evaluatable, and it's very easily searchable also because the the set of possible continuations is much much smaller. We can even calculate how probable each solution is by simply taking solution, doing forward pass, calculating the logits, summing them, and then we know the cutoff needs to be this low for our DFS to find a correct solution guaranteed. And, that makes the problem very tractable in this sense. And all the sampling algorithms are built for language, because LLMs work on language, obviously. Which is also why we didn't find the DFS, because we to reimplement it ourselves, because it's it's not a valid way of sampling a language. It just can't work. It's the language has too much possible paths, and each path, as long as it's valid, should roughly have the same probability. So if you have 1,000,000 possible continuations, each probability is 1 by 1,000,000,000. So it's very hard to use it in this context. But because our problem is discrete, small, we can sample very, very efficiently.

Tim Scarfe 00:19:45

This raises so many interesting questions, because you could argue then that the reason why these models are misaligned between program or knowledge correctness and how we sample them is by your argument because there are too many degrees of freedom in natural language. So, you know, it's almost like they've they're dealing with this ambiguity in their training. Therefore, when we sample from them, we're not very likely to get the correct answer.

Jan Disselhoff 00:20:12

You have to be careful about that, because we have to differentiate between the probability of a path and the probability of the answer. You can have a very improbable path that leads to a specific answer, And then you can have multiple variations of it. The probability of the answer is not dependent on any path, but on the sum of all paths that lead to this answer. So in language you have this broad set of paths that all lead to the same solution. And when you use standard sampling techniques, you sample the final answer correctly. Like that's the that's the use of sampling. In the ARC contest there are no alternative intermediate steps. Right? So you don't need to marginalize over all possible paths basically. You can just you need to get the correct path basically. So in normal LLMs this is completely fine, but I think in the age of reasoning models we get into like o 1 or o 3, you also need the correct reasoning steps, the correct intermediate steps to get to the right solution, especially if you use it for code generation or answering complex mathematical questions. So this will be I think in the future this will be a much larger problem. But up till now sampling was simply the correct approach.

Symmetry Augmentation and Scoring Mechanism

Tim Scarfe 00:21:53

A couple of things on that. So certainly perhaps in your solution the way that you traversed that sparse space and the reason I say sparse is it's a bit like in reinforcement learning that you need to take several steps without knowing what the value is going to be And this is really really difficult when we have some kind of traversal optimization algorithm where we rely on a monotonic signal. So we're getting better better better better like we're in in this case the space is sparse and we have to take several steps and enter the unknown. But the thing that really interests me though is is even if we don't know what the reasoning steps are, Ryan Greenblatt's solution really impressed me and surprised me because I assumed that this is just an exponentially large problem and that using a language model to guide the search would not be significantly better than just exhaustively finding programs. And what he demonstrated quite, you know, succinctly is that it's tractable. I mean, yeah, still need to generate loads of completions and do some search. But a reasonable amount of search, gets you to the solution. And what 1 saying there is that even though there is an apparent orthogonality between correctness and searching this space, the correct solution isn't that far away. That's fascinating. Yeah. Think the interesting thing about Ryan Greenblatt's solutions from my perspective

is that he showed that code generation can work, like that the the models can see the visual problems, like the 1D representation of the grids, and then produce code that works to move these 2 d objects even though they were never seen, even without fine tuning. On the other hand,[11][12]

Jan Disselhoff 00:23:36

code generation has a massive advantage compared to our approach because you can just test your code. Like, you can check does the code that's generated perform in the way I want it to perform? And if it does, then it's probably a good solution. Whereas our approach doesn't have that. Like, we had to find a way to select which candidate we want to submit without the possibility of having any guarantee that it's right.

Tim Scarfe 00:24:07

Well, yeah, this gets us on to the most delicious part of all. So as you just articulated, Ryan Greenbat's solution was technically a neuro symbolic architecture. So, I mean, Sabaro Kambahati would have been delighted because, you know, he loves having formal guarantees. He loves actually knowing for sure. I mean, actually, in this case, we don't know for sure because many correct programs are not correct, of course. But but but at least, you know, we can run a program on a Python interpreter and we can get a yes or no answer. It might be a false positive. Now, you guys have done something very interesting. You're you're using the language model to verify its own value as you traverse through this tree structure. You can you tell me about that?

