Inference Scaling and the Future of Training Compute Thresholds

MIRI Technical Governance Fellowship | Fernando Ruiloba

Reasoning models

In late 2024, OpenAI released a new type of model, a "reasoning" model that spends more time "thinking" before responding. Reasoning models allocate additional compute at inference time to generate chains of reasoning. This chain of thought helps the model break down complex tasks, recognize its own mistakes, and correct its strategies before providing an answer. On certain benchmarks (GPQA, MATH, and Mock AIME), reasoning models achieved a 10x compute equivalent gain (CEG), meaning a non-reasoning model of similar scale would need 10 times more training compute to match the same level of performance.

Inference scaling for deployment

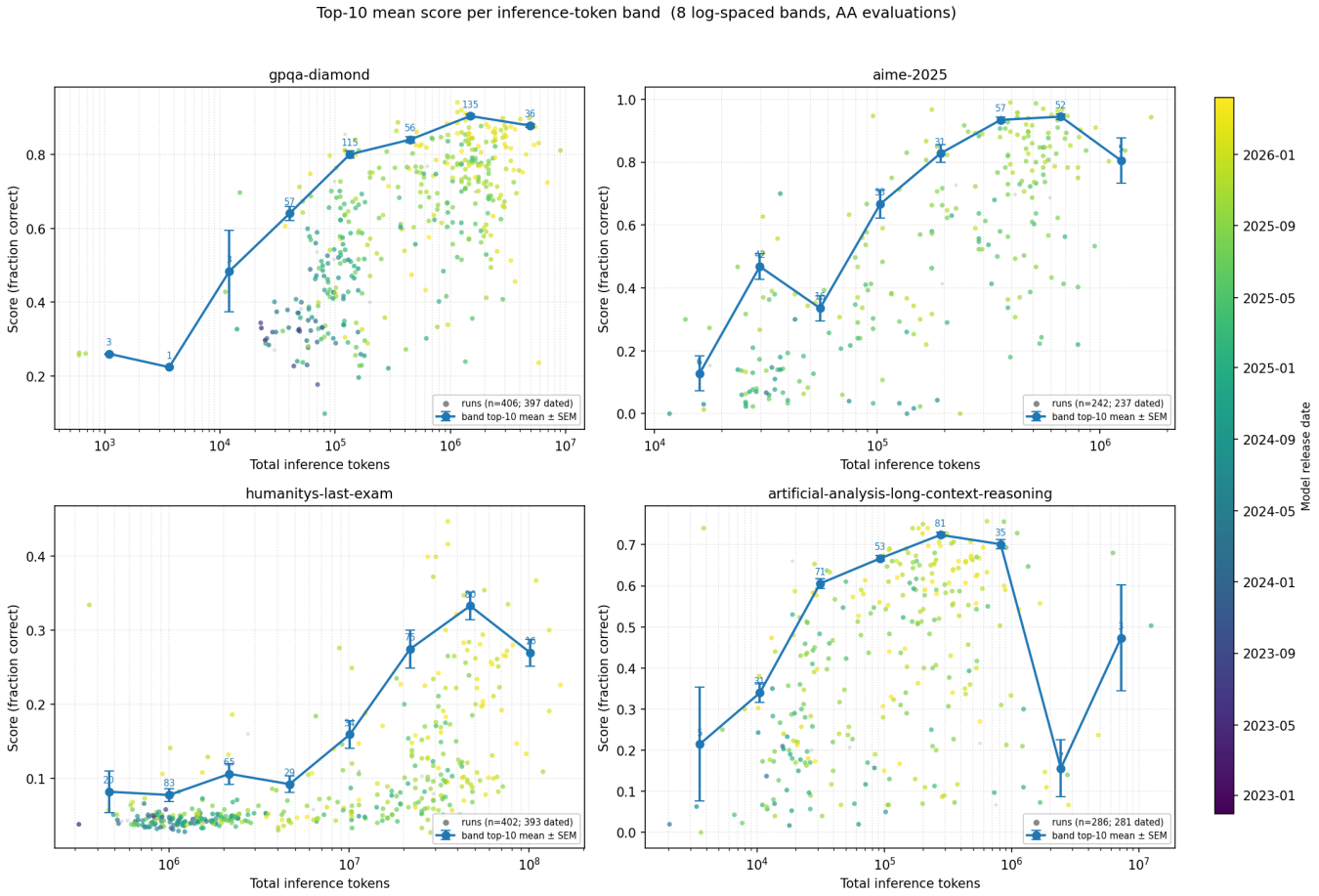
This leads us to Toby Ord's first argument for how scaling inference could "derail the current paradigm of AI governance via training compute thresholds". This argument basically boils down to: if scaling inference gets you more capabilities at a lower cost compared with pre-training, then we can just redirect our compute resources towards inference, making training compute less relevant. My intuition is that it doesn't work like that, so I decided to perform some experiments.

Inference scaling has decreasing returns

It would be worrisome for training compute thresholds if we could just keep scaling inference compute and keep getting better results. The reality is that inference scaling has diminishing returns, and performance plateaus past a certain number of tokens.

The graph above plots model scores in different benchmarks against the number of inference tokens used to reach that score. Each model is represented as a colored circle, where the color represents the model release date. The graph also draws a trend line over the average score for the top-10 models for each inference token budget.

The trend-line shows the decreasing returns behavior. The graph also shows that some models achieve much higher performance than others for the same inference token budget. Another aspect is that newer models tend to score higher across benchmarks and can use more inference tokens. These results suggest that there are other variables at play driving inference scaling behavior.

