You Don't Need to Run Every Eval

I used Claude Code to build a benchmark prediction system, Codex to audit it for bugs, and Claude Sonnet to try to beat it for $1. Here's what I found: LLM evals are so low-rank (in fact rank 2-3) that most evals are approximately... redundant.

Let me give you an example for why one should expect this.

Here's a table. Three frontier models and three benchmarks.

	GPQA Diamond	AIME 2025	SWE-bench
Opus 4.6	91	100	81
GPT-5.2	???	100	80
Gemini 3 Pro	92	95	76

What would you guess the missing entry is? Maybe around 92-94? Sounds about right, it's 93.

You didn't need to evaluate GPT-5.2 on GPQA-D which would have a nontrivial $ cost. What made you guess reasonably well? It was the fact that the rows are nearly identical. In fact these models, on evals at least, are nearly identical.

The point that I want to stress is that there's a ton of similarity that shows up everywhere, across models from the same generation and similar sizes, evals that test similar skills that correlate almost perfectly (e.g., AIME vs HMMT), and a benchmark that's hard for GPT-5.2 is hard for Claude Opus 4.6 (e.g., Terminal-Bench 2.0). There's redundancy in basically every direction you can think of measuring things about these models and benchmarks.

So, can you exploit this structure to predict evals?

Every atom in my brain says yes, and I wanted to check.

So here goes, a few million Claude Code and Codex tokens later...

But a short history lesson first.

An old trick to obtain a million dollar matrix

When I was a grad student, I (and every other EE theory kid 15 years ago) was really into compressed sensing and matrix completion (hi Ben!). The whole point of that theory is: observe a few entries of a matrix, and if you're lucky and the ground truth matrix is approximately low rank, you can recover the rest from surprisingly few measurements by doing something that looks like a singular value decomposition.

There was even a famous $1M Netflix Prize for this: in 2006, Netflix released a matrix of about 100 million (user, rating) pairs and offered $1M (so like 4 weeks of 1024 H100s on Lambda) to whoever could predict the missing ratings 10% better than Netflix's own algorithm, or something like this. The competition ran for a few

years, drew thousands of teams, and even though a team did get the $1M, Netflix never actually deployed the winning solution, lol. But if I'm not wrong, the competition basically invented modern recommender systems and launched a few thousand ML careers.

Here's how it would work in our setting. Suppose each model is secretly described by a vector of two entries -- say, how good it is at general tasks and how good it is at hard novel reasoning. And each benchmark is described the same way, e.g. how much it tests general knowledge vs. how much it tests reasoning. The score is their dot product.

Say GPT-4o has s = (8, 2) -- strong on general tasks, weak on reasoning, and AIME 2025 has b = (1, 8), that is it's mostly testing reasoning. Then GPT-4o on AIME 2025 is 8x1 + 2x8 = 24. (Actually, I think that's close to its true score; didn't even plan this :D.) Now if MMLU has b = (9, 1) -- mostly general knowledge -- then GPT-4o on MMLU is 8x9 + 2x1 = 74.

The idea is that every (model, eval) entry can be produced by the inner product of two 2-dimensional vectors. We'll see later that these two dimensions actually tend to explain the data pretty well!

So the game is: observe a few real (model, eval) scores, solve for the hidden vectors, predict the rest.

But, is the LLM benchmark matrix actually low-rank though?

Easy to check: find a large fully-observed submatrix of the collected data and just run SVD on it. The biggest complete block I found is 31 models x 10 benchmarks, and here's how the singular values look:

