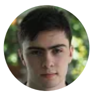
# Getting 50% (SoTA) on ARC-AGI with GPT-4o

You can just draw more samples


RYAN GREENBLATT
JUN 17, 2024

84 31 Share

I recently got to 50% [1] accuracy on the public test set for ARC-AGI by having GPT-4o generate a huge number of Python implementations of the transformation rule (around 8,000 per problem) and then selecting among these implementations based on correctness of the Python programs on the examples (if this is confusing, go to the next section) [2]. I use a variety of additional approaches and tweaks which overall substantially improve the performance of my method relative to just sampling 8,000 programs.

*[This post is on a pretty different topic than the usual posts on our substack. So regular readers should be warned!]*

The additional approaches and tweaks are:

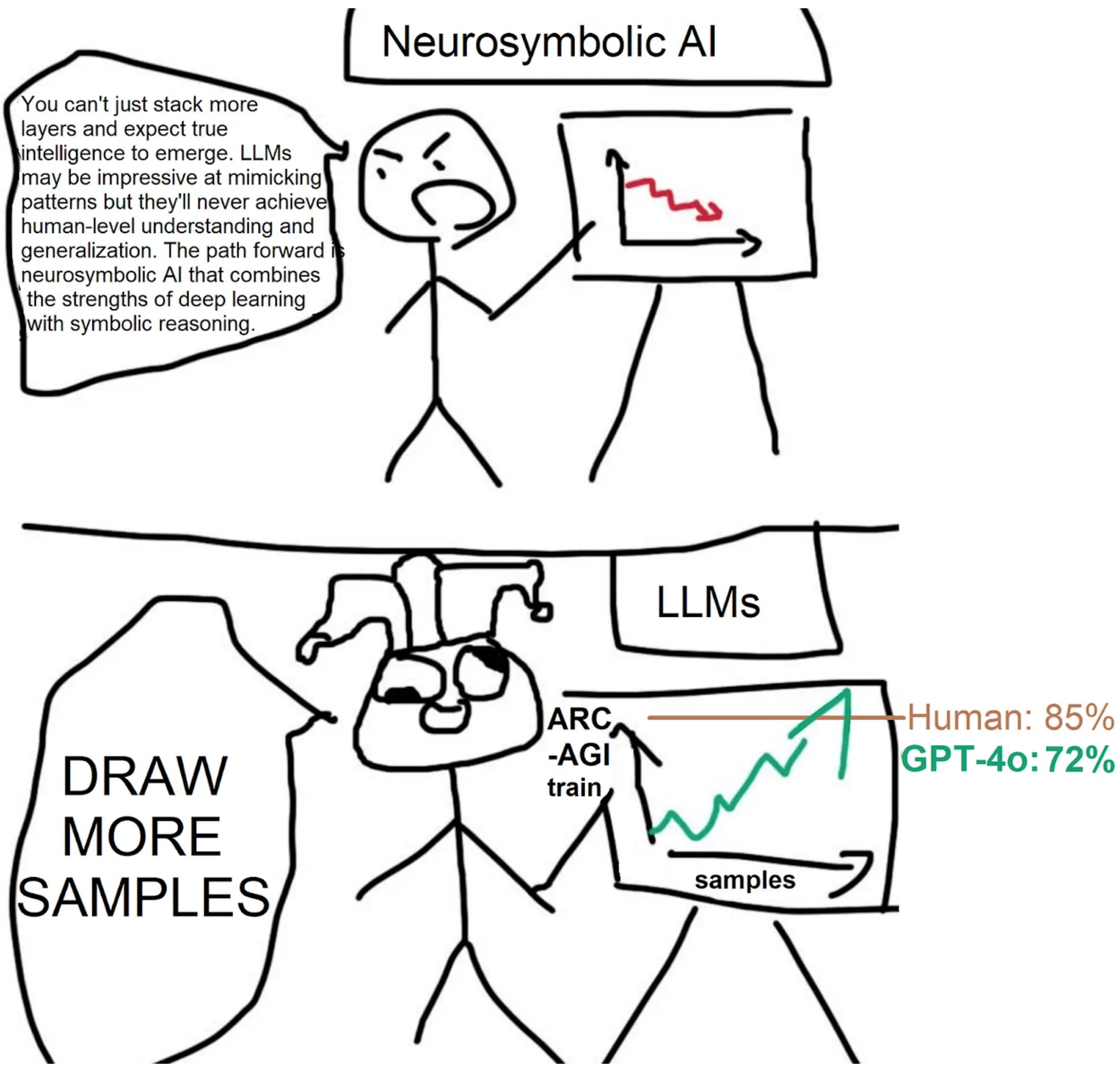
- I use few-shot prompts which perform meticulous step-by-step reasoning.
- I have GPT-4o try to revise some of the implementations after seeing what they actually output on the provided examples.
- I do some feature engineering, providing the model with considerably better grid representations than the naive approach of just providing images. (See below for details on what a "grid" in ARC-AGI is.)
- I used specialized few-shot prompts for the two main buckets of ARC-AGI problems (cases where the grid size changes vs doesn't).

The prior state of the art on this dataset was 34% accuracy, so this is a significant improvement. [3]

On a held-out subset of the train set, where humans get 85% accuracy, my solution gets

72% accuracy. [4] (The train set is significantly easier than the test set as noted here.)

Additional increases of runtime compute would further improve performance (and there are clear scaling laws), but this is left as an exercise to the reader.


Neurosymbolic AI
You can't just stack more layers and expect true intelligence to emerge. LLMs may be impressive at mimicking patterns but they'll never achieve human-level understanding and generalization. The path forward is neurosymbolic AI that combines the strengths of deep learning with symbolic reasoning.
LLMs
DRAW MORE SAMPLES
ARC -AGI train
samples
Human: 85%
GPT-4o: 72%

peter.jensen@biocomm.ai Subscribe

In this post:

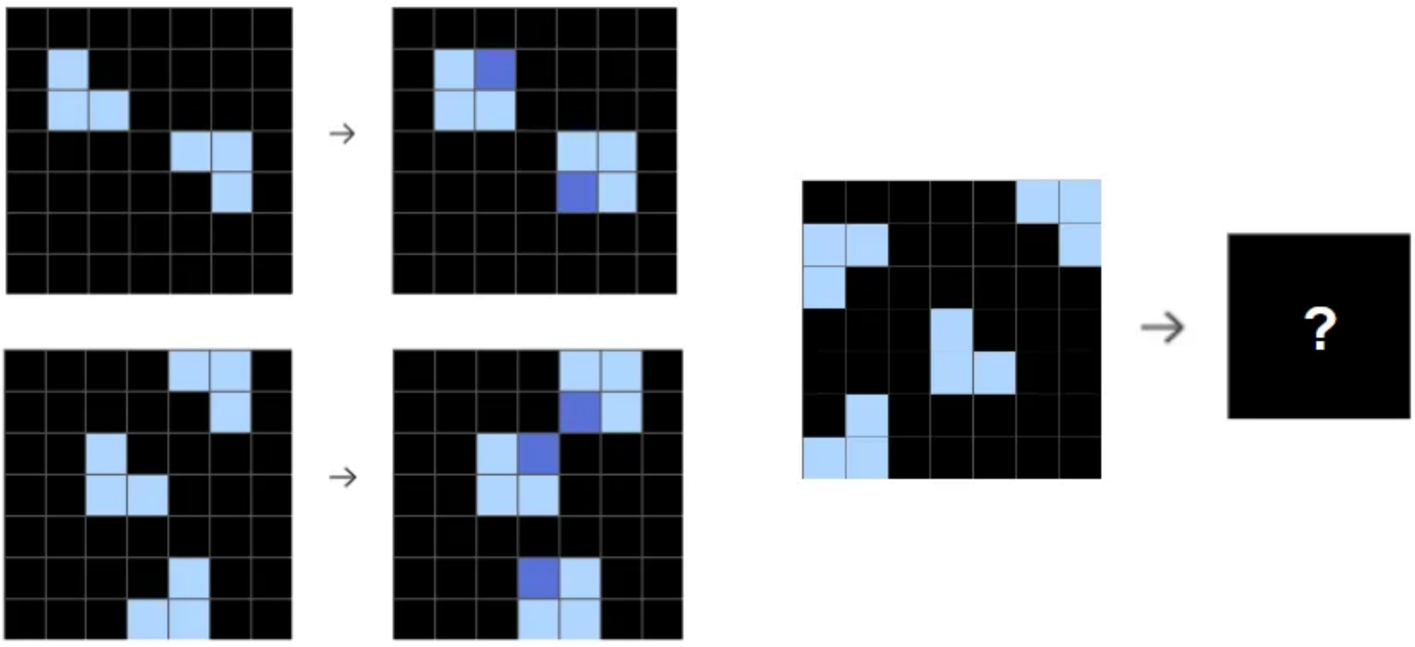
- I describe my method;
- I analyze what limits its performance and make predictions about what is needed to reach human performance;
- I comment on what it means for claims that François Chollet makes about LLMs.

> Given that current LLMs can perform decently well on ARC-AGI, do claims like "LLMs like Gemini or ChatGPT [don't work] because they're basically frozen at inference time. They're not actually learning anything." make sense? (This quote is from here.)

Thanks to Fabien Roger and Buck Shlegeris for a bit of help with this project and with writing this post.

