Quantifying Elicitation of Latent Capabilities in Language Models

Elizabeth Donoway
Anthropic
University of California, Berkeley
donoway@berkeley.edu

Hailey Joren
Anthropic
haileyj@anthropic.com

John Schulman
Thinking Machines
(pending)

Arushi Somani
Anthropic
arushi@anthropic.com

Henry Sleight
Constellation
henry@constellation.org

Julian Michael
Meta
(pending)

Michael DeWeese
University of California, Berkeley
deweese@berkeley.edu

Ethan Perez
Anthropic
ethan@anthropic.com

Fabien Roger
Anthropic
fabien@anthropic.com

Jan Leike
Anthropic
jan@anthropic.com

Abstract

Large language models often possess latent capabilities that lie dormant unless explicitly elicited, or surfaced, through fine-tuning or prompt engineering. Predicting, assessing, and understanding these latent capabilities pose significant challenges in the development of effective, safe AI systems. In this work, we recast elicitation as an information-constrained fine-tuning problem and empirically characterize an upper bound on the minimal number of parameters needed to reach a certain task accuracy. We find that training as few as 10–100 randomly chosen parameters—several orders of magnitude fewer than state-of-the-art parameter-efficient methods—can recover up to 50% of the performance gap between pretrained-only and fully fine-tuned models, and 1,000s to 10,000s of parameters can recover 95% of this performance gap. We show that a logistic curve fits the relationship between the number of trained parameters and the fraction of the performance gap recovered. To help explain this behavior, we consider a simplified picture of elicitation via fine-tuning where each trainable parameter serves as an encoding mechanism for accessing task-specific knowledge. We observe a relationship between the number of trained parameters and the effectiveness in eliciting latent capabilities, offering a potential route to distinguish elicitation from teaching.

1 Introduction

Many recent efforts in advancing the capability frontier of large language models (LLMs) focus on post-training methods, such as fine-tuning, reinforcement learning with human feedback or prompt engineering. In many cases, these methods easily achieve significant performance gains through small adjustments to a model, indicating existing capabilities that need only be unlocked rather than taught. As a result, evaluations intended to benchmark abilities can underestimate a model's true capabilities

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

[1], leading to an “elicitation gap” between their actual ceiling performance and the capabilities they ordinarily demonstrate. This raises challenges in accurately predicting the model’s behavior once released, with implications for ensuring deployed models aren’t capable of causing harm.

Prior work that aims to extrapolate LLM capabilities has focused on the impact of scale—compute, data, or model size—on the improvement of model capabilities. We propose a complementary framework that takes these as constant and instead varies the number of parameters available for training during fine-tuning. We use this as a direct proxy for the information budget usable during the model’s fine-tuning. We then recast elicitation as a question of the minimal amount of information a model requires to demonstrate a capability, and propose this quantity as a signal of whether the capability already existed within the model.

We find that fine-tuning only a tiny fraction of a model’s parameters often suffices to recover most of its latent capabilities, even when those parameters are selected completely at random (Figure 1). Training as few as ~10-10^5 randomly-chosen low-rank adapter weights can close 50-95% of the performance gap between zero-shot and fully fine-tuned LLMs across arithmetic reasoning, question answering, and natural language generation tasks. We identify a Pareto frontier of accuracy versus parameter count when eliciting a model, which assumes no prior information about that model, and uncover a logistic relationship in log-parameters: initially steep gains as parameters are added, followed by an asymptotic plateau. Further analysis reveals qualitative differences that may be useful in distinguishing elicitation, in which latent capabilities are surfaced, from teaching, wherein the model learns a new skill.

Our main contributions are:

1. **Elicitation Frontiers.** We empirically characterize how elicited performance scales with the number of randomly selected trainable parameters across five benchmark tasks and three model sizes and find that training only ~10^1-10^5 parameters can achieve 50-95% of the performance achieved by fine-tuning the full model for the task.
2. **Information-Theoretic Interpretation of Elicitation.** We present a picture of elicitation built upon Rissanen’s Minimum Description Length (MDL) principle [2] that uses a singular currency—bits—for reasoning about model accuracy, information flow, and how much must be specified to a model for a latent capability to manifest, if one exists.
3. **Distinguishing Elicitation From Teaching.** We observe qualitative differences in the information models use to represent tasks that require pre-existing versus absent capabilities, proposing that elicitation can be understood as a model’s ability to develop short descriptions of how to effectively utilize latent skills.

2 Related Work

Parameter-Efficient Fine-Tuning. Parameter-efficient fine-tuning (PEFT) methods that freeze most of a model’s weights while training only a small auxiliary module have become standard practice for adapting large language models to downstream tasks. Adapters [3] insert lightweight MLP blocks, while prefix-tuning [4] prepends trainable vectors, and LoRA [5] injects low-rank adapters. Despite using significantly fewer parameters than full fine-tuning, LoRA often nearly matches its performance, indicating that adaptation information can be highly compact [6]. Recent approaches like DoRA [7], PiSSA [8], BitFit [9], and sparse fine-tuning methods [10–13] reduce parameters further without significant performance loss, collectively highlighting model overparameterization. Our work explores the extreme lower bound of parameter efficiency, demonstrating substantial gains from training as few as 10-100 randomly selected parameters, far fewer than existing approaches.