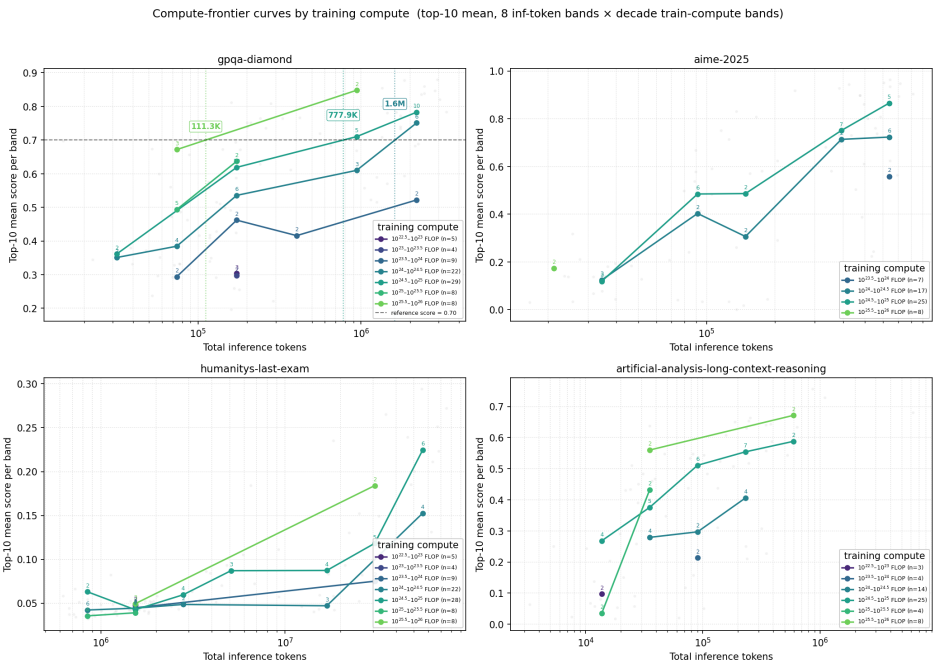


Inference and training are tightly coupled

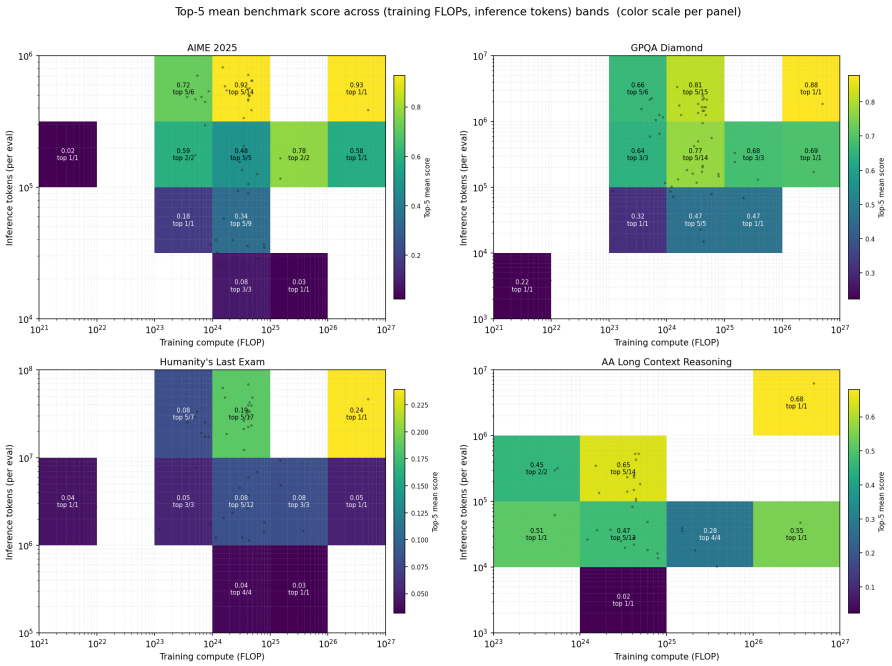
While scaling inference might be more efficient per unit of compute than scaling training, experiments suggest that models using a bigger amount of training compute still yield higher performance at any given inference token budget. To get top performance, both training and inference scaling are needed.

While they weren't explicitly talking about reasoning models, Villalobos and Atkinson explained the relationship between training and inference compute. They estimated that it was possible to increase the amount of inference compute by 1-2 orders of magnitude (OOM), in exchange for saving ~1 OOM in training compute while maintaining similar performance. This trade-off has limits, of course. We saw that scaling inference beyond a certain threshold for a given training compute doesn't yield any more results. In Villalobos and Atkinson's post the opposite is also true: scaling training indefinitely without scaling inference leads to a plateau in capabilities. They depict this relationship as L-shaped iso-capability curves, where the trade-off is effective at certain inference/training budgets.

In my experiment, to get the same GPQA-diamond score as a 10e25 model, a 10e24 model needs to use 1.6M inference tokens, which is 1.4 OOM more than the 10e25 model. This is in line with Villalobos and Atkinson's estimations.



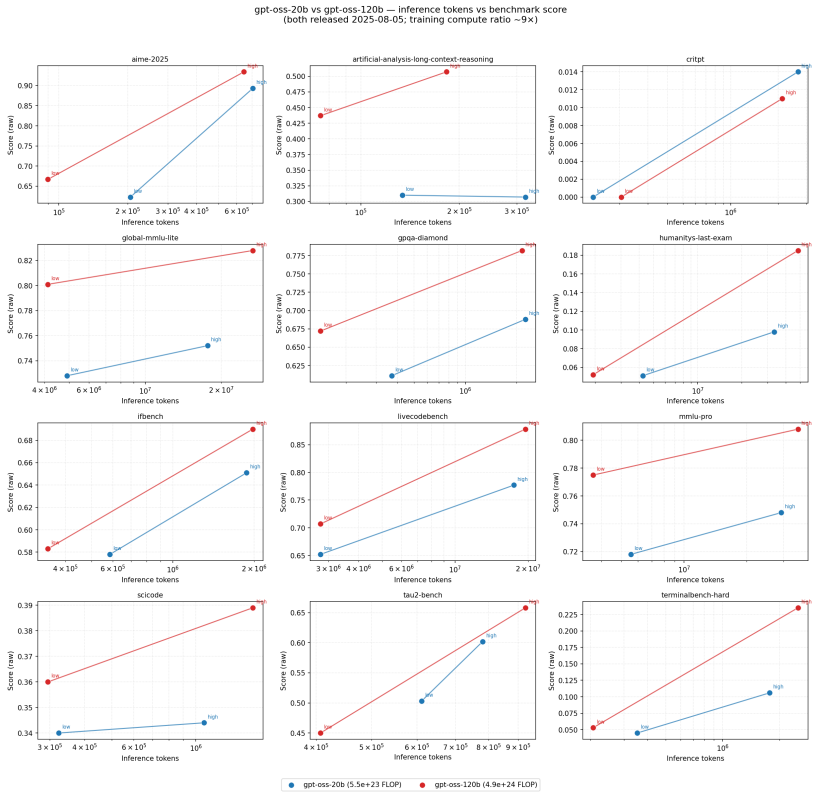
The graphs below show the top-5 mean scores across benchmarks, binned by training compute and inference tokens. Even with limited data, we can start to trace the L-shaped iso-capability curves for different amounts of training compute and inference tokens. Take Humanity's Last Exam, for example: models trained with 10e23-10e24 FLOPS needed one OOM more inference tokens to reach the same score as models trained with 10e24-10e25 FLOPs.



