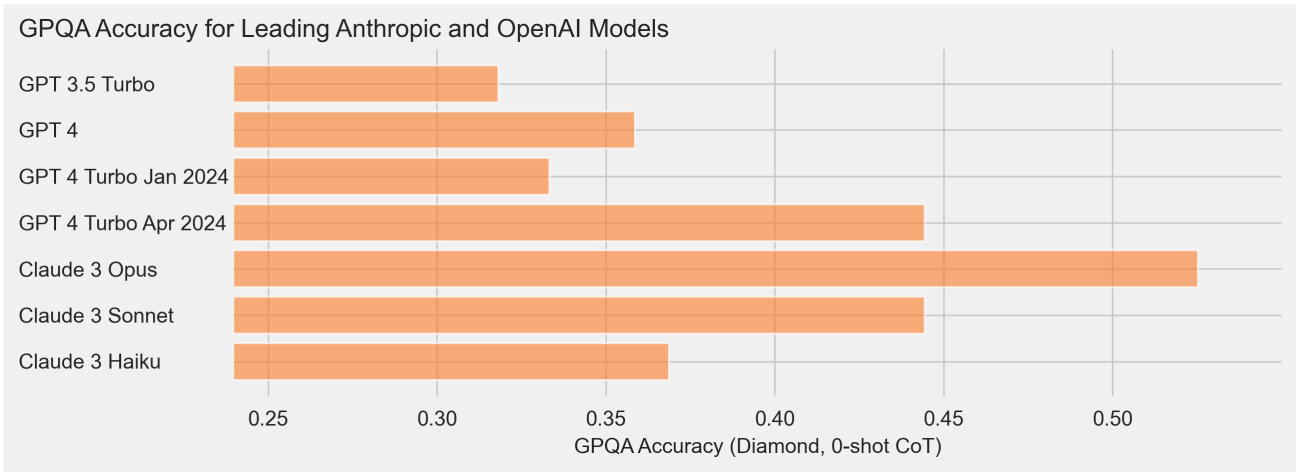
You need to be spending more money on evals.

Alternative title: I paid $440 for error bars and you should too.
Alternative alternative title: You're reading too much into individual evaluation results.

Imagine I showed you this plot; what conclusions would you draw?



Perhaps "Claude 3 Opus is a better model than GPT-4 Turbo?" Is that not what the results clearly show?

It's so so tempting to draw conclusions like this from a beautiful high-res bar plot that tells a consistent story, but actually, no, the results don't show that Opus is a better model than GPT-4 Turbo. You're just falling for small number statistics.

First, a side point about GPQA.

GPQA is a challenging dataset of 4-choice multiple choice questions that are difficult to answer even with unrestricted access to the internet. The questions are written by expert scientists in biology, physics, and chemistry. "GPQA Diamond" is a 198 question subset of the most high quality questions.

GPQA is an amazing dataset[1] that fulfils a very specific need for AI alignment research: evaluating scalable oversight techniques. These methods supervise AIs that are more capable in some way than the supervisor, and we want to know how well the techniques themselves work. For this, we need a testbed where most humans don't actually know the answer. GPQA is perfect for this because non-expert humans answer the questions only 34% of the time, even when trying really hard.

What GPQA is *not* is a general intelligence evaluation. GPQA covers not only a limited set of knowledge (three scientific fields), but it also only covers a specific *type* of scientific knowledge - that is, niche knowledge that is difficult to search online. There are other types of scientific intelligence not covered in this evaluation, such as creativity, lab skills, communication, and research taste.

The point is that a model that performs better on GPQA knows how to answer hard physics, bio, and chem questions better; but we should be careful when generalising from this to all types of model intelligence. That's just a side point, though. My main point is that everyone seems to have forgotten about confidence intervals.

Second, small numbers.

There are 198 questions on GPQA Diamond. The baseline evaluation method used for this task is `0`-shot chain of thought, meaning that you'll prompt your model with something like this...