Daniel Franzen 00:24:50

Yeah. So, our generation process essentially generates multiple solution candidates. It could be usually it's up to 10 or 20 in practice. And now we need to find a way to select the correct 1. And what we actually did is we used our model also for judging how good these solutions are. And the interesting part is, it's difficult if you do it without augmentations, because a model of course favors its own predictions, because what we use as core is the sampling probability, probably if we got it in the first place. So, what we essentially do at this point, we also use a lot of augmentation for the judging. So we essentially use 16 different augmentations of the problem and put them through the model to calculate scores for each augmentation. And then we averaged them. And this is an interesting part. We tried different algorithms. 1 was just, for example, to sum up the actual probabilities. And the other 1 was to sum up the logarithmic probabilities, which would essen-

[11] GPT-4 Code Solutions for ARC (50% SOTA) — Paper
[12] GPT-4 Code Solutions for ARC (50% SOTA) — reference

tially be equal to multiplying the probabilities. And the second 1 worked much better than the first 1 in selecting. And, yeah, we really asked us what why this is. And, so our hypothesis is essentially that the good, the correct solution looks mostly correct from every angle. No matter how you rotate it, it gets maybe not a really high score, but it doesn't get an extremely low score. And the incorrect solutions, they might also look good in some perspectives. But usually there's 1 or 2 or even more perspectives where they get an extremely low score like 0.01% or something in probability. And so essentially these low scores are important when doing the calculation. And you can't get these scores during the generation process, because these things are never sampled. So we actually need the extra scoring step to calculate these low scores to sort out the false solutions.

Tim Scarfe 00:27:03

This

Model Intelligence and Performance Characteristics

Tim Scarfe 00:27:04

is absolutely fascinating. So you generate 16 augmentations for, let's say, a test specification. And as you said before, you had already fine tuned the model on symmetry permutations, which means it's it has awareness of different symmetry transformations. And you're saying that using it to do self reflection on the value of those symmetry augmentations gives it different perspectives on what good looks like. And then you can kind of look at those, you know, look at the coherence of the different symmetry perspectives to give you an idea of which 1 is correct. Yeah. Exactly. Ironically, it's very useful to us that LLM does not have

Jan Disselhoff 00:27:48

symmetry in its predictions. And it's the interesting part is that it's useful to us that the language model does not have the 2 d symmetries. Like, if it had 2 d symmetries, we couldn't use our augmentation for scoring because the score would be the same every time. So in a way, the abused effect that LLM is not perfect at 2 d tasks, Also because it's it's always generating from left to right, from up to down. Right? So some parts of the puzzle get generated later with more information And some parts get generated earlier with less information. And if we rotate it, we generate other parts of the solution first. So a lot of the score differences between the different augmentations come from the fact that it can generate maybe complex parts or parts of the solution first or later. Like, it helps sometimes that a solution depends on previous input. And if we generate

the wrong solution and then rotate it, it sees it much earlier. Like it can see, ah, it makes no sense that like the line that goes from left to right here ends at an empty pixel. It should never happen. And because we we rotated the problem, it can see it from a new perspective and then decide that it's a invalid solution or very improbable solution.

Tim Scarfe 00:29:25

How much does this depend on the the symmetry augmentation before test time? So so did you guys say that before we even get to any test time computation we take all of the evaluation tasks, and then we do some symmetry transformations and we fine tune. If you if you don't do that, how does that affect this evaluation?

Jan Disselhoff 00:29:47

We have never tested it.

Daniel Franzen 00:29:50

So you mean, not doing the augmentations during training or?

Tim Scarfe 00:29:53

Yes.

Daniel Franzen 00:29:55

Yeah, we have never tried that out. We always use augmentations from the beginning.

Tim Scarfe 00:29:59

So I'm a bit confused, because you seem to be saying it's actually a good thing that it doesn't know about the augmentations. So doing the augmentations, it seems to me that that would be a good thing, but maybe it isn't a good thing.

Jan Disselhoff 00:30:11

It's a trade off. Like, we would lose a lot of training data and it's also it it is good if it can just generate the correct solution from every perspective. Right? But some problems are simply easier from some perspective than others. And in theory, the network should converge to a correct solution from every perspective. Yeah. If it's deep enough, if it's strong enough, but sometimes it just can't. And then we can use that fact by generating

Tokenization and Numerical Processing

Jan Disselhoff 00:30:43

different augmentations and scoring them correctly.

Tim Scarfe 00:30:46

Trying to understand as well, so that there are different perspectives on on language models. So, Sabaro Kambhati, he called it jagged intelligence, think, which is what Andre Kapathi called it. You know, that and that's this idea that point holistically, they're they're actually very good, which which means they have these islands of knowledge. So you can say, I'm gonna do a symmetry transformation and art. It it knows. So it it it given an example of something, it knows knows if it's good or bad, but it doesn't know how to get there. So so there seems to be like these concentric circles where they are better at discriminating than generating, and they also are capable of knowing if they don't know how to get there.

Jan Disselhoff 00:31:28

I think in some ways, yes. It depends it depends so strongly on the task, like always. Right? There are problems where it's true that they have a much easier time of discriminating the solutions than generating them. But at the same time, it depends how you measure it, basically. Like, we measure the score of a task or of a solution by calculating basically the probability that it would be generated from normal sampling. So in our case, generation probability or generation and evaluation is exactly the same. The only way we can then use the discriminatory abilities of the LLMs is by using these transformations. Because like Daniel said, LLMs prefer their own outputs. So we have to we have to trick the LLM to think that it's a output it would have never done.