Inference scaling is domain-dependent

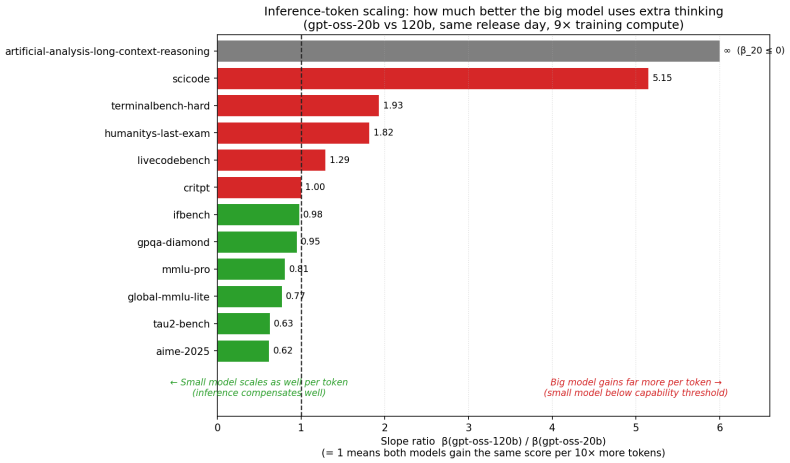
Villalobos and Atkinson's trade-off rule is helpful for understanding the relationship between training and inference compute. However, the reality might be a little more nuanced. The graph below shows

the inference scaling behavior for two models, gpt-oss-20b and gpt-oss-120b. They were both released on the same date but differ in their number of parameters (21B total / 3.6B active vs 117B total / 5.1B active) and the amount of compute used to train each of them (the 120B model used 9x more training compute). Usually, for both models, the score increases as we scale the inference token budget. However, we can see some crucial differences. First, the bigger model gets better scores across all inference token budgets. Second, and more interestingly, the score increases at different rates depending on both the model size and the benchmark.



This result is something Ord already suspected: that scaling inference is much more useful for some tasks than others. Villalobos and Atkinson also theorized about different tradeoff rules for different types of tasks.

To better compare the differences in inference scaling behavior, the figure below plots the ratio between the slope of the bigger model and the smaller one. It shows that, for the Scicode benchmark, the bigger model is able to get 5.15 times more performance per additional inference token than the smaller model. On the other hand, the smaller model derives more capability gains per inference token used than the bigger one on tau2-bench.



One explanation is that tasks where the solution is objectively verifiable (such as coding tasks) are more amenable to novel reinforcement-learning techniques such as RLVR due to them having a clear optimization objective (i.e. create code that passes unit tests). The results are mostly in accordance with this hypothesis. Scicode measures the capabilities of language models to generate code for solving scientific research problems while Tau2 bench is a communication skills test. The main exception is AIME, which consists of high school math problems. A plausible reason is that the problems are tractable enough for the smaller model to score well without any further scaling.