## What is ARC-AGI?

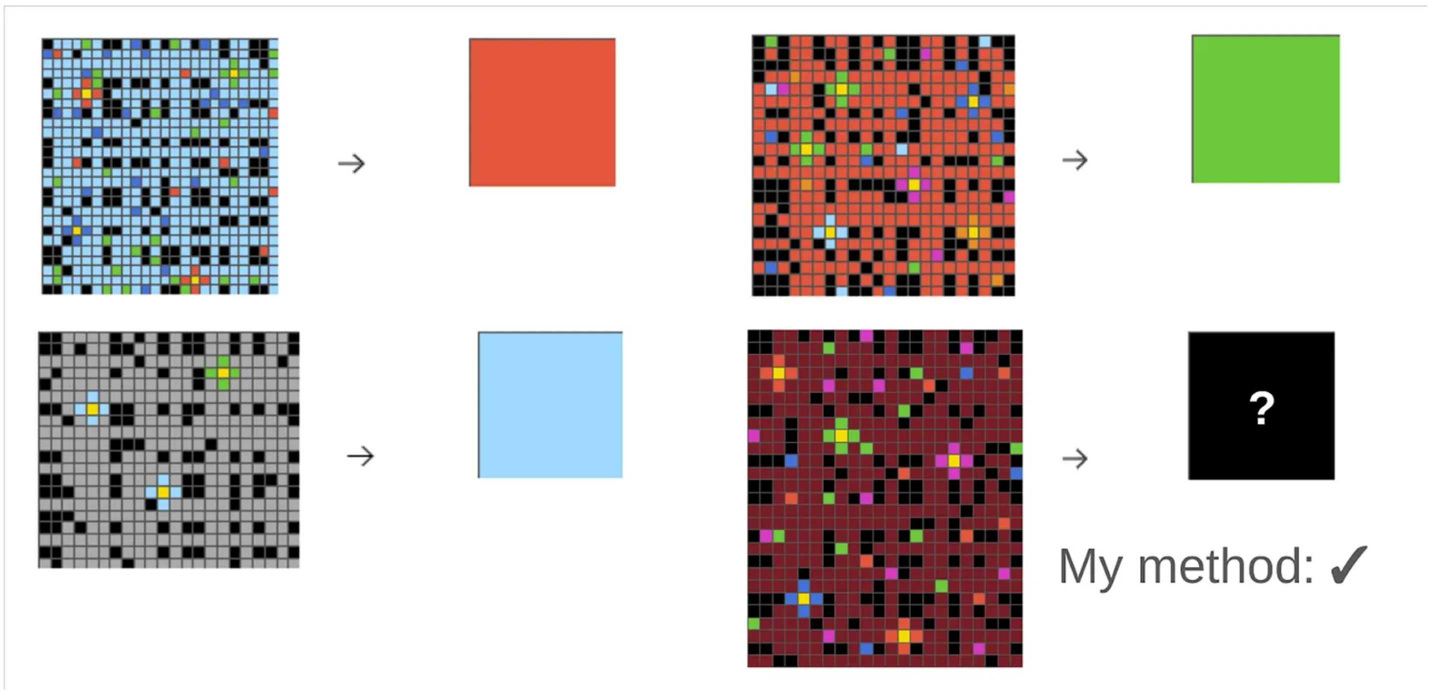
ARC-AGI is a dataset built to evaluate the general reasoning abilities of AIs. It consists of visual problems like the below, where there are input-output examples which are grids of colored cells. The task is to guess the transformation from input to output and then fill out the missing grid. Here is an example from the tutorial:



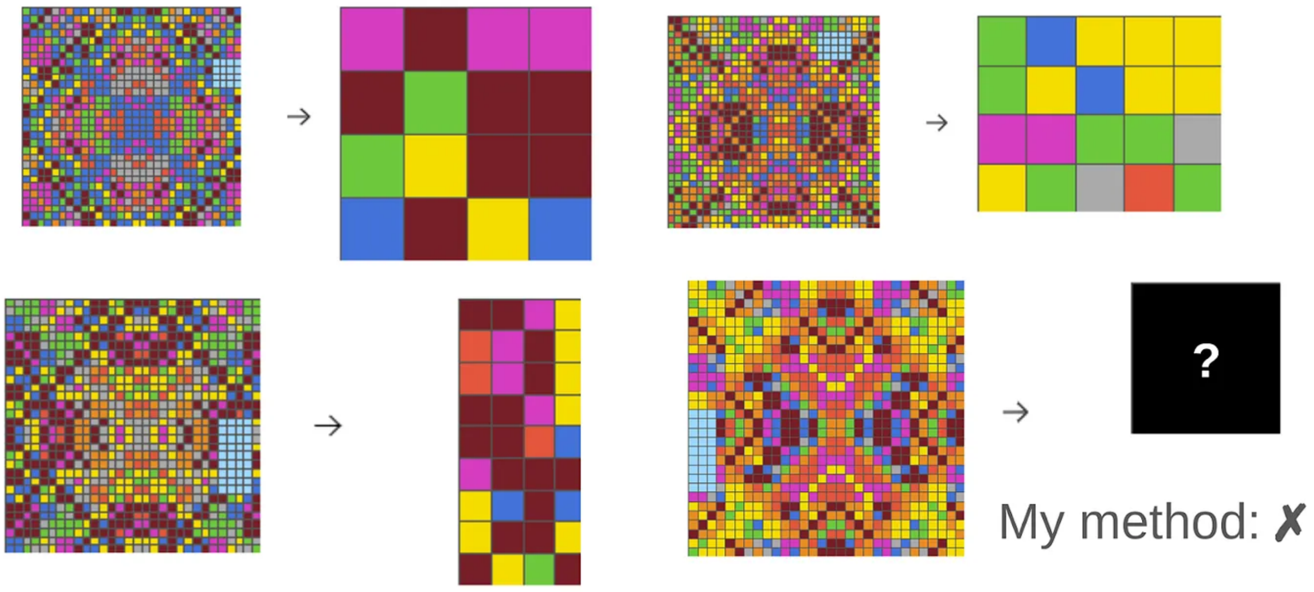
This one is easy, and it's easy to get GPT-4o to solve it. But the tasks from the public test set are *much* harder; they're often non-trivial for (typical) humans. There is a reported MTurk human baseline for the train distribution of 85%, but no human baseline for the public test set which is known to be significantly more difficult.

Here are representative problems from the test set [5], and whether my GPT-4o-based solution gets them correct or not.

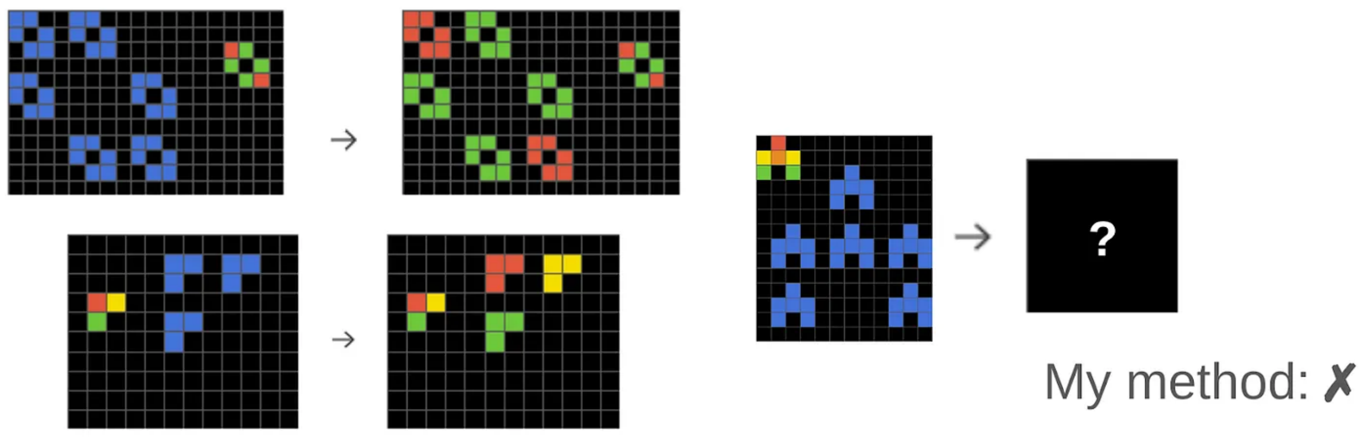
Problem 1:



Problem 2:



Problem 3:



## My method

The main idea behind my solution is very simple: get GPT-4o to generate around 8,000 python programs which attempt to implement the transformation, select a program which is right on all the examples (usually there are 3 examples), and then submit the output this function produces when applied to the additional test input(s). I show GPT-4o the problem as images and in various ascii representations.

My approach is similar in spirit to the approach applied in AlphaCode in which a model generates millions of completions attempting to solve a programming problem and then aggregates over them to determine what to submit.

Actually getting to 50% with this main idea took me about 6 days of work. This work includes constructing few-shot prompts, building better text representations of these grids, iterating against the train set, and implementing various other tweaks to improve performance.

I started on this project a few days before Dwarkesh Patel recorded the recent podcast with Chollet. This was inspired by Dwarkesh talking to my coworker Buck about ARC-AGI, and then Buck being like “come on, surely you can do better than current SOTA using LLMs”. Then, I tested GPT-4o a bit and it seemed to get what was going on. I’ve recently been doing some research that involved getting Claude 3 Opus to do reasoning right at the edge of its capabilities, and thought I might have an edge based on my experience from that project.