Tim Scarfe 00:32:26

There's also this thing that Thomas Dietrich was telling me that it is very interesting looking at the divergence between epistemic risk and alliteric risk and that that's simply that there's a divergence between things that are true and things that the language model will kind of like, you know, will give you if you stochastically sample them. And and he said that before RLHF, there is much more of a correspondence between the likelihood of a trajectory and epistemic factfulness. So whether the thing is actually true or not, and RLHF deranges that. But I'm thinking that doesn't really affect your situation because you're doing fine tuning on the top. So you're kind of I mean, what you're doing is is either over-

writing or orthogonal to any RLHF training. So you're you're actually deliberately saying, you know, here is what good looks like. And if you can generalize from that example of goodness, you can use the likelihood of a trajectory as a proxy for, you know, correctness.

Daniel Franzen 00:33:30

We actually used a LAMA and sensor model in our experiments, and we also compared it to just a normal model, and the uncensored model was a little bit better. So maybe that makes a little difference.

Tim Scarfe 00:33:47

Was it retrospectively uncensored, or was it just never RLHF trained? Retrospectively, it was from Schwann? Yeah. Yeah. Yeah. Because I've I've played with uncensored models, you know, like the the the the Gwen the Gwen models. Mhmm. And I found that they are lobotomized. And for whatever reason, like, however they are uncensored, think there are various different approaches to to uncensoring them. They just seem to degrade to the performance of a model, you know, half the size. So like a 72,000,000,000 model will now have the reasoning abilities of a 32,000,000,000 model, but that's just my experience.

Jan Disselhoff 00:34:26

We are actually not sure if it's at all important that we use a pre trained model. Like 1 of our experiments in the pipeline at the moment is just resetting all the weights and then training to check if we even use the pre trained language capabilities or if it's just the architecture, actually.

Tim Scarfe 00:34:45

That's very interesting. Yeah. I I interviewed Randall Bellastriero at NeurIPS and he made the same discovery. So he's got this really interesting paper out. And, you know, we are taught that we need to have these large pre trained models and we're leveraging the the knowledge inside the model and so on. And he apparently, no 1 has really done this experiment that you can just kind of and this is for discrimination, not generation. Right? So if you're just doing some kind of classification problem, you can take a very large model. You can train it from scratch for doing some, like, discriminative task, and you can actually get better performance than, like, frontier models pretty pretty much with hardly any training and hardly any data. Isn't that and it almost goes back to those papers from years ago, you know, there are so many inductive biases just in the architecture of the model itself that, you know, you don't actually need that much training.

Jan Disselhoff 00:35:37

I think 1 of the interesting parts is that we relobutomize our LLM too. Like, at the end, we have an LLM that has basically only 140 tokens or something. Like, we remove all language capabilities from the model, just delete it, And really make sure that it can only argue in the space of ARC. Like it can only output numbers 0 to 9, end line tokens and some letters. And we did so mostly because of computation requirements, like it saves surprisingly a lot of, RAM. But here, we we remove all language capabilities. So we are often asked, did we do any pre prompting? Is it like chain of thought thinking? It can't. It's all gone.

Tim Scarfe 00:36:27

Well, that's really interesting because it's certainly the Minds AI team, I was talking with Jack, and, you know, they're they're they're very bullish on this idea of, like, multimodal or, you know, cross domain transfer. So they, you know, I think Jack was saying, for example, they might even give English language descriptions of the ARC challenges, and they're looking for as many ways as possible to leverage the base model in in the language model. You know? And there are 2 views on this. Like, 1 view is that these things are a general intelligence. And the whole point of o 3, for example, was that it's kind of like it's creating a novel skill program by doing some kind of compositional generalization, and it's like, you know, working in this novel situation. Whereas this this alternative view is very much that they are a jagged form of intelligence and very specialized form of intelligence, And they're mostly just working on specific instances of very similar things that they've seen before.

Jan Disselhoff 00:37:23

Indirectly, I think we could say that we found that the model performed much better at tasks where it had seen similar tasks. Like in our paper, have claimed a performance of, I don't know, 76%? 72. 72% on a left out part of the evaluation dataset. But its performance drops dropped at the time noticeably if we never trained on any evaluation dataset tasks. And we believe that to be because there is some conceptual leakage, we called it, from some tasks of the evaluation dataset to other tasks. Like, are not the same, but they have similar ideas. Which is probably also a reason why the additional data sets like concept art or bark helped, because they maybe introduced some novel ideas in some way. And if the LLM has never seen an idea conceptually in training, then it's very unlikely that it will perform well on it. So for example, if we train the LLM on arc challenges that are, I don't know, fill flood fill algorithms or something like that, then it will not transfer its skills into a counting task or something like that. But as long as it has seen some counting tasks, it might be able to combine the ideas to form, like, novel challenges.