Scaling Laws and Information-Theoretic Bounds. Recent work on scaling laws characterizes performance improvement with increased compute, data, or total model parameters [14–16]. Complementing this, our “elicitation frontier” explicitly focuses on the minimal parameter budget required for task adaptation. We leverage Rissanen’s Minimum Description Length (MDL) [2] principle and related work [17] that directly relates the utility of a capability to reduction of the MDL to explain the empirically observed scaling behavior.

Lottery Tickets and Random-Parameter Hypotheses. The Lottery Ticket Hypothesis [18] demonstrates that sparse subnets (“winning tickets”) can match dense performance but requires expensive

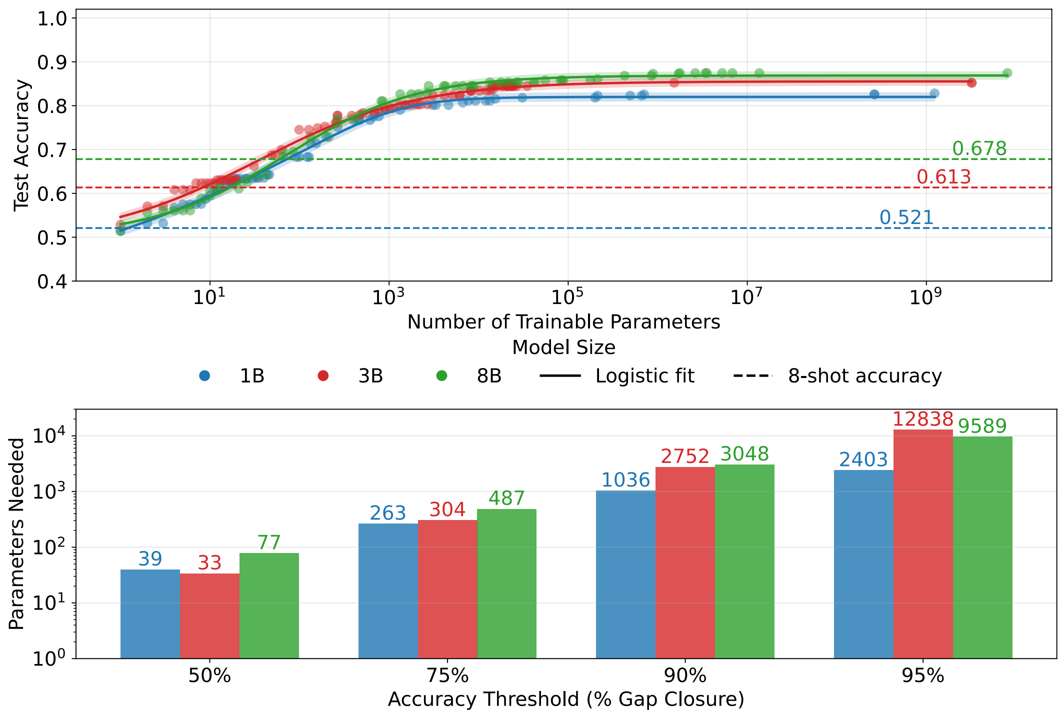


Figure 1: **Fine-tuning only 10-100 randomly selected parameters recovers 50% of performance gap between zero-shot and fully fine-tuned models.** *(Top)* Logistic fits (solid lines) to Pareto frontier points (circles) with bootstrapped 95% confidence intervals (shaded regions) for Llama models of sizes 1B, 3B, and 8B parameters fine-tuned on GSM-8K-CoT-Choice. Dashed lines indicate the 97.5 percentile eight-shot accuracy for each model. *(Bottom)* Minimum number of trained parameters at which various levels of gap closure (50%, 75%, 90%, and 95%) are met or exceeded for each model size (1B, 3B, and 8B parameters).

search. Our findings show random-parameter selection achieves nearly identical Pareto frontiers without specialized search, suggesting that adaptation information is both compact and distributed throughout the model [19].

3 Methodology

3.1 Base Models & PEFT Setup

In our experiments, we fine-tune a randomly chosen subset of a model's parameters to convergence on different tasks using their full available training sets, and we measure how performance on held-out data varies with the minimum number of parameters needed to specify the updated weights of the fine-tuned model.

Models and Baselines. We fine-tune widely used language models from the Llama family (Llama-3.2-1B/3B, Llama-3.1-8B) [20]. The models used in our experiments are all pretrained base models, unless otherwise noted.

We compare with full fine-tuning (Full FT) updating all learnable parameters in all layers, which defines each model's upper performance ceiling, zero-shot accuracy (performance floor) and eight-shot accuracy (prompt-based elicitation) for each model, as well as standard LoRA. In binary classification tasks (BoolQ, GSM-8K-CoT-Choice), we also compare performance with that of a trained, task-specific classifier head on top of the unembedding layer of the base model, as well as a linear probe trained directly on the representations of the middle two layers of the model. Additional details can be found in Appendix A.