At a high level, the method I use is:

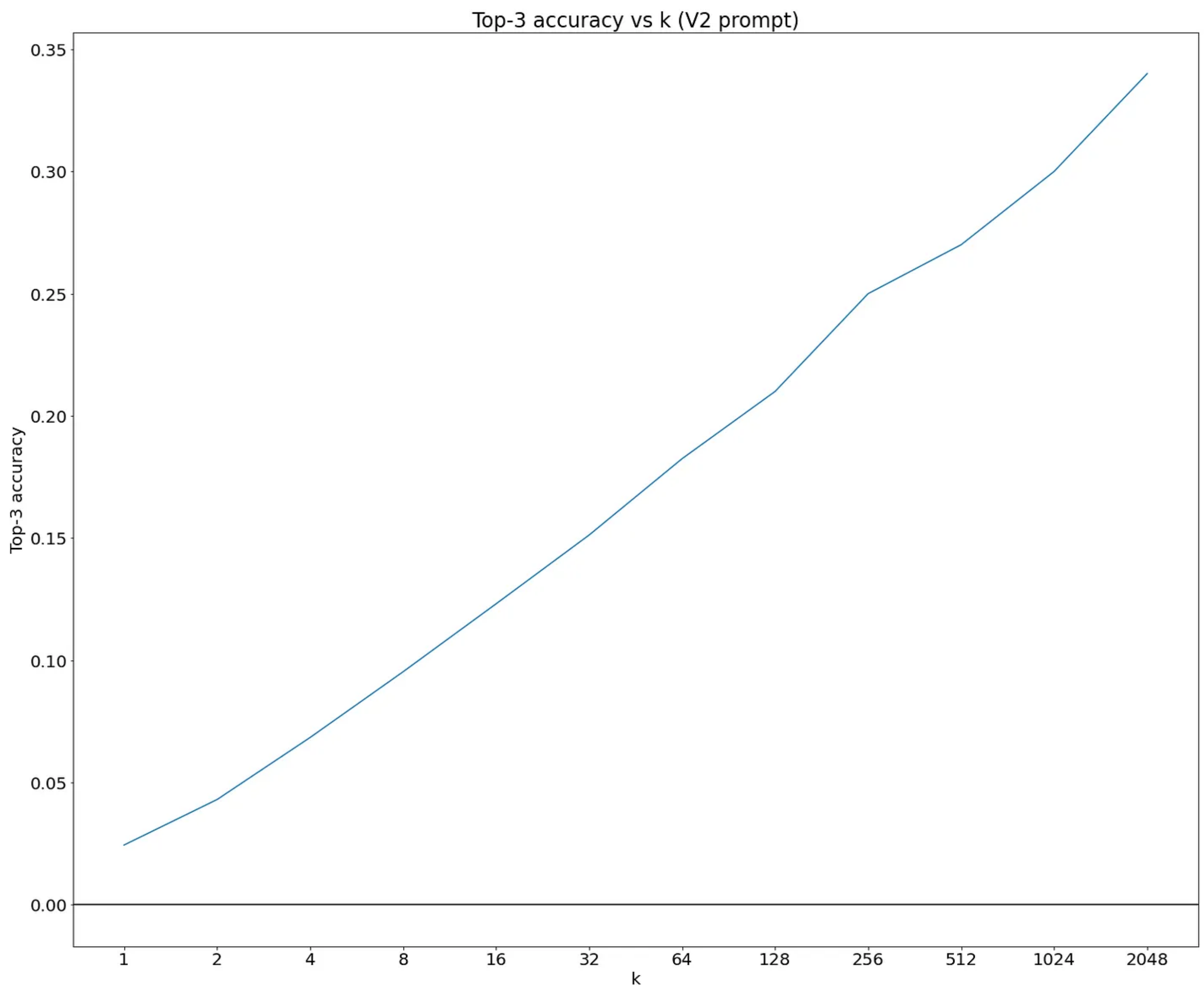
- Provide the ARC-AGI problem to GPT-4o, with both an image representation and with various text representations for each grid in the problem. The text representations include showing which cells are occupied by different connected components of colors and showing diffs between the input and output (in cases where the grid shapes are the same).
- Instruct GPT-4o to reason about what the transformation is, reason how to implement the transformation as code, and then finally actually implement the transformation in code.
- Use a few-shot prompt with several carefully handwritten examples of step-by-step reasoning [6] to actually get GPT-4o to do this reasoning somewhat effectively. The resulting prompt is usually around 30k tokens long including images.
    - We actually have a pair of few-shot prompts: one prompt for when the grid size changes (in any of the examples) and one for when it doesn't.
    - We also ensemble over multiple pairs of few-shot prompts. This doesn't help much, but I already had the samples in the cache.
- Sample vast, vast numbers of completions (~5,000 per problem) from GPT-4o.
- Take the most promising 12 completions for each problem, and then try to fix each by showing GPT-4o what this program actually outputs on the examples, and then asking GPT-4o to revise the code to make it correct. We sample ~3,000 completions that attempt to fix per problem in total across these 12 starting implementations.
    - We use a few-shot prompt for revisions with step-by-step reasoning.
    - We also have a text representation which shows the diff between the actual output and the expected output.
- Then, we select 3 submissions [7] to make based on a majority vote over programs which get the examples correct. (In the case where we don't have 3 distinct submissions from programs which get the examples right, we apply some heuristics to pick a submission, but this doesn't matter much.).

In addition to iterating on the training set, I also did a small amount of iteration on a 100 problem subset of the public test set. All the results I presented here were computed on a different subset of the public test set that does not overlap. The train and test set are not IID, and the test set is both much harder and somewhat qualitatively different (I think), so using a subset of the test set for iteration was useful for quickly getting a better sense of how things change with difficulty. It's unfortunate that these sets aren't IID: it makes iteration harder and more confusing. [8]

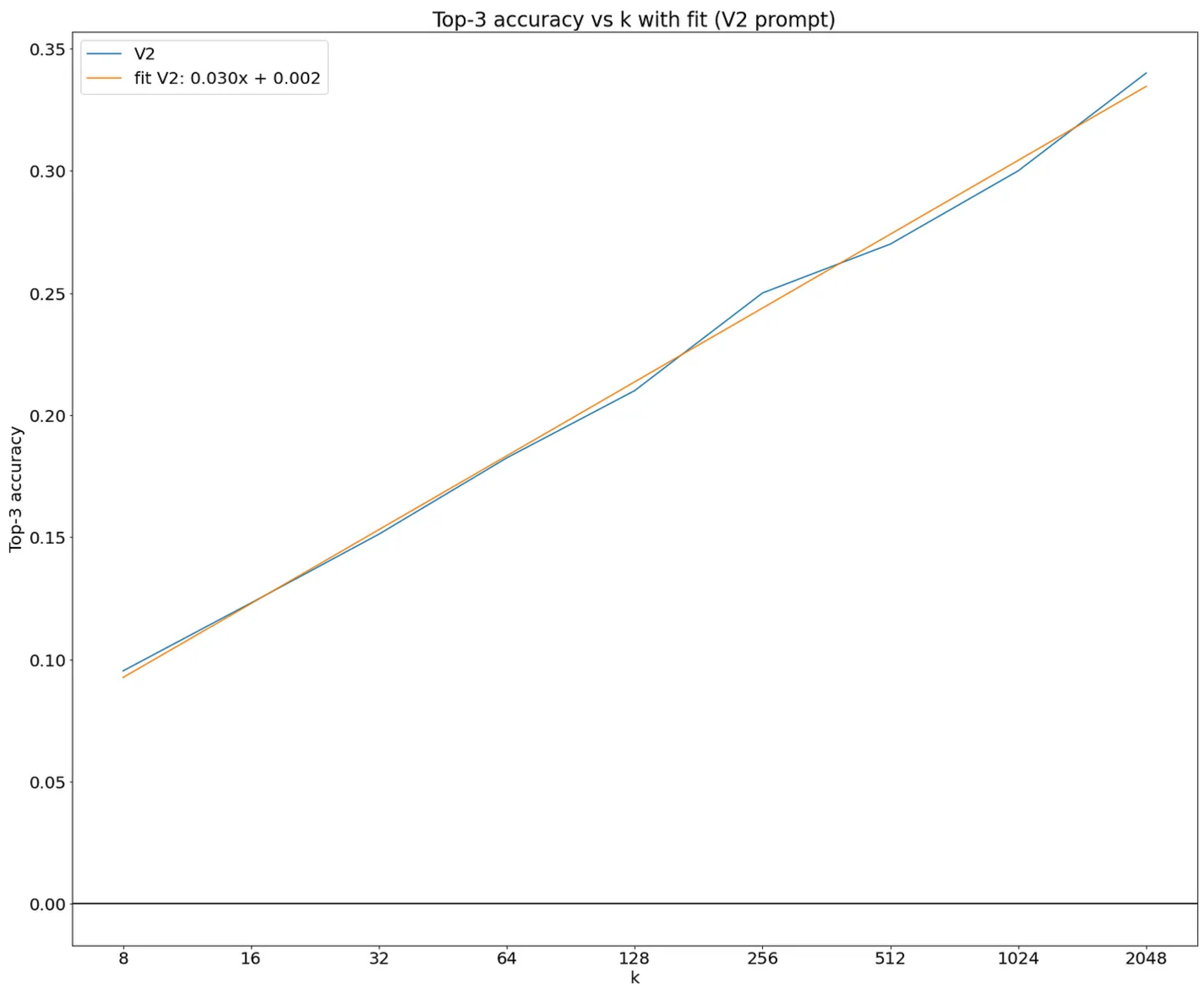
More of the details of my approach and a bunch of tricks I use to improve performance, can be found at the bottom of this post. You can find the full solution in this GitHub repo: https://github.com/rgreenblatt/arc_draw_more_samples_pub.