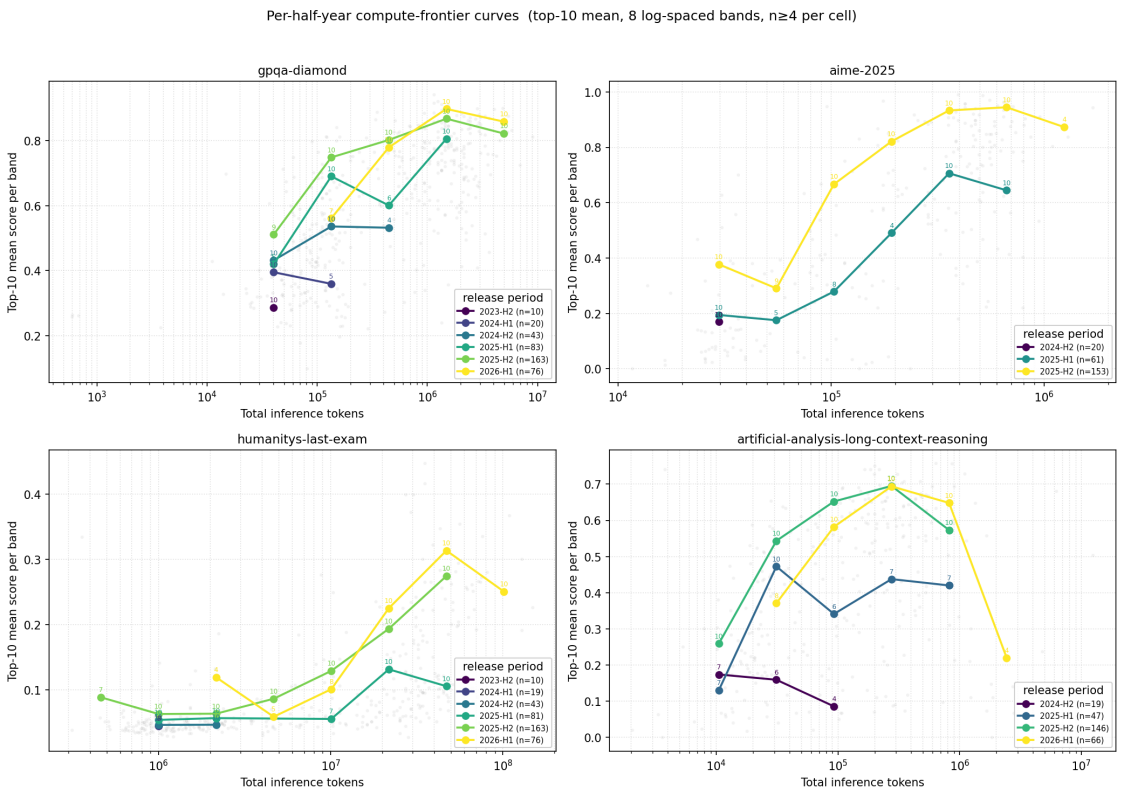
Inference scaling over time

So far, there is evidence that inference scaling is still dependent on the amount of training compute, meaning that training compute thresholds remain a valuable tool for AI governance. This relationship could erode over time, however, undermining the case for thresholds. I decided to perform more experiments to understand how this relationship has changed over time.

Diminishing returns are persistent across time

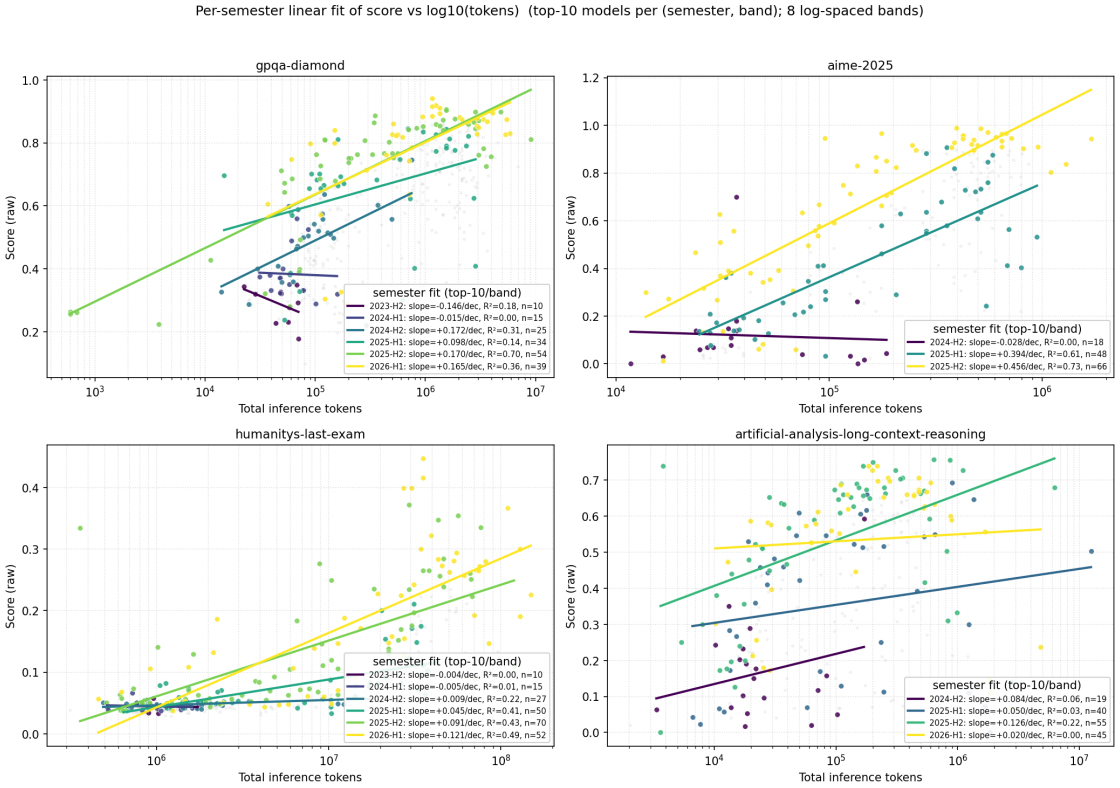
A true source of worry would be whether diminishing returns are becoming "less diminishing". This would mean that we could scale inference indefinitely and keep obtaining capability gains. I performed a naive analysis by plotting the mean score from the top 10 models across different inference token budgets for different periods of time to see whether the diminishing returns behavior has changed.

It looks like the curves are moving upwards, meaning that models are getting more performance for the same number of inference tokens. However, all periods still display diminishing returns. For now, there is no sign of infinite gains through unbounded inference scaling.



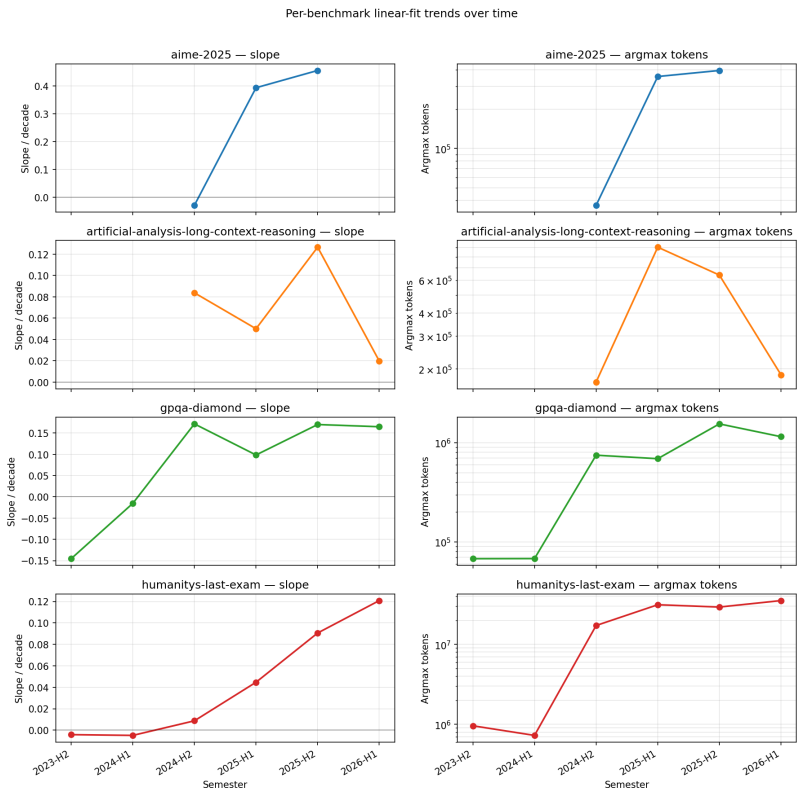
Inference scaling is getting more efficient and is able to get returns for longer