Parameter Selection and Fine-Tuning. In experiments that train random subsets of parameters within the LoRA modules, learnable subsets of the LoRA adapter matrices are trained while the remaining adapter weights are held at their initialized values. We select these trainable parameters using uniform random sampling across all LoRA modules in all layers, with each parameter having equal probability of selection regardless of its position in the network. This selection is implemented by applying a random binary mask over gradients during the backward pass, preserving gradient updates only for the selected parameters.

In all experiments, we train until convergence, regardless of the number of epochs required. To verify that only the intended number of learnable parameters has been updated during training, we compare the weights of the LoRA modules at initialization to the weights following fine-tuning and ensure that only the unmasked parameters have been updated. See Appendix C for additional details about fine-tuning procedure details and hyperparameter selection.

3.2 Datasets

We fine-tune and evaluate on five tasks: four multiple-choice benchmarks and one generative task. The multiple-choice commonsense reasoning benchmarks and datasets span reading comprehension, arithmetic and mathematics, and science: GSM-8K-CoT-Choice (see Section D.1 for dataset details), ARC-Easy, ARC-Challenge [21], and BoolQ [22]. For natural language generation tasks, we fine-tune and evaluate on the Alpaca dataset for instruction-tuning [23]. This dataset contains a diverse set of instructions and corresponding responses, allowing us to assess how effectively our approach generalizes to open-ended generation tasks beyond multiple-choice classification.

4 Empirical Results

Here, we present our core empirical findings that complement subsequent information-theoretic analysis in Section 5. First, in Section 4.1 we extract and fit single-task Pareto frontiers on the arithmetic reasoning GSM-8K-CoT-Choice dataset, demonstrating the low parameter budgets required for substantial gap closure. In Section 4.2 we show that this logistic-in-log-parameters relationship generalizes across additional question-answering tasks. Finally, Section 4.3 examines the robustness of these frontiers to seed and hyperparameter variation.

4.1 Fine-tuning fewer than 50 *randomly-chosen* parameters achieves over 50% gap closure

Pareto frontier and logistic fit. Following the procedure of Appendix E, we select a sequence of trainable parameter budgets spanning from a single parameter to the full set of LoRA adapter weights (ranks 1, 2, 4, 8, and 16), in addition to all parameters in the full model. We define the empirical frontier as the maximum accuracy achieved across all seeds for each parameter budget.

Figure 1 (top) plots the resulting frontier points for GSM-8K-CoT-Choice on a log-scale x–axis, along with the best-fit generalized logistic curve and corresponding bootstrapped 95% confidence intervals. The data follow a logistic-in-log-parameters relationship, with a steep rise in performance as parameter count increases from 1 to approximately 100, followed by a saturation of gains approaching the accuracy ceiling of the fully fine-tuned model's performance.

Gap closure thresholds. To quantify how many parameters are needed for common performance targets, we compute the budgets at which the given performance gap recovery thresholds are met, with the extent of performance gap defined as the difference between the full fine-tuned model's performance (or maximum of the logistic fit, whichever is higher) and its zero-shot accuracy.

Figure 1 (bottom) reports these values for the three model sizes. Table 1 reports these values for the 50% and 90% thresholds across all models and multiple-choice reasoning datasets, comparing them with the performance of zero-shot evaluation on the pretrained-only base model and that of the full fine-tuned model with all ~10^9 parameters trained.

Only ~30 parameters are required to recover 50% of the zero-shot to full-fine-tune gap on GSM-8K-CoT-Choice, meaning that adjusting exclusively a few dozen randomly selected dimensions across all layers of the model can achieve half of the performance gain of full fine-tuning.

Table 1: **Number of trainable parameters required to close X% of the performance (accuracy) gap on multiple-choice reasoning datasets** (GSM-8K-CoT-Choice, ARC-Easy, ARC-Challenge). The extent of the full performance gap is defined as the difference between the performance of the fine-tuned model with all parameters trained (Full FT) and the zero-shot accuracy of the pretrained base model. #Params denotes the minimum number of trained parameters at which the model's performance matches or exceeds 50% or 90% of the gap above the zero-shot accuracy. %Params denotes the percentage of trained parameters at each threshold relative to the size of the pretrained base model backbone. Results for natural language generation can be found in Section G.1.

Dataset	**Model**	Baselines (%Acc.)		50% Gap Closure		90% Gap Closure	
		Zero-shot	**Full FT**	**#Params**	**%Params**	**#Params**	**%Params**
GSM-8K-CoT-Choice	Llama-3.2-1B	47.56	87.46	39	2.67×10^{-6}	1036	8.38×10^{-5}
	Llama-3.2-3B	48.36	85.21	33	1.21×10^{-6}	2752	8.56×10^{-5}
	Llama-3.1-8B	51.27	87.46	77	9.58×10^{-7}	2403	3.80×10^{-5}
ARC-Easy	Llama-3.2-1B	24.81	76.45	216	1.74×10^{-5}	4144	3.35×10^{-4}
	Llama-3.2-3B	74.08	87.06	774	2.41×10^{-5}	159k	4.95×10^{-3}
	Llama-3.1-8B	80.46	91.27	99	1.28×10^{-6}	318k	3.96×10^{-1}
ARC-Challenge	Llama-3.2-1B	24.21	53.78	3023	2.44×10^{-4}	154k	1.24×10^{-2}
	Llama-3.2-3B	53.81	75.01	236	7.34×10^{-6}	3077	9.58×10^{-5}
	Llama-3.1-8B	61.11	82.66	98	1.22×10^{-7}	415	5.17×10^{-6}