## What are the returns to more sampling?

We can analyze how more samples improves performance on test. For ease of interpretation (and implementation), I show returns to samples on the primary and most performant prompting variant I use (in the section below, it has the name "V2") [9]. I also don't incorporate revision (as this would require a separate run for each number of samples).



There appears to be a relatively clean scaling law [10]. Of course, a linear fit from log(k) to accuracy can't go on forever as it would imply you eventually go above 100% accuracy! [11]



The fit is in terms of log base 2. So, it indicates an additional 3% correct per doubling of k.

While the number of samples I use is large, using far more samples is certainly possible. (For reference, AlphaCode uses up to a million samples per problem).

## What are the returns to better prompting and code fixing?

I find that improving prompting and adding a revision step is important to improve the accuracy. Huge numbers of samples, better prompting, and a revision step are all essential for the performance I achieve.

Here is a breakdown of the performance by version of the prompt:

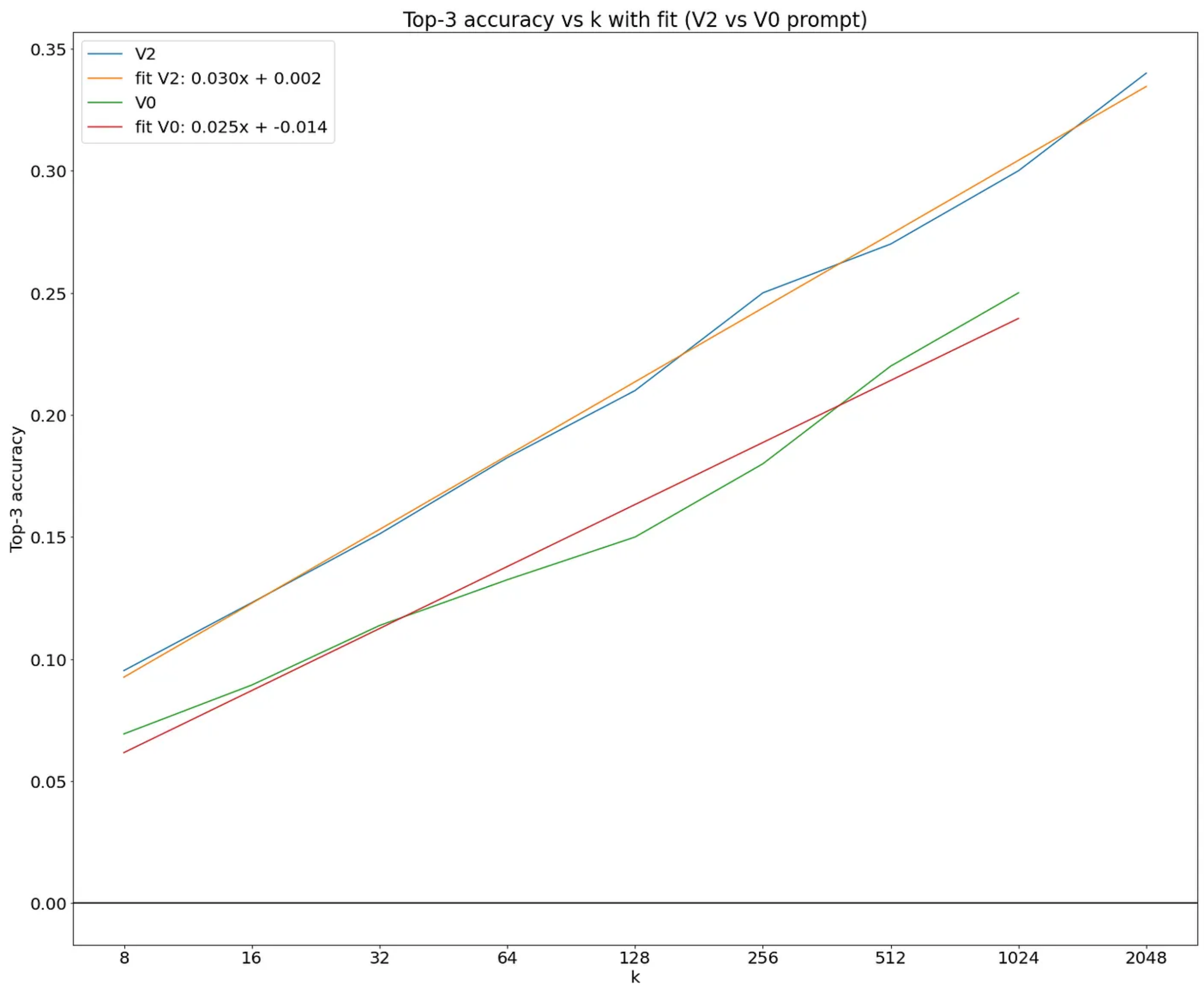
- V0 - 1024 samples/pb: 25% (test set performance). This version doesn't split into two buckets and doesn't use the ascii representations described above. (It does show a simple ascii grid like 8|0|3 where 8, 0, 3 are the colors of that row.)
- V1 - 1024 samples/pb: 27%. This variant splits into two buckets and uses the spreadsheet style ascii representations (when in the equal image size bucket). It uses different few-shot prompt examples (though the examples overlap with V0).
    - V1.1 - 1024 samples/pb: 28%: This is just a variant with somewhat arbitrarily different prompts for diversity and because I thought these prompts might be better. (They aren't really better.)
- V2 - 1024 samples/pb: 30%. This variant includes the ascii diff between the input and output and again includes different examples.
- V2 - 2048 samples/pb: 34%
- V0 (1024) + V1 (1024) + V1.1 (1024) + V2 (2048): 37%.
    - This performance is the same as what I would expect from 4096 samples from V2, so the diversity doesn't seem to help more than the downside of V0/V1/V1.1 being worse than V2.
    - To reach 37% with just V0, I would need around 43,000 samples! (Based on the fit I show below.) So, using better prompting and representations makes a big difference.
- Final version (Revision step (3040 / pb with most problems selected) + V0 (1024) + V1 (1024) + V1.1 (1024) + V2 (2048)): 50%
    - To reach 50% with just samples from V2, I would need 100,000 samples! So the revision is doing a huge amount of work. (With just V0, I would need 1,500,000 samples.)
    - I was surprised by how much revision helps.
    - Further rounds of revision might also help, but I didn't get around to running this. (I had this implemented in an earlier version, but I removed it.)
    - Revision seems to fix around 20% of the remaining incorrect solutions on both test and train. As in, if we're getting 37% correct, then we get an additional 0.63 * 0.2 = 0.13 correct for a total of 0.5. Accordingly, revision yields a notably

smaller absolute improvement on train (an absolute improvement of only 6% vs 13%).

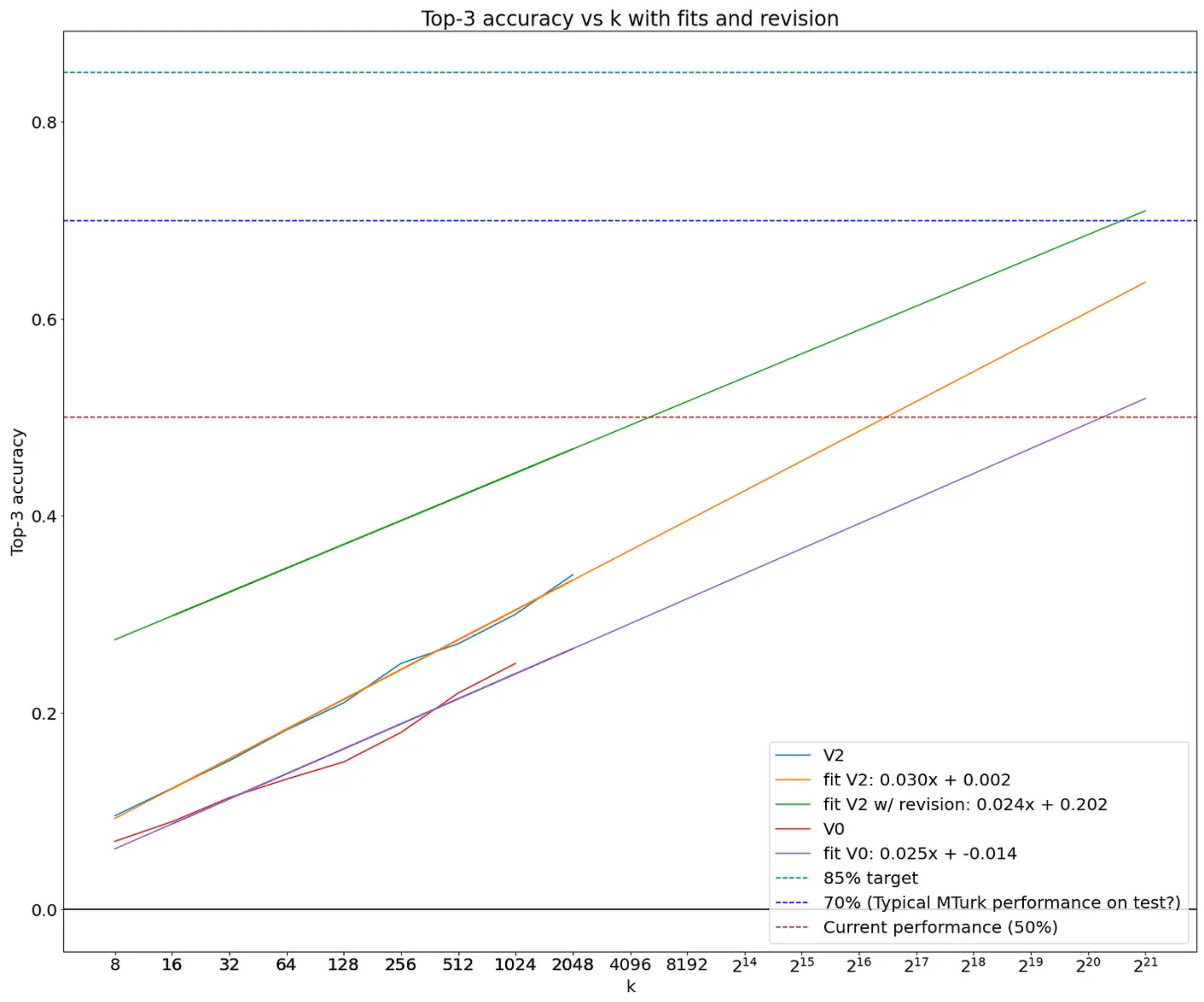
I don't have numbers for cutting reasoning while using comparable amounts of samples.