We saw that the curves shift upwards over time, but we are also interested in whether these curves are getting steeper. This can tell us not only how much the baseline performance increases for a given number of inference tokens, but also at what rate we can scale inference tokens to gain performance. The image below fits a straight line across model performance for a given inference token budget. As we saw previously, model performance starts to decrease after a certain number of inference tokens is reached, so I decided to leave out models whose performance was lower than the peak score while using more inference tokens. This is fine since the idea is to see general trends in the rates at which models are able to use tokens to gain capabilities. The image does not show a general trend across semesters in the rates at which models increase performance as they scale inference token usage.



To be sure, the image below plots the rate at which score increases per additional inference token (the slope from the lines above) for every semester. We can see on the left column that changes in the slope are mixed across benchmarks. AIME shows a consistent deceleration, while the others show a mixed result, with rates fluctuating over time. All benchmarks show a decline between the first and second semester of 2024. This is an interesting result given that the first reasoning model was released in the second semester of 2024. The slopes are also relatively more stable and have less variation in the second semester of 2024 and all of 2025. This seems to be the case for AIME, GPQA and HLE. This is due to the fact that we have more data for those time periods.

We can also look at the maximum number of tokens that can be used before getting diminishing returns. This is important because it means that models are able to extract capability gains for longer periods. This is plotted on the right column of the image below. We can see that, over time, models are able to scale their inference tokens for longer before plateauing or decreasing in performance, with AALCR being the only significant exception. The stars represent the maximum number of tokens used by any model for each period. For most periods, using the most amount of inference tokens did not mean obtaining the highest score, showing that inference tokens were scaled beyond the point after which decreasing returns begin to appear. Some benchmarks monotonically increased the maximum amount of inference tokens used per semester (AIME, HLE) while others showed a slight decline (AALCR, GPQA).



Inference scaling for training more capable models

Ord's second argument might be more worrisome for training compute thresholds. He describes a recursive self-improvement loop in which models leverage inference scaling to produce high-quality synthetic data, which is then used to train a superior model of comparable scale. This idea, also called iterated distillation and amplification, is similar to the mechanism behind Alpha Go Zero, and was independently discovered by Anthony et al. and, in the context of AI Safety, by Cristiano. While the exact mechanism of how this could work is still uncertain, we can still look at current trends to understand the role of inference scaling in the training process.

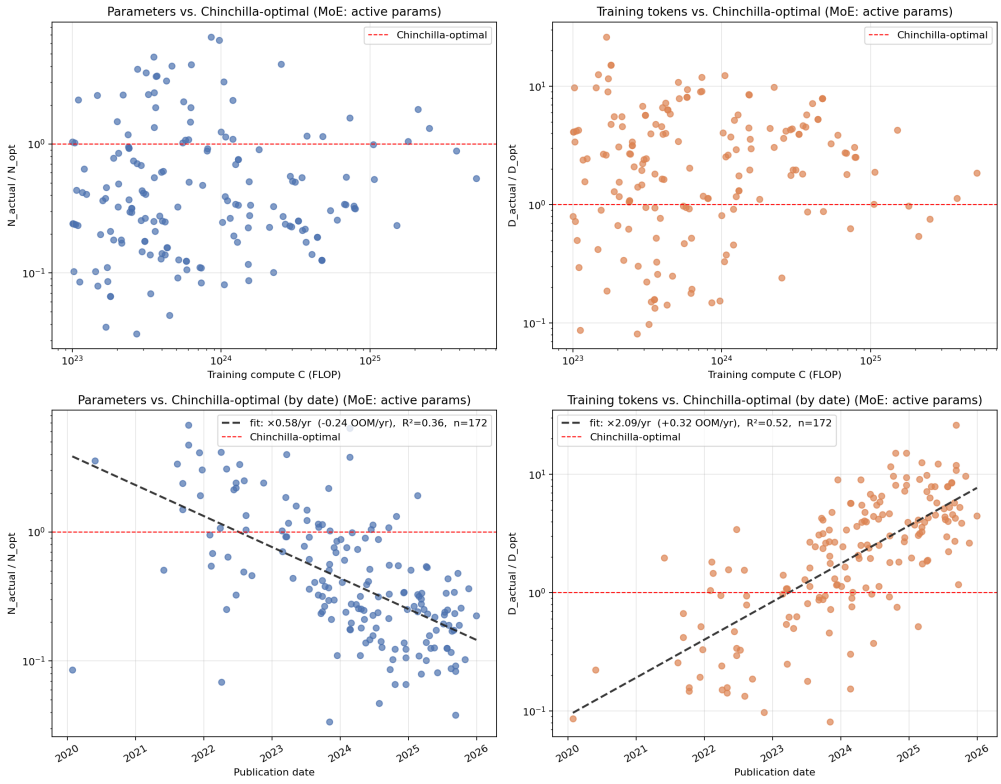
Models are being optimized for inference

Since generating vast amounts of synthetic training data could be expensive (especially for high quality data requiring long chains of thought and inference scaling), we care about cheap and efficient inference. Fast and cheap inference is driven by several factors, including hardware improvements and algorithmic efficiency gains.

Beyond these, AI labs can also optimize models for inference-heavy tasks. The main way to do this is by reducing the model size. Smaller models make inference faster and cheaper and are easier to use for researchers and developers with limited GPU resources. One approach for reducing model size is pruning, which involves the periodical removal of irrelevant weights during training. Another approach, explained by de Vries, is varying the scaling policy. For a given training compute budget, AI labs can decide to overtrain a smaller model on more tokens than Chinchilla scaling laws would suggest.