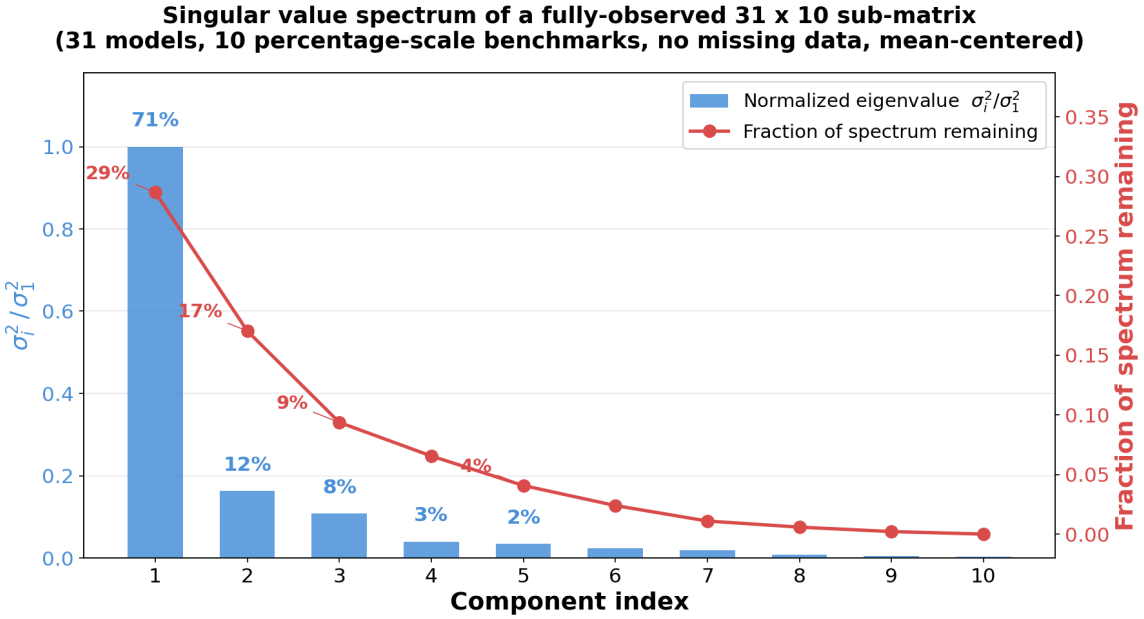


Figure: Singular value spectrum of the largest fully-observed sub-matrix (31 models x 10 benchmarks), mean-centered. Blue bars: each squared singular value normalized by the largest. Red curve: fraction of total spectrum not yet captured by the top i components. The first component captures 71% of the spectrum; two components capture 83%. By component 5, only 4% remains. This steep decay — from raw data with no imputation — confirms the approximate low-rank structure of LLM benchmark scores.