I use the exact same settings for the train distribution numbers I report.

We can also compare the scaling of different prompts:



Above, I noted that revision seems to fix around 20% of the remaining incorrect. Supposing that this scales indefinitely (a dubious assumption), we can see how long it takes to reach various performance thresholds with my current method and various ablations.



Getting to "just" 70% performance with my current method would take 2^21 samples (around 2 million)! I think 70% is probably similar to the performance of typical people on MTurk on the test set, though no number is known.

## Qualitative analysis

GPT-4o is limited by failures other than reasoning, such as:

1. **GPT-4o's vision is terrible on grids**. When asked to describe what is in a somewhat large grid, it often fails to "see" the input correctly, and states wrong facts about what colors are in some location or what shapes are present.
   a. In particular, it totally fails to extract the colors of cells from an image for images >12x12 and is quite bad at 8x8.
   b. If humans had visual abilities as poor as GPT-4o, it would often take them quite

a bit of effort to solve even simple ARC-AGI problems. (If you want a frustrating time, try solving some ARC-AGI problems without using vision other than reading: that is, try to do them without ever drawing out the grids in 2d, forcing yourself to instead just interact with a textual representation of the data. For hard mode, you could try doing this blindfolded, with a friend allowing you to dictate Python lines of code to run on the image. I think this would be quite hard.)

2. **GPT-4o isn't that good at coding** (especially not for these sort of geometric manipulation problems), and makes simple mistakes like off-by-one errors extremely often.
    a. We don't do multi-round debugging because it's probably cheaper and more effective to just get more samples in the current regime.
3. **GPT-4o is worse at using long contexts** than other models:
    a. I think the long context for GPT-4o is quite bad and starts taking a big hit after about ~32k to 40k tokens (based on my qualitative impression), which limited my ability to use longer prompts with more examples and more detailed representations.
    b. It doesn't seem to respect my few-shot prompt and often does somewhat worse stuff than what it should do based on the few-shot examples. For instance, it systematically responds with much shorter completions than it is supposed to (even if I give it very specific instructions to do otherwise!).
4. **Not having flexible prefix caching substantially limits approaches**. With the n option in the OpenAI api, but without more flexible prefix caching, efficient approaches are limited to just drawing lots of samples rather than efficiently having a large number of reasoning traces which diverge, proceed in parallel, and receive outputs from code (while sharing a few-shot prompt). This massively increases costs for approaches which focus more on doing multi rounds of revision and iteration on all or most implementations rather than just doing a large number of 1-step revisions on one implementation (or a small number of implementations).

Removing these non-reasoning weaknesses would improve the performance of my solution by a significant amount. Vision is an especially large weakness.

To be clear, GPT-4o is also limited by being very dumb. A smart human with only access to the text representation could substantially outperform GPT-4o by spending a bunch of time on each problem.

## Caveats

- I used over 1000x more runtime compute per problem than prior work on this benchmark. Maybe prior work on this benchmark scales well with compute and would have gotten higher accuracy with higher resources.
- There is a broad distribution of problem difficulty even in the test set. For example, some of the problems have a difficult-to-guess component which only has a few sensible options (e.g. pick which of two possible colors to use in some location), and the evaluation metric is top-3 accuracy. Therefore, the gap between our solution's performance and human performance in the test set may be larger than you would expect when just looking at the numbers.
    - In practice, changing to top-2 accuracy only affects performance by a tiny amount. (it goes to 48% on test and 70% on train)
- To decrease cost and to allow for having a subset to iterate on (especially for the train set!), I only ran my solution on a randomly selected set of 100 train and 100 test problems. The resulting standard error of the mean is reasonably tight (5%), but this means my results might be slightly overoptimistic.
- This submission is ineligible for the ARC-AGI prize and main leaderboard as it uses a closed source model and too much runtime compute.
- I haven't yet submitted to the private leaderboard. (I need to port it into a kaggle notebook and make sure it doesn't use too much RAM).

## Predictions

- 70% probability: A team of 3 top research ML engineers with fine-tuning access to GPT-4o (including SFT and RL), $10 million in compute, and 1 year of time could use GPT-4o to surpass typical naive MTurk performance at ARC-AGI on the test set while using less than $100 per problem at runtime (as denominated by GPT-4o API costs).
    - Note that typical naive MTurk performance is probably not 85% on the test set (the prior baseline is on the easier train set) and there is no known human baseline. (Perhaps performance is around 70%?)
- 35% probability: Under the above conditions, 85% on the test set would be achieved.

It's unclear which humans perform at >=85% on the test set, though this is probably not that hard for smart humans.

- 60% probability: If a next generation frontier LLM (e.g. GPT-5) was much better at basic visual understanding (e.g. above 85% accuracy on Vibe-Eval hard), using my exact method (with minor adaptation tweaks as needed) on that LLM would surpass typical naive MTurk performance.
- 30% probability: Under the above conditions, 85% on the test set would be achieved.
- 80% probability: next generation multi-model models (e.g. GPT-5) will be able to substantially advance performance on ARC-AGI.

## What it means about current LLMs

Chollet says:

> If you were right [that LLMs can do in-context learning], LLMs would do really well on ARC puzzles because ARC puzzles are not complex. Each one of them requires very little knowledge. Each one of them is very low on complexity. You don't need to think very hard about it. They're actually extremely obvious for human [sic]
>
> Even children can do them but LLMs cannot. Even LLMs that have 100,000x more knowledge than you do still cannot. The only thing that makes ARC special is that it was designed with this intent to resist memorization. This is the only thing. This is the huge blocker for LLM performance.
>
> If you look at LLMs closely, it's pretty obvious that they're not really synthesizing new programs on the fly to solve the task that they're faced with.

Contra Chollet, I think that current LLMs are well described as doing at least some useful learning when doing in-context learning.

In particular, given my results you have to reject one of the following claims:

1. Getting modest performance on ARC-AGI (e.g. 50%+ on test) requires at least a little runtime "learning" on each problem.
2. Program selection on only a moderate (e.g. 6,000) number of programs (like the way I do it) doesn't count as "learning" in the typical way people think of learning.

3. Current LLMs never "learn" at runtime (e.g. the in-context learning they can do isn't real learning).

Claim 1 seems likely true to me for a reasonable notion of “learning”. I think François Chollet agrees here. Most of my doubts about this claim are concerns that you can basically brute force ARC-AGI without interestingly doing learning (e.g. brute-force search over some sort of DSL or training on a huge array of very similar problems). These concerns apply much less to the kind of approach I used.

Claim 2 seems true to me: the distribution of programs you are searching over has to be pretty close to the right program for Best-of-6k to work at all: if you did best-of-6k for random python programs, this would not work! Perhaps François Chollet disagrees here, but I think this view would be unreasonable.

Therefore, I think Claim 3 is false: **I think LLMs actually do some relevant “learning” when doing in-context learning.** Overall performance is very weak (otherwise I wouldn’t have needed to draw thousands of samples in my solution), but it’s some learning nevertheless. (Though there are various obstacles to GPT-4o performing well other than reasoning and learning ability such as vision and coding limitations.)

(One caveat is that this could be false if a substantial fraction of GPT-4o’s performance comes from dataset contamination. This seems very unlikely to me.)

It’s worth emphasizing that GPT-4o’s learning within a single context seems much less competent than typical human learning. But it is learning nonetheless. My view isn’t that GPT-4o is smart relative to humans in the typical way we mean smart, but I do think it has something which is well described as intelligence.

## What ARC-AGI tells us about AGI

Progress has not stalled. I think ARC-AGI will be one benchmark among many that just gets solved by scale, and that as LLMs are scaled up, we should expect them to be able to solve tasks of increasing complexity when used with the appropriate tools, resources and prompting/scaffolding. (In this case, performance is due to scale and 6 days of iteration.)