Tim Scarfe 00:38:54

It is very interesting that I mean, just take multiplication, for example. Frontier models, they don't generalize very well. You know? I mean, they do a little bit, you know, 3 digits, maybe 4 digits or something like that. But you can take GPT 2, and you can fine tune it up to 20 digit multiplication, and and it works better than any frontier model. So there's always this notion that doing fine tuning for a specific task is is better. Architecturally, that doesn't work very well. Right? Because, you know, we're we're in this world and we're subject to novelty all the time and we can't possibly be fine tuning in every possible situation. So maybe that's when this active fine tuning, like this test time computation comes into the play. Because increasingly, build architectures where we are learning online and maybe sharing that knowledge in a clever way to to other models. But it it it kind of feels that with this new online version of of prediction, we can kind of have our cake and eat it.

Daniel Franzen 00:39:51

So the interesting thing is for the multiplication, tokenization plays a big role because the models tokenize numbers differently. And the model we used was using, for some digits, just 1 token, but sometimes it combined digits to 1 token. So essentially, there were digits 0 to 9 it could tokenize. But then there were some 2 digit numbers and some 3 digit numbers. And, yeah, I think newer models do this differently. I'll probably have all 3 digit numbers integrated. But for us, this was a problem in ARC. So we also removed all this from the tokenizer. So we just have single digits, because this would have been a big problem, I guess, if the length changes of the outputs.

Tim Scarfe 00:40:43

Oh, interesting. So you modified the tokenize tokenizer to remove all of the compounded numbers. We essentially removed anything

Daniel Franzen 00:40:50

except numbers, end of line, and end of problem, and input and output prefix.

Tim Scarfe 00:40:59

Very interesting. Yeah. And I know, of course, you weren't deranging it because you were doing the training on the top. So you're you're kind of like doing a higher resolution type conversion of of training.

Jan Disselhoff 00:41:08

Exactly. Yeah. And I think for the multiplication task too, it's a common problem I see on Twitter is that people misunderstand how complicated a problem is for LLMs. Like, I think OpenAI has almost every 3 digit number as a token. So if you tell ChachiPT to multiply 2 3 digit numbers, it has to basically remember 1,000,000 multiplication pairs. Like it's a very, very complicated task. And then people are like, yeah, it can't multiply 3 digit numbers, but it's a hard task for judgepity. The same with the typical strawberry task, Like in like my my way of explaining it is imagine like every syllable is a token in a letter in a language you have never seen and you have to remember how many r's are in each of these tokens and you have to remember that for 130,000 different things. Like it's a complex task. And then you have to learn it without ever seeing the letter r. It's hard. LLMs are very, very strong problem solvers if you fine tune them, but they are not the intended solution of ARC in the way that we use them. So in a way it's a it's a hacky solution. But because of the challenge, have to use the best performing model. Like you you could do a very creative solution that goes further

into the direction of AGI. Right? But you wouldn't win. You win with LLMs and abusing, like, fine tuning speed and stuff like that. So LLMs are too strong for this contest to enable any other solution, basically.

Tim Scarfe 00:42:54

Interesting. What about the thing that right now, this is quite an interesting coincidence that ARC was designed to be solvable by humans. And 1 of those restrictions is that they shouldn't be more than about 30 times 30 cells on on a grid because otherwise humans would start making loads of mistakes. But that also just almost coincidentally happens to be the upper bound of what we can, you know, correctly do with an LLM. If it was 50 by 50 or even 60 by 60, no 1 would be using LLMs to solve R.

Jan Disselhoff 00:43:26

We would have only solved the smaller problems. Yeah.

Tim Scarfe 00:43:29

Yeah. Because even now, like, when when we look at the failure modes of o 3, there's definitely, like, you you can see it tailing off with solution size. So it starts to make more and more mistakes, you know, when it gets to 30 by by 30. It does raise the question though of the complexity of of the problem. So again, my intuition is that this problem has a kind of exponential complexity to it. And I would have said the same thing about language. It's it's interesting that language, if you think about all of the ways that a reference can hierarchically, you know, connect to each other, it it seems like as the length of of even a book, for example, the the information complexity of a book should be exponentially large because there are all of these, like, connecting things between all the paragraphs and the sentences and the words and so on. And language models don't have any problems understanding and explaining books. And I wanna understand what what what what that is. So there's there's the potential complexity of all of the possible connections between the terms, and then there's the actual embedded complexity of of language. And I think you were making this argument earlier that you feel that actually the sort of the embedded complexity doesn't scale exponentially with the problem size.

Jan Disselhoff 00:44:47

Yeah. It's in many challenges at least. Like, you can can see that, like, many problems are mostly trivial. There are just a few pixels where you have to do the right decisions. We even measured that. And we found that in a intermediate stage of our solutions, where we still did normal sampling, where we didn't use DFS, we found that we had to get the model

DFS Token Selection and Probability Thresholds

Jan Disselhoff 00:45:15

to choose the second highest probability token 3 times, and then we would have solved 80% of the challenges. But we had to get it to choose these, and we don't know which of the 900 tokens the second best is the correct 1, you know. The space of solutions the LLM generates is much much smaller than the theoretical large space because a lot of the pixels are trivial. Like the background is often trivial. Like, many problems are move this object here. And if you start object at the position, then the LLM generates completely the rest of the object correctly. But it has to decide where to put this object, and that's the hard part. But the number of decisions we have to do to generate the correct solutions is surprisingly low.