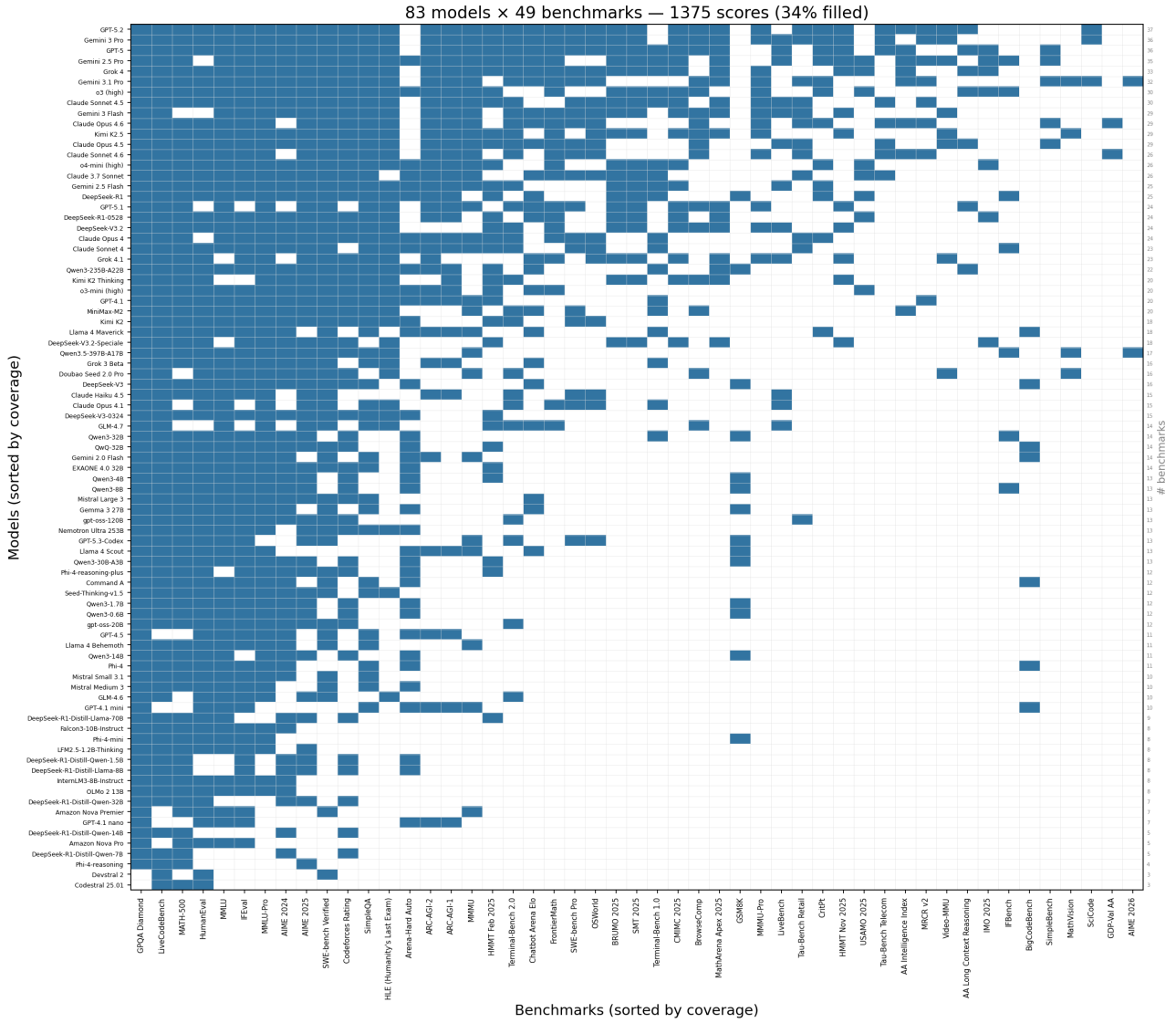
The first component alone captures 71% of the spectrum. The first three capture >90%. So yeah, as with every single matrix I've seen in my whole life, this one is kinda rank-3 too.

I was still skeptical that matrix completion would work, as it typically operates on much larger matrices. But what's the cost of checking with Claude Code and Codex?

83 x 49

I asked Claude to search and verify all possible (model, eval) pairs it could find between January 2025 and now. It found 83 models across 49 benchmarks, every entry cited with a source URL. I asked Codex to audit the findings for hallucinations and to double-check every entry is valid by independently verifying them.

The final matrix is 34% filled, meaning roughly 1,400 out of ~4,000 cells have a ground truth number. The rest are missing because Claude couldn't find a reported score. This is what it looks like:



The 83 models span 21 labs (the usual suspects), and 49 benchmarks (also the usual suspects).

So: how many benchmark scores do you need before you can predict the rest?

Cute question. Let's find out!

Does matrix completion work on 83x49?

It kind of does, for good values of "kind of"! The best method (details in the next section) predicts held-out scores with 7.2% median absolute percentage error. To make that concrete, some examples:

- Gemini 3 Pro on Terminal-Bench 2.0: actual 56.9, predicted 56.2
- Claude Opus 4.6 on AIME 2025: actual 100, predicted 98.1
- GPT-5.2 on SWE-bench Verified: actual 80.0, predicted 80.6
- Qwen3-14B on AIME 2025: actual 72, predicted 73.6

Agentic, math olympiad, coding, small open-weight model -- all within a couple points. But we got some ugly ones too:

- Claude Sonnet 4.6 on ARC-AGI-2: actual 60.4, predicted 15.3

Off by 45 points. We'll come back to why.

A $0 93%-approximation to your eval suite

Claude Code found that everything should happen in logit space: the logit transform, logit(p) = log(p/(1-p)), stretches scores near 0% and 100% and compresses the middle, so the gap between 88% and 92% gets the weight it deserves. This fix improved final accuracy by ~11%. I'm sure it's a standard trick in old school matrix completion.