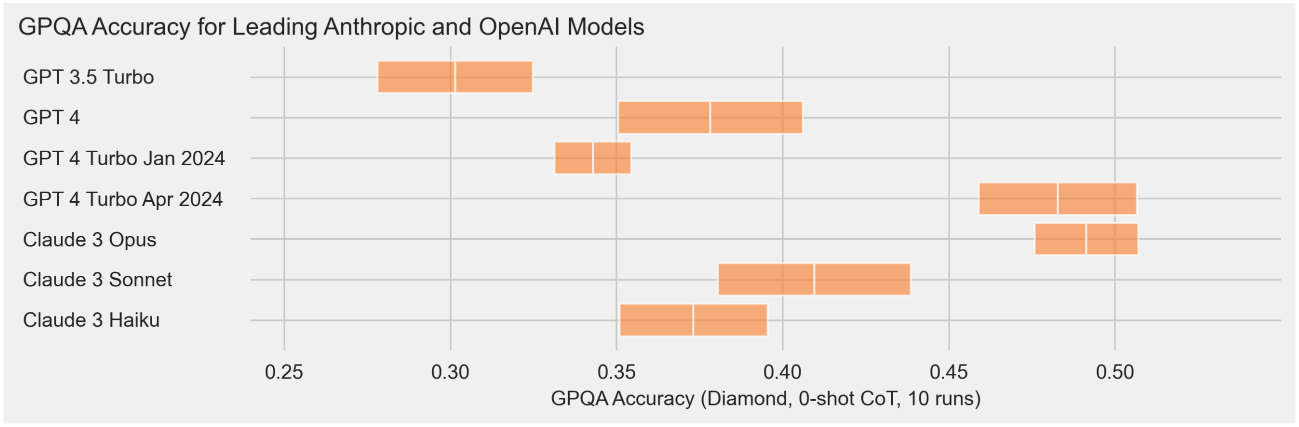
```
Question: For the hydrogen atom, which series describes electron
transitions to the N=1 orbit, the lowest energy electron orbit?
(A) Lyman series
(B) Balmer series
(C) Paschen series
(D) Pfund series

Please first provide your reasoning, then answer with a single letter
choice formatted as "The answer is X."
```

...sample some tokens, and use some method (string matching or another model call) to extract the model's final answer.

Because there's a sampling step involved for the chain of thought reasoning, you'll get a slightly different reasoning and potentially a different answer choice every time you ask the model. Although you could sample the highest probability token deterministically (T = `0`) or force the model to simply choose one letter without providing reasoning to eliminate the noise, I'd argue that this is a worse way to do an eval because it is not representative of the model at its most capable (which includes "space for thinking") nor is it representative of the way users engage with a model. There are some smaller sources of variability too, such as which letter choice the correct answer is assigned, and how the multiple choice question is formatted (e.g. (A), (B), (C) vs A., B., C.). It's fine to accept the non-deterministic system of `model + prompt + CoT + temperature` as the system of interest, but you'll have to account for that variability.

To get reliable measurements of performance, you'll need either a sufficiently large dataset, or you'll need to run the evaluation several times to get an accurate estimate of performance. If you don't, you might be fooled into thinking you've detected some difference in model performance which would in fact disappear with additional measurements. This is the curse of small number statistics.

GPQA (Diamond) is a *particularly noisy dataset* because it contains - relatively speaking - very few questions.[2] Consider the performance of the same models but reporting the mean estimate and 95% confidence interval over 10 runs.[3].



From this additional data, it's no longer "obvious" that Claude 3 Opus performs better on GPQA than GPT 4 Turbo. The only thing that's obvious is that the results are noisy and therefore you need to spend more money on evals to get better measurements.

Unfortunately, statistics.

I'm really interested in answering the question of how many examples in your dataset you need in order to meaningfully compare the performance of two models on a given benchmark. However, there are a few things that need to happen before I can even begin to do that.

First, we need to decide how we're thinking about a benchmark. Do we care about performance on the specific set of questions in a benchmark, or do we care about performance on a benchmark as an estimate of performance on the distribution that the questions were drawn from? The answer depends on your use case, but we would have to pick one perspective to work with because this will determine *how* we compare model performance.

Right now, the default is to use the highly sophisticated method of "number A is bigger than number B" to compare models. Ideally, we want to actually account for all the sources of variability that lead to a result on a benchmark, such as varied difficulties of each question and the distributions of model answers for each question, before comparing model results.