The graph below shows this dynamic. On the left column, I compare the actual number of parameters

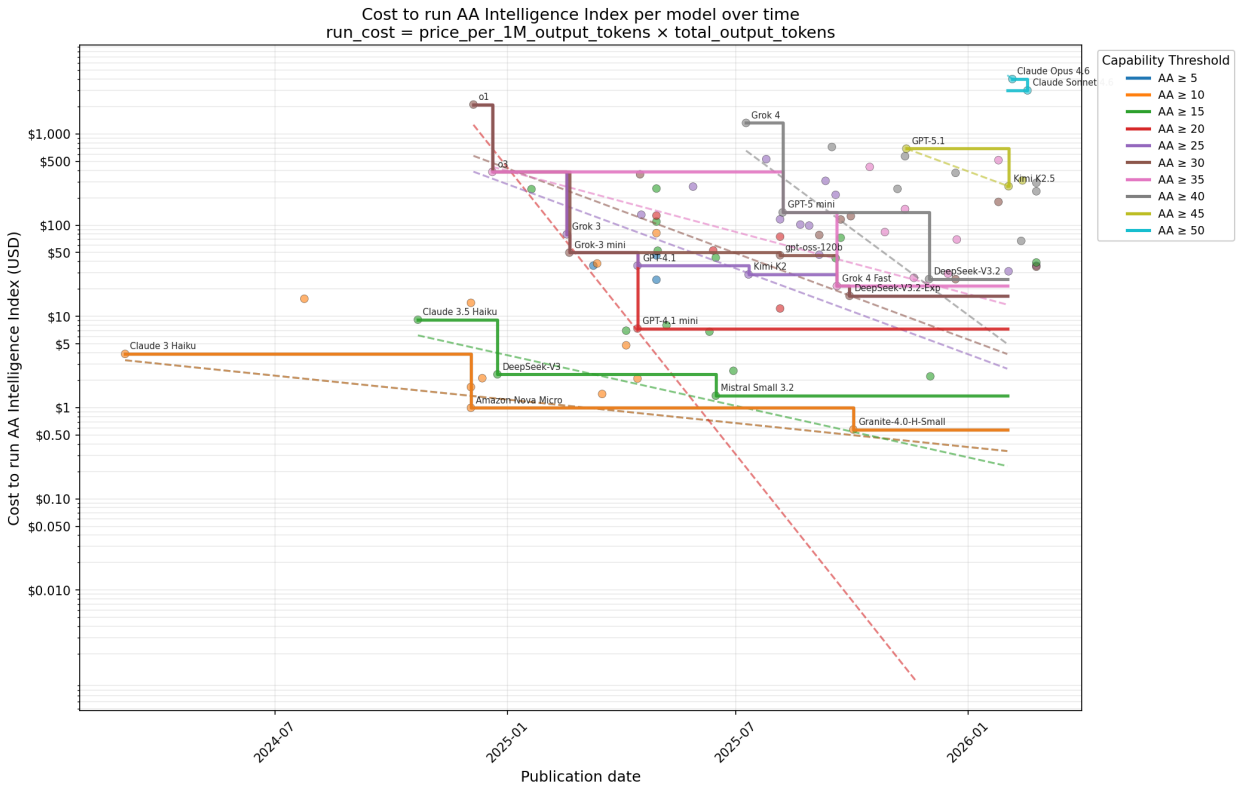
against the optimal number of parameters each model should have according to Chinchilla scaling laws. Over time, it is clear that models are getting smaller against their Chinchilla optimal number of parameters. The fitted line suggests models shrink by about x0.58 per year, or roughly half every 15 months. On the right side of the graph is the number of tokens used to train each model. The trend is also clear. Models are being increasingly trained with more tokens than Chinchilla optimality would suggest.



While this trend doesn't seem to have plateaued, an interesting question is how much smaller can models get. Sardana et al were able to shrink models down to 10k tokens per parameter without an impact on loss indefinitely while De Vries estimates that the critical model size is around 30% of the Chinchilla optimal model. Huang et al. explain why such a critical model size exists, finding that a model below the critical size cannot learn the rare and complex tasks in the tail of the data distribution even with unlimited data, because the tasks that dominate the training data consume the model's limited capacity, leaving too little for the rare ones.

Inference costs are falling

Shrinking model sizes, hardware efficiency gains, and other sources of algorithmic progress have together driven inference prices down sharply. In his analysis, Scher estimated that the cost per 1M tokens is falling at around 82x per year. He also mentions that per-token costs is not as relevant as the cost to run benchmarks (measured as the cost per token * number of tokens). I agree because, as seen in previous graphs, some models are more token efficient than others. I decided to expand his analysis and calculate the cost to run the Artificial Analysis Intelligence Index (AAII), since they publish the number of tokens used to run this benchmark. The result is a tiny bit more conservative than the per-token rate, with the cost to run AAII falling at a rate of 73x per year.



Implications for AI Governance and Compute Thresholds

It remains unclear how training compute shapes both the rate at which a model improves with more inference tokens and the point at which further tokens no longer help. What does appear strong is the relationship between training compute and token efficiency, meaning the number of tokens a model needs to reach a given performance level. This dependence between inference scaling and training compute suggests that training compute thresholds remain a viable governance instrument.

Empirical analysis shows that saving 1 OOM of training compute costs roughly 1.4 OOM of inference compute, consistent with Villalobos and Atkinson's estimates. Diminishing returns also persist across all observed release periods, with no benchmark showing the inflection toward unbounded inference scaling that would invalidate thresholds outright. These two observations undermine Ord's worry that a 10^24 FLOP model could be inference-scaled into a 10^27 FLOP.

Domain dependence narrows the threat surface further. Preliminary results suggest that inference scaling is more effective on verifiable benchmarks, such as AIME-2025 and Tau2-Bench which involve math and coding, than on non-verifiable ones. Governance should therefore prioritize evaluations targeting verifiable-task capabilities, since these are where inference compute most directly substitutes for training, at least in the near term.