4.2 Comparison of logistic scaling across tasks

The logistic scaling relationship we observe on GSM-8K-CoT-Choice is similarly exhibited in the additional multiple-choice tasks. Figure 2 shows the performance versus parameter count comparing GSM-8K-CoT-Choice with ARC-Easy and ARC-Challenge, revealing consistent logistic patterns across multiple-choice reasoning tasks in different domains with distinct task formats (Appendix D).

While the absolute performance levels and the exact number of random parameters required for specific gap closure thresholds vary across tasks and models sizes, the characteristic S-curve shape in log-parameter space remains consistent. To verify that our observed logistic relationship is not an artifact of the specific functional form chosen, we evaluated several alternative scaling relationships (power law, saturating exponential, and piecewise linear with power law). Quantitative model comparison using Bayesian and Akaike information criteria consistently favored the logistic function across all datasets and model sizes (see Appendix F for detailed comparisons). The consistency of these fits suggests that the observed scaling behavior reflects an underlying, common relationship between log-parameters and performance, rather than being an artifact of our analysis approach.

The grade-school level science datasets ARC-Easy and ARC-Challenge show a similar overall pattern, albeit with higher parameter requirements than GSM-8K-CoT-Choice, likely due to the more specialized scientific knowledge they assess. In particular, fine-tuning on ARC-Challenge, composed entirely of questions answered incorrectly by both a retrieval-based algorithm and a word co-occurrence algorithm, requires significantly more parameters for the smallest model (Llama-3.2-1B) than on other tasks as well as compared to larger models (Llama-3.2-3B and Llama-3.1-8B), potentially indicating capability acquisition rather than elicitation due to the possible absence of the capability in the 1B model following pretraining.

As with GSM-8K-CoT-Choice, we observe that larger models also require only few-parameter adjustments to achieve substantial gap closure (Table 1), supporting the hypothesis that larger models already possess most of the requisite knowledge and require minimal tuning to effectively demonstrate it. On all multiple-choice datasets, including both ARC datasets, Llama-3.1-8B achieves 50% performance gap recovery with fewer than 100 trainable parameters.

The Alpaca instruction-tuning dataset, used for adapting pretrained models to support conversational interactions with users, demonstrates that our findings extend beyond multiple-choice classification to open-ended natural language generation settings, where measuring models' capabilities is inherently more challenging (Section G.1). Even in this more complex setting, we observe that training a small

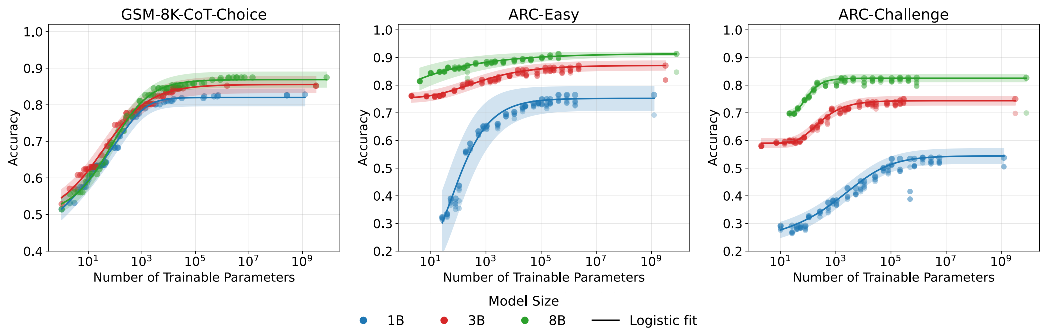


Figure 2: **Logistic scaling patterns emerge across multiple tasks and datasets for all model sizes.** Each subplot shows model performance (y-axis) for all random seeds on a different task (from left to right: GSM-8K, ARC-Easy, ARC-Challenge) against the number of trainable parameters (x-axis, log scale) for three model sizes (1B, 3B, 8B). Solid lines indicate logistic fits to Pareto frontier points, with shaded regions denoting bootstrapped 95% confidence intervals. Consistent scaling behavior as a function of trainable parameters suggests that general features broadly characterize elicitation patterns and asymptotic model performance across task types, dataset difficulty, and model size.

fraction of parameters (relative to model size) achieves most of the performance benefits of full fine-tuning.

4.3 Stability of Pareto frontier and robustness of logistic scaling across random initializations

Figure 2 shows the distribution of highest-performing models at every parameter budget for every random seed, rather than just the nominal Pareto frontier points. Each point indicates a model fine-tuned with a unique random seed; opaque points correspond to the Pareto frontier, with solid line logistic fits to the frontier-defining points. We observe that the Pareto frontier is insensitive to seed and that the highest performing runs for all seeds overwhelmingly lie within the 95% confidence interval of the fit to the actual frontier.