Tim Scarfe 00:46:09

So does that imply that there's no reason in principle why an LLM wouldn't be able to scale to a 100 by a 100 or even 1000 by 1000?

Jan Disselhoff 00:46:17

I think it's 1 of the reasons why Daniel's DFS solution worked so well for us. Because when you do normal sampling, each token has a probability of being selected wrongly. Like, there are many ways of fixing stuff like that, like min pay sampling, top k sampling or something like that. But we have for every token we generate, we have a small chance of doing it wrong. And our DFS solution just generates all solutions that are above a certain probability. So we are basically guaranteed to get the most likely solutions and some below that, which reduces a lot of the problems with the exponential problems of sampling, as long as our probability bound is low enough. How did you decide on the depth of the search and the threshold?

Daniel Franzen 00:47:13

We decided about the depth by just essentially just testing it out. So how deep can we go to find, in most cases, the correct solution? And it's, of course, a trade off because if you search down to 1%, you can get 100 solutions in theory, though in practice you get much less. And if you search to 10%, you only get 10 solutions, and you get much less solutions. So in some problems where the solution is clear, it doesn't make a difference. You just find the solution. But when the solution is unclear, you get a lot of samples for going down to 1%. And that would have been computationally infeasible on Kaggle. So in the end, it was a trade off that we just tested out, and we ended up with different values between 1017% in the end that we tried in practice. And the solution was mostly in there. Going down from 10% to 1% gave an improve, I think, from 70 to 71% on the score, so not much.

Tim Scarfe 00:48:24

And still technically a variable computational budget, because if you think about it at 1%, when you do it a 100 times, you could get a huge variance in the amount of searching that you're doing. But it's it's kind of vaguely vaguely budgeted.

Daniel Franzen 00:48:40

Is that is that correct? So so it's not clear, beforehand how much how many solutions you will get from the DFS. Yeah. So this can be more or less depending on the problems. And something that was very interesting is: In the end of the contest, we switched back to a larger model, because we fine tuned a lot. And when you do normal sampling, switching to the larger model has a certain factor by how much more compute you need for the larger model, because the inference is essentially just n times something like a compute. But for the DFS, because the larger model got better at deciding what the correct solution is, it could actually prune earlier in the DFS. So it still took a little longer, but not as much longer as we expected it to take from our previous tests. And that was quite surprising, and it also enabled us to do more in the short time, short compute time on Kaggle.

Model Size, LoRA, and Catastrophic Forgetting

Tim Scarfe 00:49:41

Did you guys just for fun at home try a 70 2,000,000,000 model or something bigger?

Daniel Franzen 00:49:47

I tried it on the evaluation set after the contest, but I did just 1 pass. I don't think it was a 70 2,000,000,000 model, but a 32,000,000,000 model or something in that range. But I didn't have time to fine tune it, really, so I ran it with the same parameters and it didn't make much of a difference in the setting. But of course, that's not final because I hadn't really time to tune it.

Tim Scarfe 00:50:15

Yeah. There seems to be an interesting relationship between the strength of the base model and how much test time computation you do. Which is to say there's an argument that it's almost better to do if if if the budget allows, it's better to do more test time computation on a smaller model.

Jan Disselhoff 00:50:33

Yeah. I think so. Yes. I think the going up in size is not necessarily always good in this case. The more parameters you have, the more parameters you have to fine tune. And for fixed compute budget, I think the smaller models are much better. You can generate more, you can evaluate more, you can filter more. But also, I think in some experiments, larger models just didn't perform better. Like, they converged roughly to the same score.

Tim Scarfe 00:51:09

So then there's this notion of adaptability. So let's imagine that you were deploying this thing. Let's say you wanted to make some money, and you deployed a web service called Arc Solver. And I'm guessing what you would notice over time is that if you persisted the LLM and you're always continuously fine tuning it and adapting it and so on, you would probably notice its improved knowledge by virtue of the fact that it was doing less tree searching because it found the solution quicker. Do you think that that would just be the case? That it would just get generally better at ARC and it would just know the solution more quickly? Or do you think you would get weird things happen where it might learn distractors on some tasks and get better get worse at other ones?

Jan Disselhoff 00:51:51

It's a good question.

Tim Scarfe 00:51:53

I mean, part of the reason I'm saying that is I think that that's what's happening with the o 1 models. So I think the reason why we're seeing a dramatic improvement over even in 3 months is because they are learning from the from the users. So they're continuously adapting and augmenting their knowledge, and the models are getting better quicker, needing to do less thinking over time.