The final prediction model has two dead simple ingredients.

Ingredient 1: sparse regression.

For each missing cell, find the 5 benchmarks that best predict the target. Concretely: to predict AIME 2025, take another benchmark like MMLU-Pro, plot every model that has scores on both as a point (x = its MMLU-Pro score, y = its AIME 2025 score), and fit a straight line through those points in logit space. The better the line fits, the more MMLU-Pro tells you about AIME 2025. Do this for all 48 other benchmarks, keep the 5 where the line fits best, and average their predictions weighted by fit quality.

Ingredient 2: rank-2 SVD.

Imagine every model is a point in 49-dimensional benchmark space. Rank-2 SVD says: find the best 2D plane through these points such that the projections onto that plane are as close to the real scores as possible. Once you have the plane, every model gets a position on it (2 coordinates) and every benchmark gets a direction on it (2 coordinates) and their dot product gives the predicted score.

There's a small problem: 66% of the coordinates are missing for this to work out of the box. A standard algorithm in matrix completion says to start with zeros, or in our case with column averages, then do SVD, project things onto the singular vectors, replace only the missing entries with the projections while keeping observed scores pinned, and repeat until the projections stop changing.

A convex combination of the two ingredients: 60% of the entry comes from regression + 40% from the SVD.

Why? Because Claude Code did a sweep of combinations and that was the best!

No neural networks or metadata used. We (by we I mean Claude) tried KNN instead of SVD, but KNN and the regression are both local methods that make correlated mistakes. Swapping in SVD gave a 14% improvement.

One thing Claude tried that I asked for and also expected to work: feeding the algorithm model metadata -- e.g., parameter count, provider, reasoning mode, open-weight status. It made things worse. Everything in "14B reasoning model distilled from DeepSeek" is already captured by the model's pattern across scores. Cool!

Also, Claude tried NMF, PMF, nuclear norm, and ALS, and none was better than good old SVD. You love to see it!

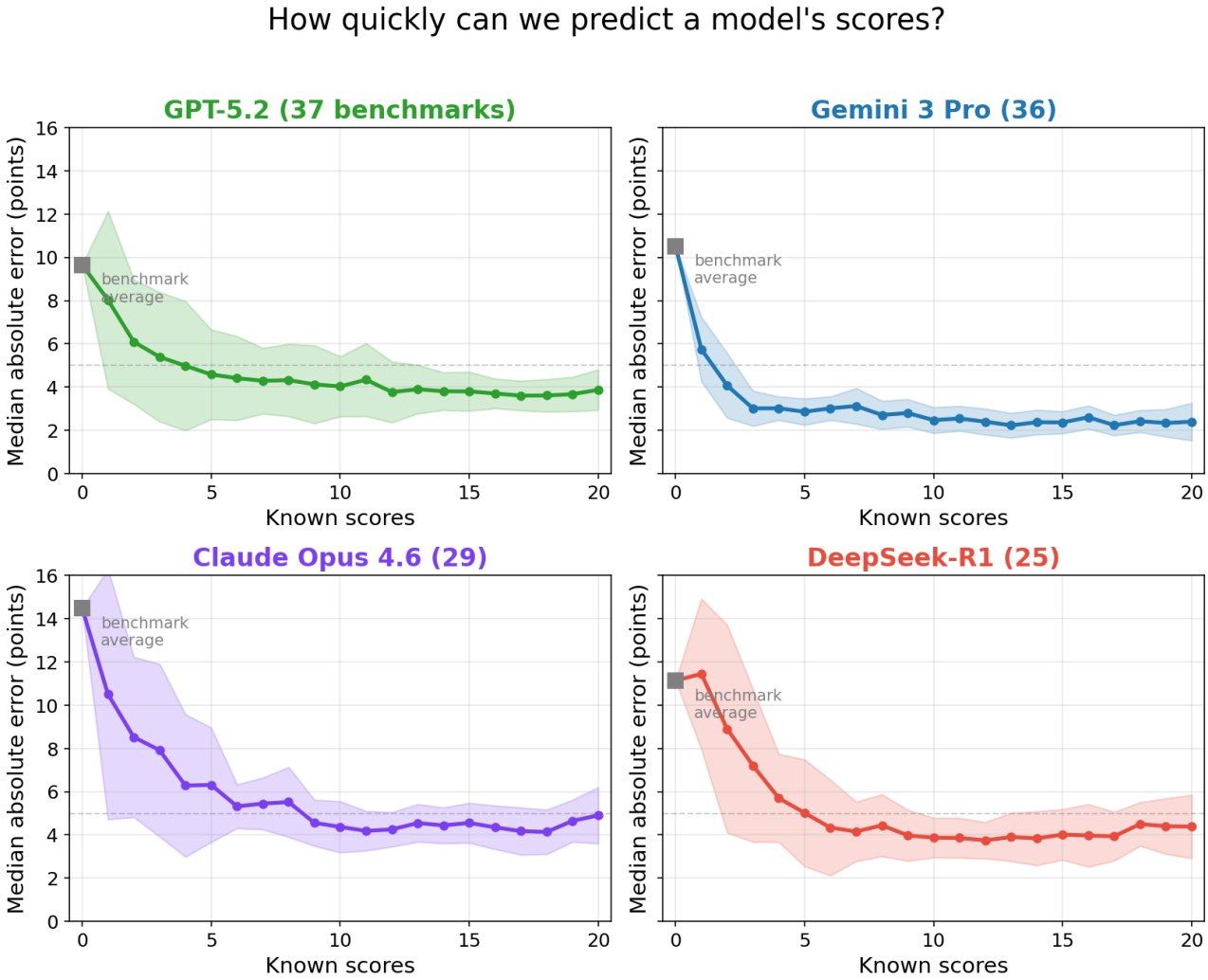
How many benchmarks do you need? Five.