While there is naturally some variation across random initializations of the LoRA matrices, as well as which modules and layers are adapted, the overall pattern of rapid performance improvement with minimal parameters is consistent. The logistic trend also remains visible across seeds, even with suboptimal hyperparameters (Appendix H), and we observe similar, but shifted, logistic fits independent of optimization effort or method.

For all models and tasks, we observe clear diminishing returns as accuracy approaches the full fine-tune performance level. This saturation effect is consistent with the following information-theoretic interpretation that there exists a minimum amount of information required to for a capability to fully manifest, beyond which additional parameters provide redundant capacity.

5 Minimum Description Length

Rissanen's Minimum Description Length (MDL) principle [2] formalizes Occam's razor as the notion that the best model of a dataset is one that provides the simplest explanation, or shortest description, for it. MDL frames algorithmic learning as optimal data compression, in which models that can reconstruct task data from shorter descriptions are said to have learned more about the task.

Perez et al. [17] find that a capability f is useful to a model *if and only if* it decreases the length of the model's shortest description of the dataset. They do so using a more computationally-tractable variant of MDL, Rissanen Data Analysis (RDA) and demonstrate that genuine capabilities always shrink MDL, whereas information accessible to a model that is uncorrelated with the data labels does not. For example, being able to perform long-form reasoning before answering a question, or breaking down complex tasks into simpler subparts may allow the model to substitute reasoning for memorization and shrink the MDL.

In our setting, we presume that a capability may be latent in the pretrained (base) model M_0 and try to elicit that capability on a dataset designed to surface it. A small adapter $\Delta\theta_k$ with only k trainable weights serves as a pointer to that capability. If a short message (containing the adapter weights $\Delta\theta_k$) lets the model compress the subsequent label stream by hundreds of bits, the original base model already likely contained a highly structured representation of the task. Conversely, if the MDL shrinks slowly or not at all until k becomes large, the capability was likely absent and must be learned from scratch rather than elicited. MDL compression, given by the difference in description lengths between the pretrained base model and the fine-tuned model, $\Delta_k = \bar{\mathcal{L}}_p^{(0)} - \bar{\mathcal{L}}^{(k)}$, directly measures how helpful the latent capability is to the model when k parameters are available to specify how to access it, quantifying the intrinsic value of the capability to the task at a specific level of elicitation.

Table 2: **Un(der)elicited versus elicited models in MDL terms.**

Elicitation regime	MDL (#Bits to encode labels)	Capability presence
Un(der)elicited base model, M_0	$\mathcal{L}_0$ bits	Implicit
Elicited fine-tuned model, M_k	$\mathcal{L}_k \ll \mathcal{L}_0$ bits after adapting k weights	Explicit; manifests in predictions
Model with capability absent, $M_\times$	$\mathcal{L}_k \sim \mathcal{L}_0$ bits for much larger k	Absent

We implement RDA to estimate upper bound MDL using prequential coding, encoding each example sequentially using the model trained only on previously observed examples, following the procedure of Perez et al. [17]. Practically, we initialize the pretrained model, present each training example exactly once, perform a gradient step immediately after an example (or batch) has been seen, and accumulate the per-example (or per-batch) cross-entropy loss across all examples in the dataset. Full details of this computation and its construction for our particular fine-tuning setting can be found in Appendix I.

5.1 Empirical MDL Scaling and Logistic Behavior

For most tasks and models, MDL compression closely tracks improvements in accuracy, monotonically increasing with increasing number of randomly selected trainable parameters and eventually saturating at a maximal ceiling (Figure 3, top right, black). Both curves follow logistic forms in log-parameter space.

For Llama-3.1-8B fine-tuned on ARC-Challenge, the accuracy and MDL derivatives nearly coincide and share similar qualitative features, such as near-Gaussian behavior. Figure 3 (top right) demonstrates the close tracking of accuracy with MDL compression (in bits). In contrast, for the same dataset, the accuracy and MDL curves for Llama-3.2-1B, along with their respective derivatives, demonstrate significant discrepancies between both each other and the behavior observed in Llama-3.1-8B.

To quantify information encoding efficiency, we analyze the derivative of the logistic MDL curve versus parameters (Figure 3, bottom). The peak in the MDL derivative indicates the parameter budget at which additional parameters maximally improve dataset compression. Comparing this to accuracy derivatives reveals subtle differences between information-theoretic encoding efficiency and downstream task performance efficiency. Whereas the two peaks coincide for Llama-3.1-8B, the significant gap between the accuracy and MDL peak efficiencies for Llama-3.2-1B (Figure 3, bottom left), suggests that the 1B model requires many more parameters to develop an efficient description of the dataset; configurations with fewer trainable parameters appear to be inadequate for the model to significantly employ a useful latent capability, if one exists, to improve its representation of the task. This discrepancy suggests that while additional parameters offer large marginal gains in both performance and dataset compression for the 8B model as a latent, task-relevant capability is made more explicit (Table 2), the 1B model initially cannot make use of such a capability to produce a more compact description of the dataset and requires a larger number of trainable parameters.