Daniel Franzen 00:52:15

Maybe I have an interesting fact to share. So for the fine tuning, we tried different number of tasks we trained on at the same time for the second fine tuning. So we did something with 1 task just on single task, with 50 tasks on Kaggle, and we also tried the full 400 tasks of the evaluation set. And that led to a very much degraded performance. So up to 50 was fine, but maybe for more tasks it just didn't work. Maybe it was too much for the model to store at the same time. I'm not sure. We also use LoRA, which also has limited number of parameters. So if we would do some online training and start with new examples, maybe it would have to forget the old ones to solve the new ones. That is quite possible.

Tim Scarfe 00:53:07

Yeah, because, you know, we are always told in the theory of neural networks that there's a problem with catastrophic forgetting and continual learning and all the rest of it. And, you know, even Cholet wrote in his book that when you do fine tuning, you have to be really careful. You have to, like, turn the learning rate right down because you'll just destroy anything it already knows, and it will only learn the thing that you're teaching it and so on. And we're just becoming quite red pilled on OpenAI now because that that, you know, we we believe that these things are like magical memory machines, and they're just sucking in all of this knowledge. And the more knowledge you give them, the better they get. And that seems to fly in the face of practical experience.[13][14]

[13] Overcoming Catastrophic Forgetting — Paper
[14] Overcoming Catastrophic Forgetting in Neural Networks — paper

Jan Disselhoff 00:53:44

And transformers are just very, good at learning facts, I think. It's frankly incredible what they can store. But there's also a slightly different thing that I find interesting. Like, in our fine tuning, we use LoRAs, obviously, because fine tuning on the whole network is very expensive. And, there's a small subtle thing with LoRAs. If you use weight decay on a LoRA on a network, then it degrades the weights of the LoRA. But you always have the base network below it. So you can't you can't fall below a specific performance. Like you you're always around the correct full network and just move a little bit in another direction. So if you do test and training or fine tuning with a LoRA, you can in a way have the safety blanket of your base model below it, which can be very useful I think in our challenge. Like we merged I think the first after pre training into the weights and then did a test time training on another LoRA. So at the worst case, it would degrade to the base model again. Like the base model from pre training.[15][16]

Tim Scarfe 00:55:06

And just for the audience, can you quickly because LoRA is a low-rank approximation, if I remember correctly. Can you explain how it works? Is it approximation or adaptation? Adaptation. Adaptation, yeah.

Jan Disselhoff 00:55:16

It's a very simple idea, very elegant idea. You have all these quadratic weight matrices in the attention layers, right? And instead of fine tuning all the weights on them, you fine tune to smaller matrices that you can multiply to get the full rank matrix. And and that's mostly enough. Like, if you have a 1,000 by 1,000 matrix instead of fine tuning 1,000,000 parameters, you say 1 1000 by 1 and 1 1 by 1,000 matrix that would be rank 1 and fine tune those 2 small vectors. And because you go down to a smaller problem, because you have a simpler problem you want to learn, it's enough to get the network to go in the right direction, basically. I think we had a surprisingly high rank. Right?

[15] LoRA: Low-Rank Adaptation of Large Language Models — Paper
[16] LoRA: Low-Rank Adaptation of Large Language Models — paper

Training Infrastructure and Optimization

Daniel Franzen 00:56:10

Tried out different ranks from, I think, 64 to 256. And the original model, I think, has a rank of 4,096. And anything above 128 didn't actually make a difference for final performance. We also tried full fine tuning once. It was memory very memory consuming, and that also didn't make a difference for us. This low rank is probably enough for the problem.

Tim Scarfe 00:56:42

And just to just to give folks an idea of how long it took, I mean, how how you know, are we are we talking like half an hour, 2 hours a day?

Daniel Franzen 00:56:51

So competing in the challenge?

Tim Scarfe 00:56:52

As in to to do, you know, like, let let's say at at the beginning when you're when you're fine tuning the model, how long did

Daniel Franzen 00:56:58

it take? Oh, at the beginning, we had, smaller models, and we had also shorter training times, especially when we're still working on the original training set, which we couldn't put in too often without overfitting. So then it was a few hours, maybe 2 to 4 hours.

Tim Scarfe 00:57:16

Okay.

Daniel Franzen 00:57:16

And it got longer and longer during the challenge. So at some point, we were at 2 days on an on an NVIDIA H100, and the final model was essentially 8 days on an NVIDIA H100. Wow. So that was quite long. And the interesting thing is when the heavy ARC dataset appeared, when we found it, there were only, I think, 4 days left till the end of the contest. So what we did there, we actually started with multi GPU training to do the work of 8 days in 4 days in 2 days. So luckily, we got a lot of compute. Lambda offered us to use a machine with 8 times h 100 GPUs, Lambda Labs. And, yeah, so we could still train the model in time and submit it on the last day. That was the model with the 56.5 score, which sadly didn't finish in time then.