I think it is plausible that scaling LLMs [12] by another 2-10 OOMs in effective training compute, and giving them tools, resources and scaffolding to solve real-world tasks can result in "AGI" [13], understood as AI capable of massively accelerating R&D. Such a technology would likely be the most transformative technology in human history, and I prefer to refer to this as "Transformative AI" (TAI) due to the potential ambiguity of "AGI".

TAI poses huge risks. Making mistaken predictions about where LLMs are heading could result in a dramatic underestimate of the dangers they could pose. If, like Mike Knoop (co-host of the ARC-AGI prize), you oppose bills like SB-1047 because you think LLMs won't scale [14], then it really matters that you are right about LLMs not scaling. And every time you get evidence that indicates that scaling might be dangerously powerful (and I hope this post provided some), you should update appropriately in favor of more caution.

ARC-AGI probably isn't a good benchmark for evaluating progress towards TAI: substantial "elicitation" effort could massively improve performance on ARC-AGI in a way that might not transfer to more important and realistic tasks. I am more excited about benchmarks that directly test the ability of AIs to take the role of research scientists and engineers, for example those that METR is developing. (I think developing these evaluations and the science of conducting these evaluations is a highly leveraged way of reducing the risk that powerful AGI takes humanity by surprise; if you're interested in contributing to them, you can see open roles at METR here. Note that I have various COIs with METR.) I still think that work like ARC-AGI can be good on the margin for getting a better understanding of current AI capabilities.

(I'm ambivalent about advancing AI progress overall, especially in a broadly proliferated fashion. If I thought that my work on ARC-AGI would likely substantially advance AI progress, I would not have published this blog post.)

More minimally, I think that ARC-AGI would be a better evaluation of progress towards TAI if it used purely text based problems or at least had a text based subset: good vision isn't necessary for TAI and improved vision has outsized effects on ARC-AGI relative to TAI progress.

I also think that the ARC-AGI prize is made worse by not allowing SOTA (closed source) LLMs in submission and by overly restricting runtime computation [15]. (There is an open leaderboard which has no such constraints.) I expect that SOTA LLMs will be pushing the frontier of progress in ARC-AGI based on these results and general views about what will happen with SOTA LLMs in the next few years. Higher limits on runtime compute seem important for advance warning: if an approach currently costs 10x human labor costs but can do a task, then it will probably cost way less in a few years (or less time) as further optimizations accrue. For instance, substantially optimizing the runtime compute used by my approach seems doable.

Overall, I appreciate the approach of ARC-AGI and I appreciate that Chollet and Knoop have made strong and relatively specific claims. (Some of which now seem to be contradicted!) Nonetheless, I think there are substantially better ways to benchmark progress toward transformative AI.

## Appendix: A bunch of tricks used in my solutions

I use a bunch of tricks to improve performance:

- I split problems into two buckets: problems where the grid is the same size in the input and the output (in the examples) and problems where the size differs on any example input. I use a different prompt for these two cases. I use different examples and I also use a more extensive ascii representation in the case where the grids are the same size, as this representation is more useful in this case. I also use different prompts when having GPT-4o fix attempted implementations.
- I use various ascii representations (in the case where grids are the same size) to make it easier for GPT-4o, which is very bad at this type of vision, to analyze the grid. Few-shot examples use these representations when reasoning. We show:
    - The grid as a 2d array where each location has both the color and that location is spreadsheet/chess style notation (e.g. A7, B3, etc.)
    - The list of all locations (in spreadsheet style notation) at which a given color occurs. We split this up into connected components to make shapes easier to understand.
    - A normalized view of connect components (shapes) which shows the spreadsheet notation with shapes translated to have minimum row and column at 0. This makes it easier to compare shapes without getting confused by offsets.
    - A “diff” between the input and output represented as pairs of distinct colors and the locations at which the input has the first color and the output has the second color (e.g., black changes to green).[^diffonly] This is only used in a

subset of examples (see below about ensembling over few-shot prompts).

    - When revising implementations, I also show an ascii diff between the expected and actual output for each of the examples. This is shown regardless of whether the input/output grid for the examples are the same size. (Though it can only be shown if the expected and actual output have the same shape.)

- I do a bit of local search to optimize prompts (on different subsets of the data from the subsets where I report performance!). Mostly, this looks like deciding which few-shot examples to include and which to put at the start/end of the prompt (which the model probably pays more attention to).
- I use a few-shot prompt for the revision operation rather than just asking the model to revise. In the few shot prompt, I include some cases where revision is both reasonably doable and the model failed to successfully revise (on a prior version of the prompt on the train set).
- I use a somewhat non-trivial hamming distance approach for picking which implementations to revise though I don't think details here matter much (based on my testing). I end up selecting 12 solutions which are ranked in closeness to the output and then using 3040 samples split over these solutions. I allocate the samples by reducing the samples by 25% for each successive item (the first item gets 384 samples, the second gets 288, the third 224 (I round to the closest 32), and so on).
    - I use the geometric mean of the hamming distance ranks to each of the outputs in the examples. (That is, I get the distances for each implementation, I rank them based on these distances, and then I take the geometric mean of these ranks across examples.) I also encourage slightly more diversity by penalizing for being too close to the solutions I've already selected. I only revise implementations that produce valid grids. I also skip implementations that produce a grid with a size that doesn't match the expected output for problems where the size shouldn't change (based on the examples).
    - I somewhat arbitrary upsample completions from better prompts, in particular the V2 prompt described above is upsampled. I just guessed at a constant offset to add for V2 to to remove for V0 based on some eyeballing of the typical geometric mean and what I thought might make sense. I didn't optimize these values.
    - I only do revision on problems with fewer than 32 solutions that are correct on the examples with the idea being that we probably got the other solutions

    correct and thus don't need to run this. I filter out solutions which are correct on the examples (as it's unclear what revision should do in this case).

- I ensemble over several few-shot prompts. I think this somewhat improves performance over more samples with a fixed few-shot prompt (the diversity improvement is larger on the test distribution than the train distribution I think) and I already had a bunch of these samples from prior runs.
- To pick what to submit, I do a majority vote and pick the top-3. First, I do a majority vote over the implementations which are correct on the training set. If this doesn't produce three outputs (because there are less than 3 distinct outputs from implementations which are correct on the training set), I do a majority vote over implementations where I weight implementations based on how close they are to the expected output on the examples. In particular, I use the same geometric mean of hamming distance ranks metric discussed above where the weight for the majority vote is the reciprocal of this geometric mean.
    - I don't think the details of this matter much (and it isn't optimized), I just thought something like this would be a good idea.
    - I also reject outputs which are equal to the test input. There are no cases in ARC-AGI where the test output is equal to the test input. This is more important than you might have thought because implementations which make no changes sometimes do well according to the hamming distance metric but are surely wrong. (If there are multiple test inputs, we reject if any are the same.)
- I use the n option from the GPT-4 API to make more samples cheaper.

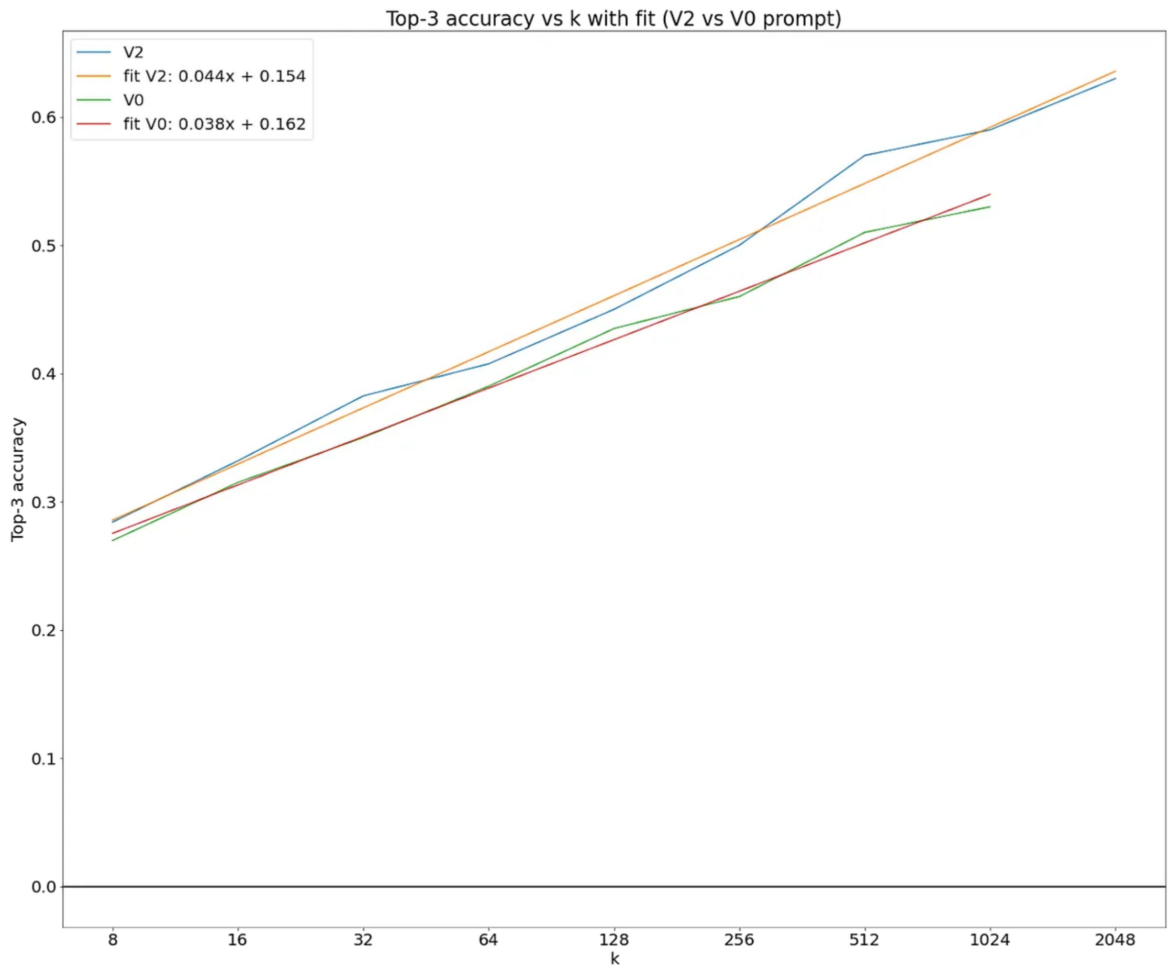
Some tricks I don't use that might improve performance:

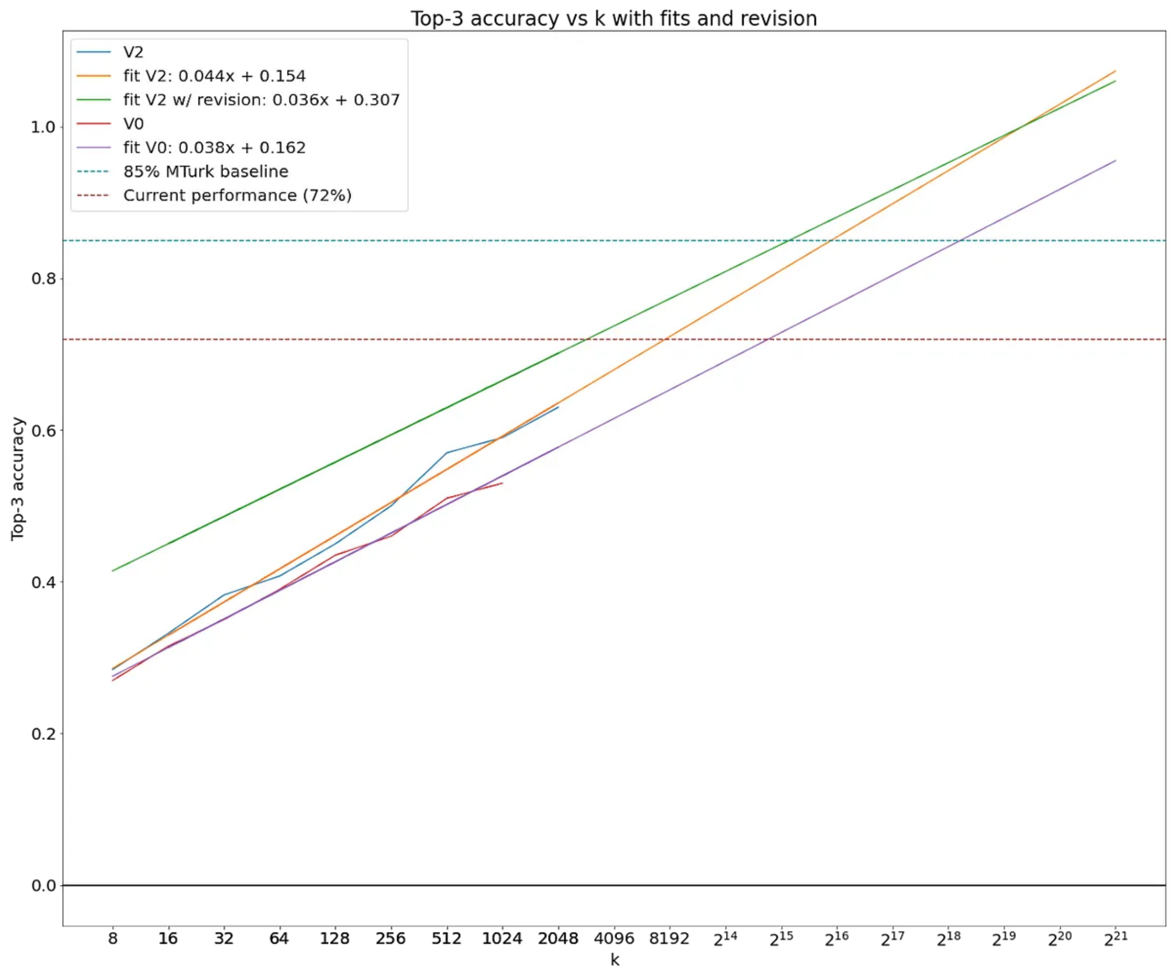
- I currently spend the same number of samples on each problem. You could terminate early on problems which are consistently solved. I think this might reduce cost by perhaps 25-35%.
- In the OpenAI API, I use n < 128 (typically 32 or 16) because n=128 typically errors from what I've seen. Currently it seems like about half of my cost is input tokens, so going to n=128 would roughly halve the cost.
- It would probably help to divide the problems into substantially more categories and then build specialized prompts and tools for each category. This somewhat

defeats the point of ARC-AGI though and I'm not sure what these categories would be.

- Doing a second or third revision round could help substantially. (Relative to spending these samples elsewhere.)
- Further extending the debugging/revision process could help substantially.
- Fine-tuning of GPT-4o to better understand the representations I use (and be able to see) would surely help a bunch (though it would be expensive).