6 Discussion

In Section 4 and Section 5, we presented two complementary views of elicitation: an empirical view, where performance rises logistically with the number of trained parameters, and an information-theoretic view, where the Minimum Description Length (MDL) of the label stream

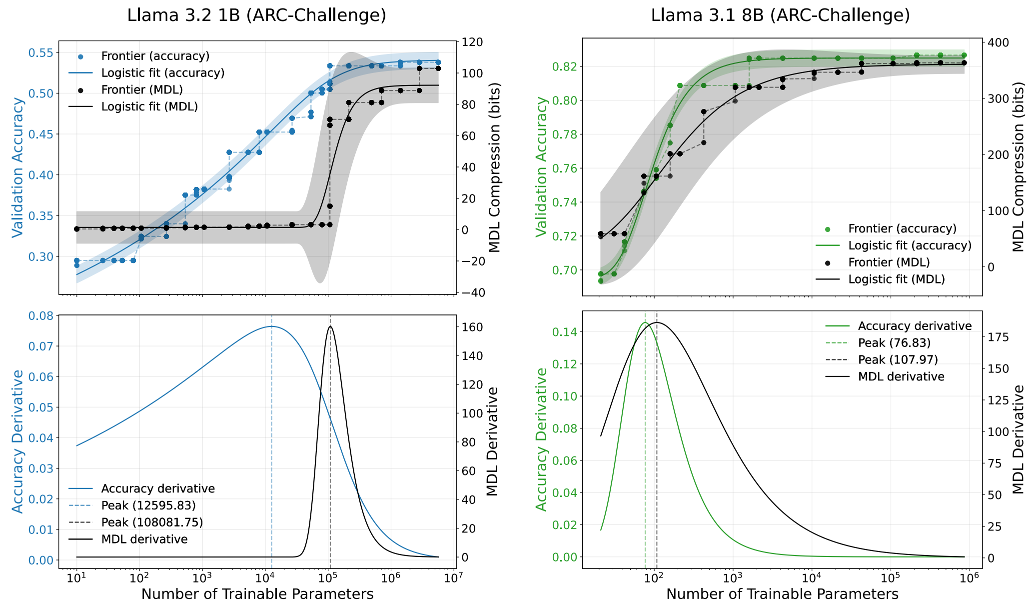


Figure 3: **Accuracy and MDL compression track each other when eliciting latent capabilities but deviate when the required capability is initially absent in the model.** *(Top panels)* Accuracy (colored, left y-axis) and MDL compression (black, right y-axis) plotted against parameter count (x-axis, log scale) for Llama-3.2-1B (left) and Llama-3.1-8B (right) on ARC-Challenge. For 8B, as parameter count increases, MDL compression and accuracy both rise following logistic curves in log-parameter space. In contrast, MDL compression in 1B remains flat up to ~10^5 parameters, at which point it discontinuously jumps, though accuracy rises smoothly. Shaded regions indicate bootstrapped 95% confidence intervals on logistic fits. *(Bottom panels)* Derivatives of the accuracy (colored) and MDL compression (black) curves, showing peak rates of change that correspond to highest per-parameter compression and performance efficiency (vertical dashed lines). Both peaks occur at similar parameter counts for Llama-3.1-8B (right) but are separated by approximately an order of magnitude for Llama-3.2-1B (left), indicating significant differences in how efficiently the models are each able to explicitly use the necessary capability to generate simpler models of the dataset, relative to the number of parameters trained.

falls as soon as a latent capability is unlocked. Here, we synthesise these perspectives and discuss their implications.

6.1 Linking performance frontiers and MDL compression

Figure 1 and Table 1 showed that training only ~10-10^5 randomly chosen parameters recovers 50–95% of the zero-shot to full-fine-tune gap. Section 5 then established that the same parameter budgets yield hundreds of bits of MDL reduction. Taken together, these results indicate that the logistic frontiers of Section 4 can be viewed analogously to compression curves: each additional parameter provides the model with an incremental number of bits it can spend on describing the task. The knee of the logistic curve therefore pinpoints an information threshold beyond which adding capacity yields diminishing returns. Models that already contain the requisite capability (e.g. Llama-3.1-8B on ARC-Challenge) reach this threshold quickly, whereas models that may need to learn the capability (Llama-3.2-1B on the same task) require orders of magnitude more parameters before significant reductions in MDL occur.

6.2 Practical and safety implications

1. **Emergent behaviors.** Because only a handful of adapter parameters can double a model's accuracy, small code snippets may be sufficient to unlock sophisticated behavior. This increases the attack surface for adversaries seeking to subvert deployed systems.