Tim Scarfe 00:58:19

Yeah. They said that necessity is another important mention. Yeah. I'm 1 of these guys that, you know, like, what you're supposed to do when you're coding is you add 1 feature at a time, you check it, and then you, like, commit it and so on. It's I I have this kind of not ADHD, not quite, but, know, like, I'm I'm just I've got so many cool things I wanna do, and I wanna just, like, stick them all in there. Where do you fall on that spectrum?

Jan Disselhoff 00:58:42

Normally, I would fall similar to you. But in in the setting of the contest, we just didn't have time. Like, it didn't matter. We had to push like new solutions. We had to experiment quickly. And trying to to clean code or trying to refactor in a way that makes it more flexible would have taken, I don't know, 1 or 2 days. But that was too long. Like, had we had so many ideas we wanted to experiment on and try that we had to use the code base that we had. So when we would have put

Daniel Franzen 00:59:19

in new features again, we would have to essentially refactor it again because, yeah, we didn't know exactly what things we would try out during the time. So that was probably a little bit pointless to refactor it all the time during the contest. It felt a little bit like a mad dash. So there were some refactors. So when it got too complicated, it's like 2 or 3 steps where I completely refactored it in between during the challenge. But in between these refactorings, just used it over and over and just added stuff, essentially.

Tim Scarfe 00:59:51

And would you say your intuitions were fairly correct throughout? Or because you know, like, we get a bit superstitious about things, so we we think, oh, that thing's really important. We need to keep that in there. Were there any situations where you realized to your horror later on that something you thought was good was actually not good?

Jan Disselhoff 01:00:06

Yeah. But sometimes. You were you were constantly, like, testing different things, like leaving some stuff out and putting some stuff in. And we also used a lot of our submissions for testing, like what happens if we remove this feature? What happens if we remove that feature? Often, our favorite ideas that we tried didn't pan out. And then we had to decide, do we invest more time in making them work? Or do we just say, screw it. We we don't have the the time to investigate it deeper. We have to push the things that work. And often, it was the right decision to ignore our crazy fun ideas and just do the thing that works better.

Tim Scarfe 01:00:51

And what were the top 2 things out of all of this stuff that you did that you think moved the needle the most?

Daniel Franzen 01:00:56

So the things that were most important, I guess, in our implementation were, first of all, the scoring process. So scoring with the augmentations, which allowed us to select the solutions reliably. So this was 1 thing that gave a big jump. I think it was from 30 to 37 at that point when we implemented it. And the other thing, think, was the depth first search algorithm. Because it gave us such a performance gain that we could change a lot of things after. Like using a bigger model, doing more augmentations, and things like that that we would couldn't have done without it.

Tim Scarfe 01:01:37

Well, that's interesting because part of experimentation is actually having useful signal. And you're saying that certain features actually unlocked a signal on other things you could try. Yes.

Jan Disselhoff 01:01:46

The DFS was a very natural extension of the scoring, I think. After we had a good working scoring and selection algorithm, the challenge became to generate good candidates. And we we tried a lot of stuff to do that. Some some LLM stuff and some external stuff like combining snippets of different solutions to generate new solutions. But what really worked well was Daniel's DFS algorithm. It was very quick and gave us so many different candidates we could score and very few bad candidates. So, yeah. It the 2 approaches worked very well together. It was a very synergistic approach.

Search Tree Analysis and Failure Modes

Tim Scarfe 01:02:34

And did you ever see any catastrophic failure modes with the DFS where it would just search for an unreasonable amount of time before hitting the threshold?

Daniel Franzen 01:02:43

No, not really, because we limited the search depth to 10%.

Tim Scarfe 01:02:48

Oh, could say that was bad.

Daniel Franzen 01:02:49

It essentially happened that it searched quite a lot of solutions there sometimes. But we set the limit in such a way that it didn't lead to a big problem.

Jan Disselhoff 01:02:59

Because the probability mass has to distribute over all paths. Right? So if the lower limit is 10%, then it can't be more than 10 paths that are valid at the same time.

Tim Scarfe 01:03:10

Do you think it would be possible before you do the depth first search to predict what the computational budget is or to predict what the search tree might look like?

Daniel Franzen 01:03:21

That's an interesting question. I think it might be quite difficult because you don't know until you have actually done some searching. So could probably predict it in an estimation after you've searched a part of it. Or you could go for iterative deepening, like you first go to 10%, then to 5%, something if you want to search deeper. It would also be a possibility if you want to limit the compute.

Tim Scarfe 01:03:52

I'm just thinking you could take the prompt the model with that solution, and then you could just use a classifier or something and train the classifier on different topologies of trees that you found previously. And you might be able to sort of predict a priory what how how much is this gonna take me? You know? How how much computation should I invest in this problem?

Jan Disselhoff 01:04:12

If we have the solution, then that works. Like, you can can just check the the logits for each pixel and then measure basically the entropy, like how distributed is probability. And if the probability is distributed enough then that is a branch in the tree, but then you only have like the 1 path through the tree and then branches from there. Predicting the whole tree is probably basically equivalent to predicting the correct solution in some ways. Like, you have to know the solution to know what the tree looks like.