Here's an experiment: take a model, hide all its scores, then reveal them one at a time in random order. After each reveal, build the above predictor and use it to predict the rest. How quickly does the error come down?

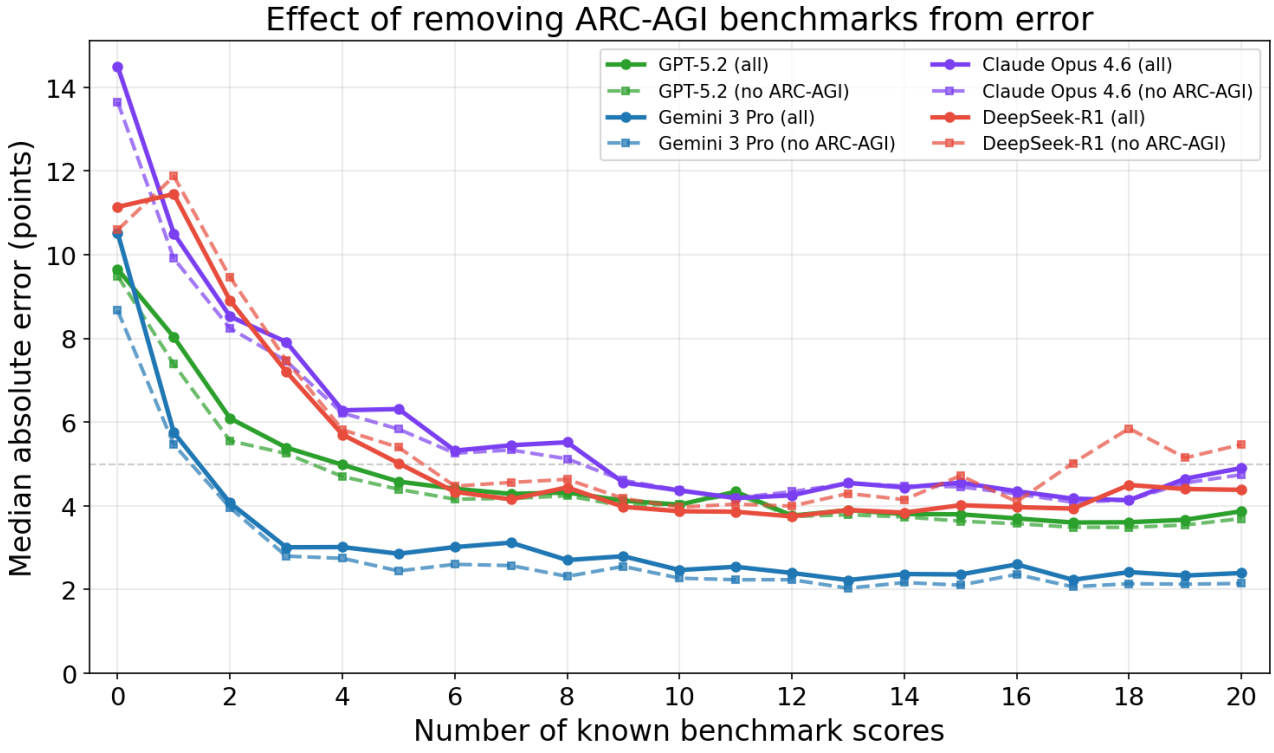
The answer is: fast. With just 1 known score, median error (across all models!) is about 12%. By 5 known scores it drops to 9%. After that at some point it's diminishing returns as the precision of individual predictions keeps improving but the median stabilizes because a few hard-to-predict benchmarks stay noisy.