2. **Evaluation gaps.** Benchmarks relying on zero-shot or in-context performance may underestimate a model's true capabilities by a large margin. MDL may provide a useful diagnostic: if a small adapter suffices to significantly reduce a model's MDL, evaluators should treat the corresponding skill as present, even if accuracy is low in the frozen state or in prompt-based elicitation settings.
3. **Information bottlenecks.** The steep drops in MDL observed at low parameter counts suggest that pretraining stores task-relevant information in a form that is addressable but not automatically retrieved. Elicitation removes this bottleneck.
4. **Distributed representations.** The success of random parameter selection suggests that knowledge is spread throughout the network rather than localized in specific layers.
5. **Logistic-in-log-parameters scaling.** Consistent scaling patterns across tasks and model sizes hints at a possible common property of elicitation that may offer use as a means of predicting future elicitation thresholds for capabilities of interest.

6.3 Limitations

- **Task diversity:** While we have evaluated our approach across several multiple-choice reasoning and instruction-following tasks spanning multiple subjects, our findings may not generalize to all possible task types or domains. Some capabilities or specialized knowledge might require more substantial parameter adjustments than others, as is potentially exhibited on ARC-Challenge by Llama-3.2-1B.
- **Model architectures:** Our experiments focused exclusively on the Llama family of models. Different model families or architectures (e.g., MoE models, reasoning models) might exhibit different elicitation scaling relations or require different parameter budgets.
- **Theoretical approximations:** Our information-theoretic analysis relies on approximations such as first-epoch loss as a proxy for MDL. More precise formulations might reveal additional nuances in the elicitation process.
- **Random selection:** While our random parameter selection approach yields surprisingly strong results, it may not be optimal. Structured approaches to parameter selection or low-probability "winning ticket" parameter selections or initializations could potentially improve efficiency further.
- **Elicitation methods:** There are potentially methods of eliciting capabilities that vastly outperform what current methods are able to achieve, making it difficult or impossible to provide a hard minimum upper bound on the number of parameters needed for elicitation.

7 Conclusion

We present a novel framework for quantifying the elicitation of latent capabilities in language models based on the minimum number of trainable parameters required to achieve various performance levels on downstream tasks. We propose an "elicitation frontier," a curve that maps the number of trained parameters to both accuracy and description-length improvements. Across five tasks and three model sizes, we find that a few dozen to a few thousand randomly selected parameters suffice to recover most of the performance gap between zero-shot and fully fine-tuned models. By framing fine-tuning as an encoding game and borrowing Rissanen Data Analysis to estimate MDL, we translate these empirical frontiers into bits of latent capability. The correspondence between logistic accuracy gains and MDL compression suggests that latent skills manifest precisely when the information budget available to a model crosses a critical threshold.

Our elicitation frontier framework offers a complementary perspective to traditional scaling laws, focusing on the minimal information required to unlock capabilities rather than the maximal capacity of models. The consistent logistic scaling relationship we observe across tasks and model sizes suggests that fundamental, shared principles may govern the ways in which capabilities scale with parameter adjustments. Our study complements classic compute- and data-centric scaling laws with an information-centric perspective: how much must be said to a model before it shows what it already knows? Understanding this budget is crucial for capability forecasting, responsible deployment, and the design of safe, reliable, and predictable AI systems.

References

[1] Teun van der Weij, Felix Hofstätter, Ollie Jaffe, Samuel F Brown, and Francis Rhys Ward. Ai sandbagging: Language models can strategically underperform on evaluations. *arXiv preprint arXiv:2406.07358*, 2024.

[2] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[3] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. *CoRR*, abs/1902.00751, 2019. URL http://arxiv.org/abs/1902.00751.

[4] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *CoRR*, abs/2101.00190, 2021. URL https://arxiv.org/abs/2101.00190.

[5] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021. URL https://arxiv.org/abs/2106.09685.

[6] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *CoRR*, abs/2012.13255, 2020. URL https://arxiv.org/abs/2012.13255.

[7] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024.

[8] Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37:121038–121072, 2024.

[9] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*, 2021.

[10] Jing Xu and Jingzhao Zhang. Random masking finds winning tickets for parameter efficient fine-tuning, 2024. URL https://arxiv.org/abs/2405.02596.

[11] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021. URL https://arxiv.org/abs/2104.08691.

[12] Yusheng Su, Chi-Min Chan, Jiali Cheng, Yujia Qin, Yankai Lin, Shengding Hu, Zonghan Yang, Ning Ding, Xingzhi Sun, Guotong Xie, Zhiyuan Liu, and Maosong Sun. Exploring the impact of model scaling on parameter-efficient tuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, page 15062–15078. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.931. URL http://dx.doi.org/10.18653/v1/2023.emnlp-main.931.

[13] Max Ploner and Alan Akbik. Parameter-efficient fine-tuning: Is there an optimal subset of parameters to tune? In Yvette Graham and Matthew Purver, editors, *Findings of the Association for Computational Linguistics: EACL 2024*, pages 1743–1759, St. Julian's, Malta, March 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-eacl.122/.

[14] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

[15] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[16] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[17] Ethan Perez, Douwe Kiela, and Kyunghyun Cho. Rissanen data analysis: Examining dataset characteristics via description length. In *International Conference on Machine Learning*, pages 8500–8513. PMLR, 2021.

[18] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[19] Mengxia Yu, De Wang, Qi Shan, Colorado Reed, and Alvin Wan. The super weight in large language models, 2024. URL https://arxiv.org/abs/2411.07191.