However, the relationship between training and inference compute is not guaranteed. Slopes are steepening on some benchmarks and argmax tokens are rising, meaning models extract more capability per inference token and sustain gains for longer before plateauing.

On Ord's second argument regarding iterated distillation and amplification, the evidence is more circumstantial. Models are consistently smaller and more overtrained than Chinchilla-optimal, which is

consistent with labs optimizing for cheap inference, exactly the configuration that synthetic data pipelines require. Inference costs are also falling sharply, 73x per year on AAII. Together these trends mean the marginal cost of producing high-quality reasoning traces for post-training is collapsing. Whether labs are actually closing an iterated distillation and amplification loop is not directly observable from this data, but the preconditions are clearly in place.

Based on this data, the evidence that inference scaling will inevitably erode training compute thresholds is mixed. Ord's first argument, on scaling inference on deployed models, is weakened, while his second argument, on iterated distillation and amplification, remains a challenge. Training compute thresholds are still useful, but inference scaling trends are worth monitoring so the thresholds can be adjusted as needed. The dependence of inference scaling on training compute should also be tracked continuously, since any decoupling would undermine the assumption that a model's training compute bounds its eventual capabilities.

Limitations

The dominant limitation of this analysis is data scarcity. Public evaluation runs are sparse when sliced by training compute, inference tokens, and release period simultaneously, which prevents finer-grained analysis of the two questions that matter most for governance. The first of these questions is whether diminishing returns are themselves changing over time. The current analysis shows that diminishing returns persist across all observed periods, but the data is too thin to tell whether the *rate* of diminishing returns is softening within a fixed training compute band. The second is whether the training-inference trade-off ratio is shifting. The 1.4 OOM trade-off observed here is a snapshot, and tracking how this ratio evolves over time requires repeated measurements across the same training compute bands, which the current dataset does not support.

Several smaller limitations are downstream of this same scarcity. The benchmark suite is narrow and skewed toward STEM and verifiable tasks, since these are the benchmarks with enough public evaluation runs to support the analysis. Training compute values are estimates rather than disclosed figures for most closed models, with error bars wide enough to affect the trade-off ratio. Inference compute is approximated by inference tokens which loses precision across architectures with different per-token FLOP costs.

Appendix

For each of the ten graphs above, I list the source script, the input datasets it reads, the processing steps it applies, and the output CSV that holds the underlying numbers. Graphs 1-9 live in this repo (feruiloba/miri-fellowship-inference-scaling) on the master branch. Graph 10 lives in the sibling repo (feruiloba/miri-fellowship-alg-progress) on the main branch.

Shared inputs

Most graphs read from a small set of curated CSVs under data/:

aa_evaluations_combined.csv - per-benchmark evaluation runs scraped from Artificial Analysis. One row per (model_slug, benchmark) with score_raw, total_output_tokens, and related fields. This is the dominant data source for the inference-token analyses.

artificial_analysis_llm_stats.csv - per-slug model metadata, used here for release_date

aa_output_tokens.csv - per-model token counts used to compute AA-Index.

epoch_all_ai_models.csv - Epoch AI's curated model table. Source of Training compute (FLOP) for closed-weights models.

merged_datasets.csv - internal join of Epoch + AA tables. Provides Training compute (FLOP), Publication date, parameter counts, and an is_moe flag for the Chinchilla analysis and the iso-capability plot

Two graphs are produced by scripts that read derived CSVs written by other scripts in this repo; those producer-consumer chains are called out explicitly below.

1. Inference scaling has diminishing returns

Graph: top10_per_token_band.png

Script: top10_per_token_band.py

Table: top10_per_token_band.csv

Inputs: aa_evaluations_combined.csv, artificial_analysis_llm_stats.csv (release dates for the colour scale).

Method: For each of four target benchmarks (GPQA-Diamond, AIME 2025, Humanity's Last Exam, AA long-context-reasoning), filter to runs with a valid positive total_output_tokens and score_raw. Bucket runs into 8 log-spaced inference-token bands spanning the observed token range. Within each band, take the top-10 runs by score_raw and compute their mean and standard error. Plot one panel per benchmark: a single aggregate line (top-10 mean with ±SEM whiskers) overlaid on a scatter of every run coloured by release date. The CSV has one row per (benchmark, band) with n, mean_top10_score, sem, and the band bounds.

2. Iso-capability bands by training compute

Graph: top10_grouped_by_training_compute_band.png

Script: top10_grouped_by_training_compute_band.py

Table: top10_grouped_by_training_compute_band.csv

Inputs: aa_evaluations_combined.csv plus epoch_all_ai_models.csv for Training compute (FLOP). Slugs are matched against the Epoch table with a fallback that strips parenthesised suffixes from model names.

Method: Same skeleton as graph 1 (4 benchmarks, 8 log-spaced inference-token bands, top-10 mean per band), but runs are first grouped into half-decade training-compute bands with edges at every 0.5 dex from 10^{22} to $10^{2?}$ FLOP. Each compute band becomes a separate viridis-coloured curve so the L-shape across (training compute x inference tokens) is visible. For GPQA-Diamond the script also draws a horizontal reference line at score = 0.7 and annotates, for each curve, the interpolated token count at which it crosses that line this is the source of the "1.4 OOM trade-off" figure quoted in the writeup. MoE models are included. The CSV has one row per (benchmark, compute_band, token_band) cell.

3. Iso-capability frontier (training FLOPs vs inference tokens)

Graph: train_flops_vs_inference_tokens_top5_score.png

Script: train_flops_vs_inference_tokens_top5_score.py

Table: train_flops_vs_inference_tokens_top5_score.csv

Inputs: aa_evaluations_combined.csv joined to merged_datasets.csv on AA_slug for Training compute (FLOP).

Method: Per benchmark, restrict to the top-5 scoring runs at each combination of training-compute and inference-token band. The chart plots training FLOP on the X axis vs inference tokens on the Y axis, with marker colour encoding the score and contours implied by the top-5 mean per cell this directly visualises the trade-off frontier the writeup refers to as "L-shaped". The CSV records each (training compute band, inference token band, benchmark, top-5 score).