Tim Scarfe 01:04:45

We didn't touch on this before. So you were doing a minimum entropy search. But what would happen if you tried? I'm not sure if we do a minimum research. It's a little bit like, you know, Monte Carlo tree search. So it's not doing a minimum entry, but like if you look at the the entropy distribution, it's like it's it's searching the space of low entropy.

Jan Disselhoff 01:05:06

The problem is the problem here is, I think, that we have we have like a you can imagine we have a rank order for each pixel for the next prediction. Like pixel color 1 is most probable, then pixel color 0. And for basically every problem, there's never more than 3 possible continuations. So, you have to be very careful if you like to tweak it or try different things that you don't get into the the 7 things that are basically impossible. Like, for many problems, are, for example, only 3 colors. And if you do a maximum entropy or a very broad search, you would get colors that are just impossible. And you have to safeguard against it. But at the end, every smart algorithmic improvement we tried performed worse. Like, the LLMs are just smarter than us.

Tim Scarfe 01:06:01

There must have been some relationship between you you stick a candidate solution in there and you've got this entropy distribution and you're saying in in some circumstances it's quite it's quite pointed. So it's like the the model will say I want to go in this direction and no other direction and sometimes it's spread out. What can you infer from that?

Jan Disselhoff 01:06:20

In general, I think the pointed ones. Like, if you have very few candidates, they are often just correct. Like, the model has a very good idea of how the problem is structured. So if there are several solutions, then they are mostly because there are some link missing in the understanding of the problem or if the model just doesn't understand the problem at all. Like, did a lot of testing. And the problems that are hardest for models were often counting or I think Flood FloodFl worked very well, but mostly counting or size estimation problems. And in that case, because the model was very unsure, it predicted often a large amount of possible candidates. In the other cases, when when when there are few candidates, then they're almost always correct, I think.

Tim Scarfe 01:07:23

And how did you deal with false positives? So I guess there are some situations where you found solutions which, you know, now now when you look at the the sum of the log probabilities, you know, like, it's telling you this is the solution. And it might just not be the solution because it's not the solution or it might be a false positive because it's, you know, like the right solution for the wrong reason or whatever. What do those failure cases look like? A pin strongly linked problem, I think.

Jan Disselhoff 01:07:50

1 of the most interesting things I've seen in analysis was a task where you have color shifting. Like, do you have 4 colors in the upper left corner, and the goal is to shift the colors in an object clockwise direction. And model solves that problem. The second best solution is also a color shift just in the wrong direction, which I found very fascinating because it means that the model conceptually understands the problem, and even the second best solution is a conceptually right solution. It just goes in the wrong direction. In the cases where the model has a false positive or doesn't solve the problem, it's almost always still conceptually in the right direction. Like it understands I should fill the space with something or I have to generate an object here. But sometimes it just doesn't know how set which object. Sometimes it fills the wrong space, stuff like that.

References

[1] **Daniel Franzen**
https://github.com/da-fr
person

[2] **On the Measure of Intelligence (ARC Benchmark)**
https://arxiv.org/abs/1911.01547
paper

[3] **The ARChitects: Winning ARC Prize 2024 Solution**
https://github.com/da-fr/arc-prize-2024/blob/main/the_architects.pdf
Paper

[4] **The ARChitects: Winning ARC Prize 2024 Solution**
https://github.com/da-fr/arc-prize-2024/blob/main/the_architects.pdf
paper

[5] **Robustness of Analogical Reasoning in LLMs**
https://arxiv.org/html/2411.14215
Paper

[6] **Robustness of Analogical Reasoning in LLMs**
https://arxiv.org/html/2411.14215
paper

[7] **Re-ARC Dataset Generator**
https://github.com/michaelhodel/re-arc
Tool

[8] **Re-ARC Dataset Generator**
https://github.com/michaelhodel/re-arc
tool

[9] **Search Methods in Language Models**
https://arxiv.org/html/2408.00724v2
Paper

[10] **Search Methods in Language Models**
https://arxiv.org/html/2408.00724v2
paper

[11] **GPT-4 Code Solutions for ARC (50% SOTA)**
https://redwoodresearch.substack.com/p/getting-50-sota-on-arc-agi-with-gpt
Paper

[12] **GPT-4 Code Solutions for ARC (50% SOTA)**
https://redwoodresearch.substack.com/p/getting-50-sota-on-arc-agi-with-gpt

reference

[13] **Overcoming Catastrophic Forgetting**
https://www.pnas.org/doi/10.1073/pnas.1611835114
Paper

[14] **Overcoming Catastrophic Forgetting in Neural Networks**
https://www.pnas.org/doi/10.1073/pnas.1611835114
paper

[15] **LoRA: Low-Rank Adaptation of Large Language Models**
https://arxiv.org/abs/2106.09685
Paper

[16] **LoRA: Low-Rank Adaptation of Large Language Models**
https://arxiv.org/abs/2106.09685
paper