The practical takeaway is that 5 benchmarks are enough to get a good prediction of how the model performs on everything else.

Here are some example prediction error vs number of revealed tests for a few popular models from the past year.



In fact, if you remove ARC-AGIs the error improves :D I'd say that's a good sign for SVD, bad sign for ARC-AGI.



If you can only pick 5 to eval on, greedy forward selection gives you: {HLE, AIME 2025, LiveCodeBench, SWE-bench Verified, SimpleQA}. They span four categories and cover both principal components. Under proper holdout they get about 7.8% error, versus 7.2% for the full method. Not magic, but a practical minimum eval set.

This seems legitimately cool to me!!

Where does our cheap predictor do the worst?

The hardest benchmarks share two traits: either bimodal score distributions (ARC-AGI, IMO, USAMO) or weak correlation with other benchmarks (Terminal-Bench, Arena-Hard). The method relies on benchmark-to-benchmark structure and when it's not there, the "feature" doesn't help prediction.

So benchmarks resisting prediction: SimpleBench, ARC-AGI-2 and Terminal-Bench with high errors for predicting them. These are the interesting ones, because they measure something the rest of the matrix doesn't capture.

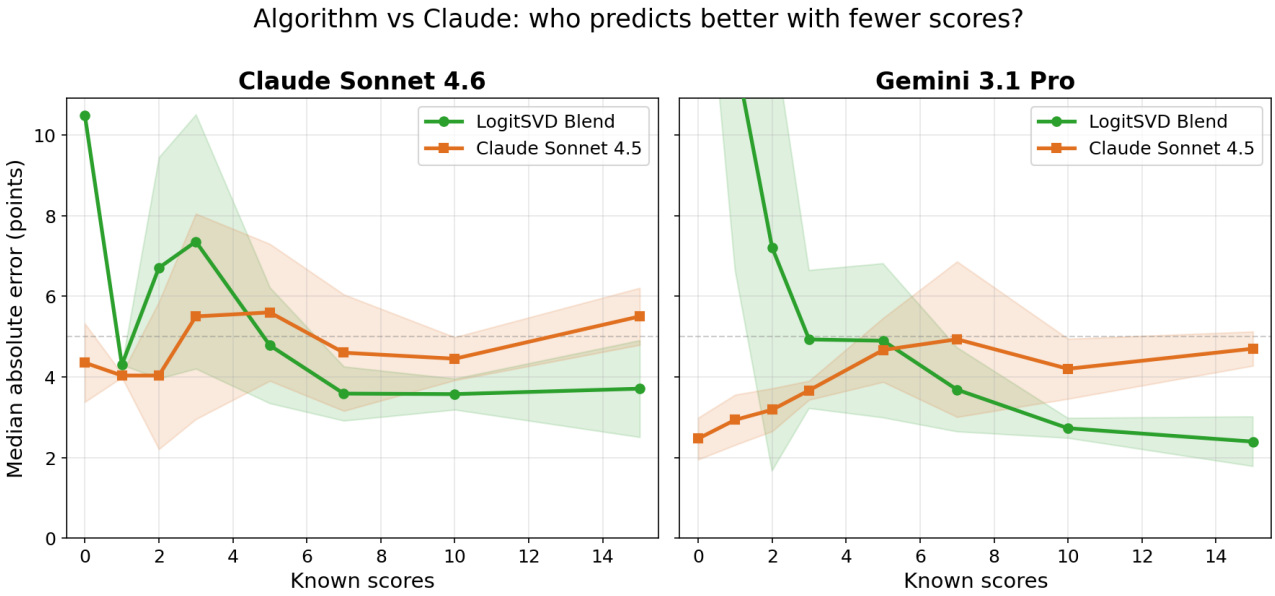
Competition math (IMO, USAMO) is bimodal: a model either solves olympiad problems or it doesn't. You can't interpolate a step function I guess!