It seems unreasonable to recommend a complex method given the current status quo of model comparison. So right now, I'm going to propose the simplest, most widely applicable statistical techniques for comparing the performance of two models that account for some of the variability in the data but certainly not all. In the future, I hope to get back to the question of what the "best" thing to do is and to estimate sample sizes.

Assumptions

We have a benchmark which contains n multiple choice questions, and for the purpose of this blog post, we assume that we want to view the test as given and not think about the questions as being drawn from some larger distribution.

A model has some underlying probability p_i to answer question i correctly each time we ask it; asking the model question i once gives you an estimate of p_i (if correct - estimate is $\hat{p}_i = 1$ else $\hat{p}_i = 0$). You can get a better estimate of p_i by running the eval more than once and computing the average performance ($\hat{p}_i = \frac{1}{m}\sum_j x_j$ where $x_j \in 0, 1$ for True and False). From the answers, we compute an expected accuracy on the benchmark with

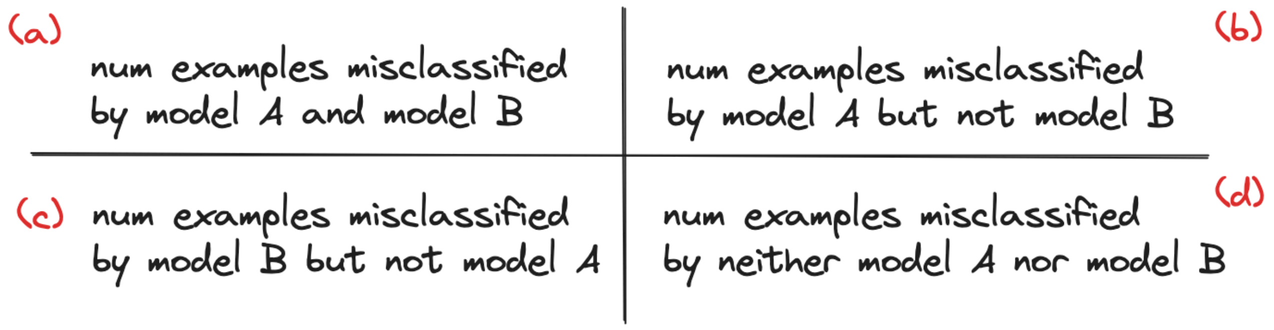
$$a_B = \frac{1}{n}\sum_{i=1}^{n} p_i.$$

Let's assume that we are interested in detecting differences in two estimates of a_B with a probability of Type 1 error (finding that the performance is different when it's actually not) less than 0.05. From here, we split into two possible practical scenarios: are you running your eval more than once or not?

Measuring once: the McNemar test

If you're computing the accuracy on your benchmark once, and you do this for two models, you have one estimate of $\hat{a}_B$ per model. The absolute simplest thing to do is to model a_B with a Binomial distribution to estimate a variance and then use a two-proportion z-test to compare the two estimates. That test would not take into account the paired nature of our benchmark measurements, so instead, we should do a little better and use a McNemar test.

This test is used to determine whether there are differences on a binary variable between two groups where you have paired samples. In our case, we have a binary variable (incorrect vs correct), and we have paired samples because the two models were tested on the same questions. We begin by building a contingency table like so:



(a) num examples misclassified by model A and model B	(b) num examples misclassified by model A but not model B
(c) num examples misclassified by model B but not model A	(d) num examples misclassified by neither model A nor model B

The null hypothesis is that the two models have the same error rate, i.e. $b = c$, with the expected count under the null for b and c being $(b+c)/2$. The McNemar test statistic is

$$\chi^2 = \frac{(|b-c|-1)^2}{b+c}$$

which you can derive from the χ^2 statistic of $\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$ where O_i is observed counts (for McNemar, b and c) and E_i is the expected counts (both $(b+c)/2$), plus a correction term (-1 in the numerator). This statistic is approximately distributed as a χ^2 distribution with 1 degree of freedom, so if the null hypothesis is correct, then the probability that the test statistic value is greater than $\chi^2_{1,0.95} = 3.841$ is less than 0.05, and we may reject the null hypothesis that the error rates are the same.