4. Inference scaling is domain-dependent (gpt-oss case study)

Graph: gpt_oss_effort_comparison.png

Script: gpt_oss_effort_comparison.py

Table: No CSV written by this script. The underlying slope numbers quoted in the writeup come from the table produced by graph 5 (gpt_oss_training_inference_tradeoff.csv, columns beta_20, beta_120).

Inputs: aa_evaluations_combined.csv, restricted to the four gpt-oss slugs: gpt-oss-20b, gpt-oss-20b-low, gpt-oss-120b, gpt-oss-120b-low.

Method: For every benchmark that has both effort levels for both model sizes (low/high effort x 20b/120b), draw a 2-point line per model in (log10 total_output_tokens, score_raw) space. The two lines per panel directly visualise per-model inference-token slopes; the writeup compares them visually for scicode vs tau2-bench.

5. Training-vs-inference substitution rate (gpt-oss)

Graph: gpt_oss_training_inference_tradeoff.png

Script: gpt_oss_training_inference_tradeoff.py

Table: gpt_oss_training_inference_tradeoff.csv

Inputs: aa_evaluations_combined.csv, same gpt-oss slug filter as graph 4. Training-compute ratio hard-coded as log10(4.94e24 / 5.49e23) ? 0.954 ? 9x (from Epoch's training-compute estimates for the two gpt-oss models).

Method: For each benchmark with all four corners (size x effort) present, compute four finite-difference slopes from the 2x2 grid:

?_low = (s(120,low) - s(20,low)) / log10(9) ?score/?log(C_train) at low effort

?_high = (s(120,high) - s(20,high)) / log10(9) ?score/?log(C_train) at high effort

?_20 = (s(20,high) - s(20,low)) / ?log(tokens_20b) ?score/?log(C_inf) at 20b

?_120 = (s(120,high) - s(120,low)) / ?log(tokens_120b) ?score/?log(C_inf) at 120b

The chart shows the inference-slope ratio ?_120 / ?_20 as a horizontal bar per benchmark. Bars \> 1 indicate the bigger model gains more per token; bars \< 1 indicate the smaller model uses extra thinking more productively. The writeup quotes the 5.15x ratio for scicode straight from this CSV (r_LL/r_UR and the ? columns).

6. Diminishing returns over time

Graph: top10_grouped_by_release_period.png

Script: top10_grouped_by_release_period.py

Table: top10_grouped_by_release_period.csv

Inputs: aa_evaluations_combined.csv joined to artificial_analysis_llm_stats.csv on slug for release_date.

Method: Same template as graph 1 (4 benchmarks, 8 log-spaced inference-token bands, top-10 mean per band), but runs are grouped by 6-month release period (YYYY-H1 / YYYY-H2). Each release period becomes a viridis-coloured curve. Cells with n \< 4 are dropped, and periods with fewer than 5 total runs are excluded entirely. The chart shows the curves shifting upward over time (capability per token improving) while still bending (diminishing returns persisting). One CSV row per (benchmark, period, token_band).

7. Per-semester linear fits

Graph: linear_fits_by_release_semester.png

Script: linear_fits_by_release_semester.py

Table: linear_fits_by_release_semester.csv

Inputs: aa_evaluations_combined.csv joined to artificial_analysis_llm_stats.csv for release dates.

Method: Same per-cell aggregation as graph 6 (8 log-spaced bands, top-10 per cell, per semester). On top of those cell means, fit an OLS line score = a + b · log10(tokens) per (benchmark, semester). Plot the fitted lines (viridis by semester) over the top-K cell scatter. The CSV records slope_per_decade, intercept, r2, n_fit_points, period, and benchmark one row per (benchmark, semester). This CSV is the input to graph 8.

8. Slope and argmax-token trends across semesters

Graph: semester_fit_trends.png

Script: semester_fit_trends.py

Table: semester_fit_trends.csv

Inputs: linear_fits_by_release_semester.csv (the table from graph 7) plus aa_evaluations_combined.csv and artificial_analysis_llm_stats.csv for the argmax-tokens computation.

Method: Two columns per benchmark row: the left column plots slope_per_decade over semester time (linear y-axis, with slope = 0 reference line); the right column plots argmax-tokens per (benchmark, semester) on a log scale, where argmax-tokens is the total_output_tokens of the single highest-scoring run in that cell. The CSV adds argmax_tokens, argmax_score, argmax_model, delta_slope, delta_intercept, and argmax_token_ratio (period-over- period change) to the per-semester fit table. This is the chart the writeup uses to argue "slopes are steepening and argmax tokens are rising" across most benchmarks.

9. Chinchilla optimality drift (parameters and training tokens)

Graph: chinchilla_param_token_vs_optimal_moe_active.png

Script: chinchilla_param_token_vs_optimal_moe_active.py (uses helpers from src/chinchilla_analysis/_chinchilla.py)

Table: chinchilla_param_token_vs_optimal_moe_active.csv

Inputs: merged_datasets.csv (read by _chinchilla.py). Provides Training compute (FLOP), Publication date, parameter counts, training tokens, and an is_moe flag.

Method: For each model with known training compute, parameters, and training tokens, compute the Chinchilla-optimal N_opt, D_opt from C = 6 · N · D and the Hoffmann et al. exponents, then take the ratios N_actual / N_opt and D_actual / D_opt. For MoE models, N_actual is replaced by N_active = C / (6 · D_actual) so the model is compared at its inference-time parameter count rather than its total parameter count. The chart is a 2x2 grid: top row plots the two ratios against training compute (log-log); bottom row plots them against publication date, with an OLS fit drawn on the date panels. The "x0.58 per year" figure quoted in the writeup is the 10**slope factor from the parameter-ratio-vs-date fit. The CSV is one row per model with the input columns plus the derived ratios and an arch column (dense or moe_active).

10. Run-cost frontier (sibling repo)

Graph: run_cost_frontier.png

Script: 03_cost_to_run_catchup_by_tokens.py

Table: run_cost_frontier_table.csv

Inputs:

merged_datasets.csv (publication dates and AA-Index per model)

aa_output_tokens.csv (per-model token counts for running AAII)