Wait! Can Claude predict the scores without the algorithm?

OK, this next part isn't purely science because some of these models were released before Sonnet's training cutoff, so Claude might just "know" certain scores. But I was curious: what if you don't use SVD and regression-type predictors and instead use a one-gazillion-parameter model for that? In this case, Claude.

For $0.87, we gave Claude Sonnet the partially-filled matrix as a big CSV and asked it to predict the missing entries. Same holdout, same evaluation.

But that comparison undersells what's interesting. The figure below shows what happens when you vary how much the predictor knows. Take a model, hide all its scores, then reveal them one at a time. After each reveal, both the algorithm and Claude predict the rest.



At k=0, zero known scores, just the model's name -- Claude already predicts Gemini 3.1 Pro's benchmarks to within 2.5 points!! The algorithm at k=0 can only guess column averages, and is off by 17 points. But by k=5, the algorithm catches up.

The takeaway: Claude's world knowledge is worth about 5 benchmark scores of information. After that, linear algebra wins.

When should you use this?

Three options for getting a number on "how does my model do on benchmark X":

Option A: Run the benchmark.

Cost: $50 to $100k? (I know for a fact TB 2.0 can be O(100k) for some models and harnesses). BUT! You get the true score.

Option B: Predict from the matrix.

Cost: ~$0. But with almost everything that is free, it has its issues, in this case your score is within about 5 points. Not great, not terrible.

Option C: Run a cheaper model as proxy?

Cost: less than A. But you get the right answer about the wrong model. Hmm maybe not?

But to be serious, the decision is a function of cost and error tolerance. If SWE-bench costs $5,000, and 6% error is fine for a go/no-go, just predict it I suppose. If AIME 2026 costs $2, run it.

Am I convinced? Hmmm.

For frontier labs, honestly probably not really. They'll run every eval regardless, I would assume. But for smaller groups trying to train a new model, having a quick sanity check could be helpful? I don't know.

But it doesn't matter, because this was fun to try!

It's kind of hilarious that in pretty much any setting where you have an incomplete matrix to complete, it almost always turns out to rank 2 or 5. Matrix completion for the win!

Perhaps the deeper meaning is that most of what we call "evaluation" is measuring the same two things (because we said rank 2, remember?) over and over, and the benchmarks that escape this pattern are the ones testing new capabilities (hello friend at Terminal-Bench!). But we've nevertheless just convinced ourselves we need to climb 100 evals for every model.

So maybe there's also a message here for benchmark creators. If your eval correlates or anti-correlates with another one it's not that meaningful. It has to be orthogonal.

The full matrix, prediction code, and results are at github.com/anadim/llm-benchmark-matrix.

Joint work with Claude Code and Codex. The matrix was assembled from official model announcements, technical reports, leaderboards, and evaluation platforms. Every entry has a source URL. The code was written by Claude Code, audited by Codex, with the audit rebutted by Claude Code and counter-rebutted by Codex. They kinda had a bit of a fight about it, and I literally told them don't fight, be constructive lol.

I mostly gave prompts. Sort of.