Let's use the example of one of my results on GPQA for Claude 3 Opus (104/198 correct) and GPT 4T Apr 2024 (85/198 correct). For these particular results, $a, b, c, d = 73, 21, 40, 64$, and we can compute the McNemar χ^2 to be 5.311. Since this statistic is larger than 3.841, we can *reject* the null hypothesis and say that the error rates are in fact different!

But alas, I cheated here, because I picked the best Opus result to compare against the worst GPT 4T result. The McNemar test could be misleading when applied on small benchmarks such as GPQA, so it should really only be used when you have thousands or more examples, or your model exhibits very little variability on the benchmark results. To get more accurate results, we need to be measuring models' accuracy on a given benchmark several times.

Measuring more than once: the (Welch's) t-test

If your benchmark is small, or if it displays an oddly large amount of variability, you must evaluate your model several times to get better estimates of the true accuracy. [4] Then, again the simplest thing to do would be to use a z-test with now actual measured standard errors. However, because we have small means, the actual analogous test is a t-test. More specifically, the correct test is a Welch's t-test because we will not be assuming that our two models display the same amount of variance. The null hypothesis in this test is that the difference of the true means is 0, and since we give the same n questions to both models, the test statistic is

$$t = \frac{\bar{\hat{a}}_1 - \bar{\hat{a}}_2}{\sqrt{\frac{1}{n}(s_1^2 + s_2^2)}}.$$

In this equation, the $\bar{\hat{a}}_i$ are the means of the sample accuracies and s_i are the sample standard deviations per model. If you squint at it, the test statistic comparing the difference in the estimated means to the estimated standard deviations, all mediated by the number of samples in your dataset.

What t value will we need to reject the null hypothesis? It will depend on the "degrees of freedom", which is given by an involved equation that simplifies to

$$\text{df} = \frac{(n-1)(s_1^2 + s_2^2)^2}{s_1^4 + s_2^4}$$

when the number of samples for each model is the same. Then, you round down to the nearest integer and look at the value of a t distribution with that many degrees of freedom to determine the result of your test.

It's a sufficiently involved test now that you should just use `scipy.stats` to compute these values. Let's again compare Claude 3 Opus and GPT 4T Apr 2024, but this time using the results from all 10 of my runs:

```
from scipy.stats import ttest_ind

opus = data["claude-3-opus-20240229"]["totals"]
gpt4new = data["gpt-4-turbo-2024-04-09"]["totals"]

ttest_ind(opus, gpt4new, equal_var=False)
```

```
# -> TtestResult(statistic=0.6824307494848405,
pvalue=0.5050138592619167, df=15.538295366181075)
```

By setting `equal_var=False`, this function will compute the result of Welch's t-test for us, which is what we want. Since the p-value is greater than 0.05, we fail to reject the null hypothesis after all. From here, the next thing to do would be to look into computing a Welch's t-interval to estimate the range of mean differences. But that's some 400-level stats and I'm going to stick to 100-level for now.

Finally, some recommendations.

So in summary, please remember confidence intervals. Bar plots and little numbers without errors reported can be really misleading, and you shouldn't trust them. Always ask yourself how many questions were on the eval and why there are no confidence intervals.

Although comparing results fairly is a difficult question, it shouldn't stop you from doing the absolute bare minimum. If you want to compare the results of two models when you have estimates of accuracy and just one eval run, use a two-proportion z-test. If you're unhappy with the outcome because your eval is small, run the same eval multiple times and use a t-test to compare the results. If you're unhappy then, accept that you've done everything you can and that your models are probably equally as good on the benchmark.

Big thank you to Michael Sklar for serving as my expert statistician. Thank you to Simon Böhm, Rahul Arora, and David Rein for review. Thank you to Liv Gorton for bankrolling this operation lol.

Code for the evaluation, plots, and result files are on github.

If you're interested in working on this topic and have a statistics background, email me!! I would love to keep working on this :)

---

1. I'm a little biased because my good friend David Rein is the lead author of GPQA. ↩
2. The test set of one of the most popular LLM model evaluations, MMLU, has 14,079 questions, in comparison. ↩
3. 10 is, of course, arbitrary. It's just how much I could afford to run. ↩

4. You can also do this for a large benchmark if you have too much time and money on your hands, in which case, I envy you. ↩