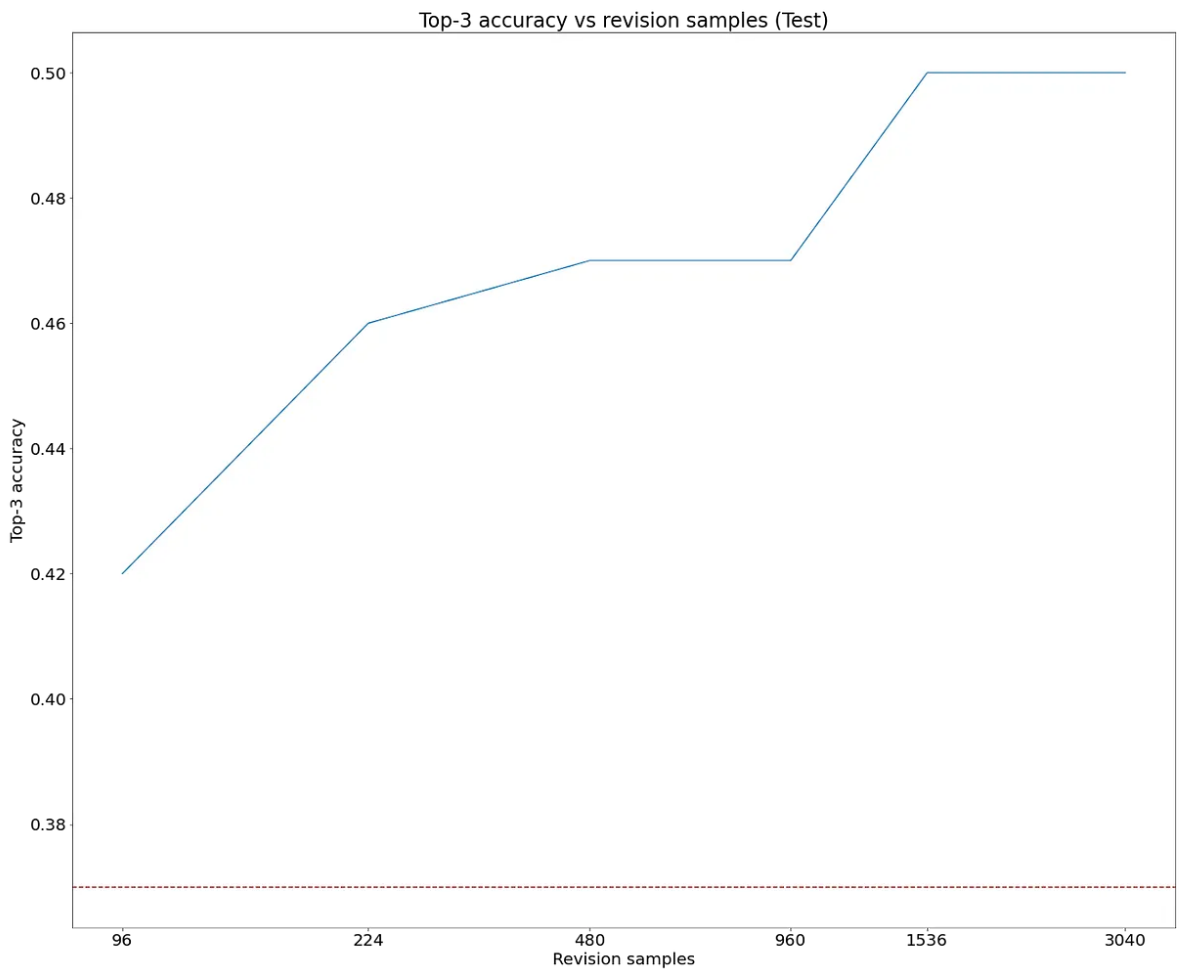
## Appendix: results for the train set

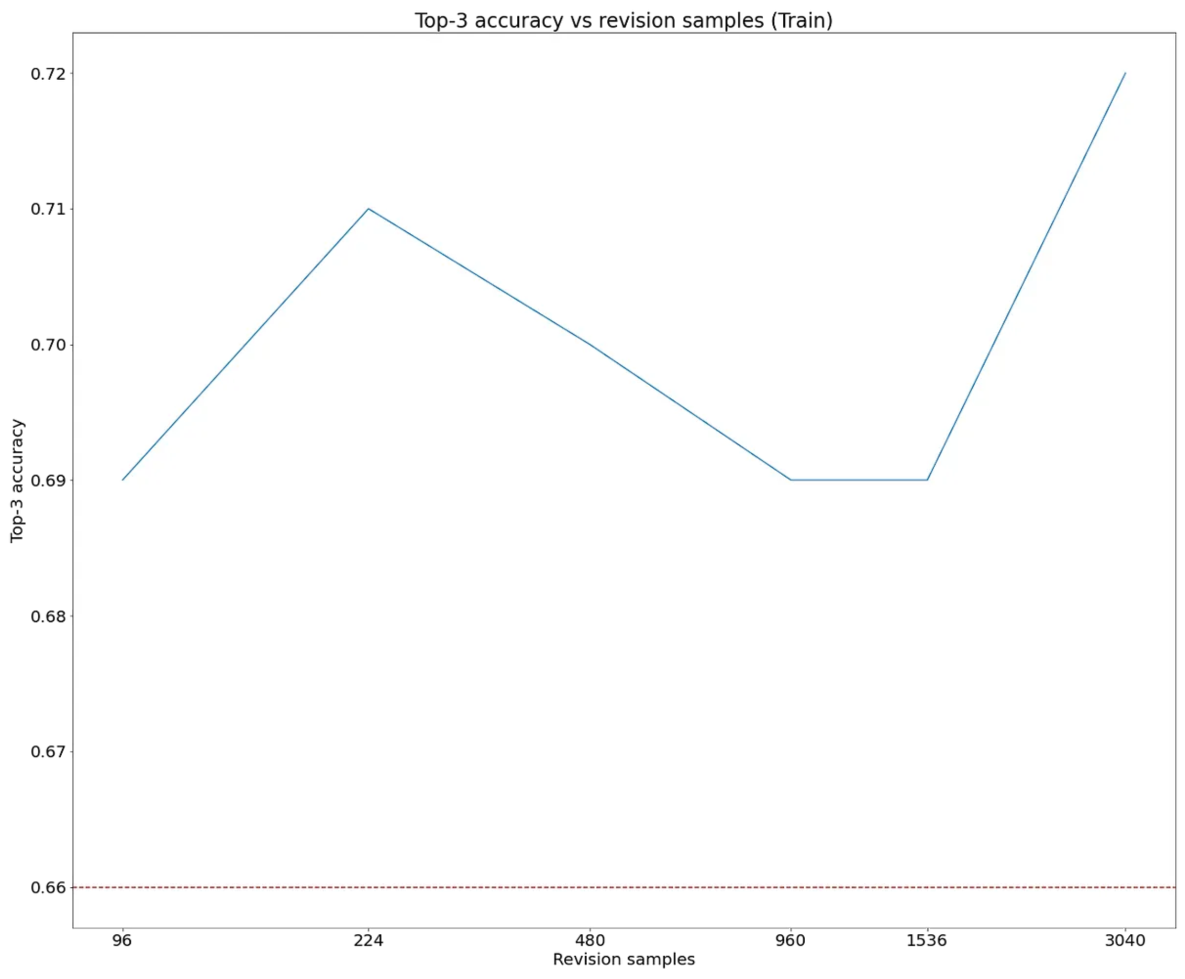




Around 2^15 or 32,000 samples would be required to reach MTurk performance on the train set.

## Appendix: Returns to revision samples


Top-3 accuracy vs revision samples (Test)
0.50
0.48
0.46
0.44
0.42
0.40
0.38
Top-3 accuracy
96
224
480
960
1536
3040
Revision samples



1 The number being exactly 50% is a coincidence. An earlier version of this method got 51%, but then I fixed a bug and reran and ended up at 50% by chance.

2 I haven't yet submitted to the public leaderboard, but I'll do this at some point. (I haven't done this yet because this requires writing a kaggle notebook and might require me to reduce ram usage somewhat. (My ram usage is highly unoptimized.))

3 Prior results didn't have access to SOTA LLMs nor did they use nearly as much compute as I did. Also, prior results are evaluated on a private test set, though this shouldn't matter much since it's supposed to be basically IID with the public test set.

4 I hold out 100 problems from the train set which I avoid doing any iteration or prompt optimization on. I then test on this held out set and report these numbers.

5 These problems were picked from the public evaluation set, and I chose the third random sample of three problems, as the first sample didn't seem representative and the second sample didn't have the AI getting one right for a prior version of my method (it now gets one right on this second sample). These are problems named '642d658d.json', '0934a4d8.json', 'fafd9572.json' on ARC-AGI's GitHub repository. You can find more here by selecting "Public evaluation set" as test set instead of the default "tutorial". You can submit up to 3 answers per problem (accuracy is top-3 accuracy); my solution got the first right on its first try. Interestingly, this third problem was actually done correctly by my method (on the first try) on a prior run of the approach, but after I fixed a bug and reran the revision step, it now is incorrect.

6 In practice, GPT-4o seems to not really bother doing a good job of following my reasoning examples, so being this meticulous in the reasoning is probably not that useful (though it probably helps some). I tried it because I've previously found that extremely detailed few-shot prompts have been really helpful for eliciting high-quality reasoning from LMs.

7 ARC-AGI allows for 3 submissions per problem (based on the description on the github) and I assume prior SOTA and the human baseline is reported with 3 submissions per problem. It seems like the contest maybe now only allows for 2 submissions. When lowering to 2 submissions, I get 70% (rather than 72%) on the train set and 48% (rather than 50%) on the test set.

8 Early in this project I didn't realize these sets differed so much and thought I had made much more progress! This is clearly pointed out in the current technical guide and was noted somewhere on the old website, so this is mostly on me.

9 It's non-obvious how to budget samples when using multiple few-shot prompts for diversity. In practice, I find that more samples from V2 basically dominates diversity from the prompt ensemble I use.

10 For the fit, I cut off sample counts lower than 8 as I found that this sometimes makes the fit considerably worse in the regime we care about. There are good reasons to expect different scaling for small numbers due to using top-3 accuracy.

11 We could resolve this by changing the y axis to log(1-accuracy) which should have the intended asymptotic properties. I've looked at this and this fit seems slightly worse in this regime. Probably both fits start to fail after less than 10 more doublings.

12 By LLMs, I really mean AIs which heavily utilize semi-supervised pretraining and are basically a continuation of the current paradigm of SOTA LLMs.

13 I don't like the term AGI. I prefer using the term transformative AI and ideally defining this term to mean something specific like "AI capable of massively accelerating R&D".

14 I don’t think that this is actually a good argument against SB 1047, since this bill only requires that companies that spend at least $100M on a training run measure dangerous capabilities (which is very cheap compared to the training run itself), and apply appropriate safety measures *if* the evaluation shows that the resulting AI can cause a catastrophe. I discuss my thoughts on SB-1047 in more detail in this blog post.

15 That said, I understand why heavily restricting runtime compute might be important for kaggle.

---

84 Likes · 9 Restacks

Previous

## 31 Comments

Write a comment...

**Nina Rimsky** Nina’s Substack Jun 17 · *edited Jun 17* Liked by Ryan Greenblatt

> Contra Chollet, I think that current LLMs are well described as doing at least some useful learning when doing in-context learning.

I agree that Chollet appears to imply that in-context learning doesn't count as learning when he states:

> "Most of the time when you're using an LLM, it's just doing static inference. The model is frozen. You're just prompting it and getting an answer. The model is not actually learning anything on the fly. Its state is not adapting to the task at hand."

(This seems misguided as we have evidence of models tracking and updating state in activation space)

However later on in the Dwarkesh interview, he says:

> "Discrete program search is very deep recombination with a very small set of primitive

programs. The LLM approach is the same but on the complete opposite end of that spectrum. You scale up the memorization by a massive factor and you're doing very shallow search. They are the same thing, just different ends of the spectrum."

My steelman of Chollet's position is that he thinks the _depth_ of search you can perform via ICL in current LLMs is too shallow, which means they rely much more on learned mechanisms that require comparatively less runtime search/computation but inherently limit generalization.

I think the directional claim "you can easily overestimate LLMs' generalization abilities by observing their performance on common tasks" is correct - LLMs are able to learn very many shallow heuristics and memorize much more information than humans, which allows them to get away with doing less in-context learning. However, it is also true that this may not limit their ability to automate many tasks, especially with the correct scaffolding, or stop them from being dangerous in various ways.

LIKE (4) REPLY SHARE

1 reply

**osmarks** Jun 17 Liked by Buck Shlegeris

Is it possible that the issues with vision are because the size of the cells in the input is different from GPT-4o's patch size?

LIKE (3) REPLY SHARE

1 reply

**29 more comments...**