[20] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[21] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

[22] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*, 2019.

[23] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

[24] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[25] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18 (185):1–52, 2018.

[26] Qiong Gao, Ming Li, and Paul Vitányi. Applying mdl to learn best model granularity. *Artificial Intelligence*, 121(1-2):1–29, 2000.

[27] A. Barron, J. Rissanen, and Bin Yu. The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2760, 1998. ISSN 0018-9448. doi: 10.1109/18.720554. URL http://dx.doi.org/10.1109/18.720554.

[28] A. P. Dawid. Present position and potential developments: Some personal views: Statistical theory: The prequential approach. *Journal of the Royal Statistical Society. Series A (General)*, 147(2):278, 1984. ISSN 0035-9238. doi: 10.2307/2981683. URL http://dx.doi.org/10.2307/2981683.

[29] A P Dawid. *Prequential Analysis, Stochastic Complexity and Bayesian Inference*, page 109–126. Oxford University PressOxford, August 1992. ISBN 9781383023756. doi: 10.1093/oso/9780198522669.003.0007. URL http://dx.doi.org/10.1093/oso/9780198522669.003.0007.

A Baselines

For all tasks, we use three baselines for comparison (zero-shot performance, eight-shot performance, and fine-tuning all parameters of the base model). We define each model's performance floor as its zero-shot accuracy (multiple-choice datasets) or zero-shot win rate (Alpaca). We take each model's highest performance (accuracy or win rate) following full fine-tuning (updating all model weights) as its nominal ceiling performance.

Full fine-tuning. For all models, we fine-tune the entire model until convergence on each dataset. We define convergence as the point where validation accuracy exhibits no improvement for 20 consecutive epochs. Specifically, if $|\mathrm{Acc}(t) - \max(\mathrm{Acc}(t-20:t))| < \varepsilon$ where $\varepsilon = 10^{-4}$, training terminates. This criterion ensures stable convergence while avoiding premature stopping.

Zero-shot evaluation. For each model, we perform zero-shot evaluation on the validation and test sets. Prompts used for zero-shot evaluation are the same as those used for all fine-tuning experiments. Prompt formats for each dataset can be found in Appendix D.

Multi-shot evaluation. We perform multi-shot evaluation similarly, using the same prompt format as used for zero-shot evaluation and prepending randomly selected examples (with properly formatted correct responses for each) from the corresponding train set. The number of shots is equal to the number of prepended train set examples with answers. We use the same formatting for each example, prepending the same train set examples each time, with the final example being the validation or test set example that we evaluate the model's response on, in which the answer field is left blank.

We tested variations with 1, 2, 4, 8, 16, and 32 shots prepended for each model and dataset, finding that 8 shots resulted in the highest performance across the board. We average performance across $n = 30$ trials for each number of shots X to obtain the overall X-shot performance. Every trial uses a different random seed that is used only to select the specific train set examples that are prepended; the same set of random seeds is used for all models, datasets, and X-shot configurations. We ensure that the prepended examples contain an approximately even split of each answer class to reduce potential sources of bias in the model outputs. Due to variation in performance across seeds, we use the 97.5 percentile (2σ above the mean) value on the distribution of scores as an approximate upper bound baseline for the model's X-shot performance.

A.1 Additional binary classification baselines

For binary classification datasets GSM-8K-CoT-Choice and BoolQ, we compare the performance of two additional baselines: (i) training a task-specific classifier head attached to the unembedding layer of the model, and (ii) training a linear probe on the representations of the middle two layers of each model. Figure 4 compares these baselines for Llama-3.2-1B on GSM-8K-CoT-Choice.

Classifier Head. We initialize each model with a classifier head that enforces the model's outputs to be one of the two label classes. The classifier head contains twice the number of parameters of the model's hidden dimension. We then freeze the model backbone and train only the weights in the classifier head when fine-tuning on each dataset, taking the highest final evaluation performance at convergence across seeds ($s = 5$) as the baseline.

Linear Probe. We initialize each model with all parameters frozen and train a linear probe out the representations of the middle two layers of each model.

B Generative Experiments

Discussion Across the Alpaca instruction-tuning runs, win-rate versus trainable-parameter count similarly increases quickly with few parameters. For Llama-3.2-1B, performance jumps from 0.80 to 0.88 wins as the budget grows from 10^3 to 10^5 updated weights. The 3B model follows a similar pattern, shifted one decade to the right. Note that overall, the win rate is higher for 1B as it is compared to a lower-capacity base model.

Dataset and splits. For the open-ended generation experiments we adopt the Stanford Alpaca instruction-following corpus. The raw 52k examples are shuffled once with a fixed random seed and partitioned 90/10 into training (234,006 instructions) and held-out evaluation (26,006 instructions). No additional filtering or augmentation is applied. All results in the main text use the full training split; the "tiny" ablations described below operate on the same data but restrict the number of trainable parameters, not the dataset size.

Prompt construction and tokenization. Each instance is formatted with the standard Alpaca template where the input block is omitted when empty. The expected answer is appended after the prompt. The entire

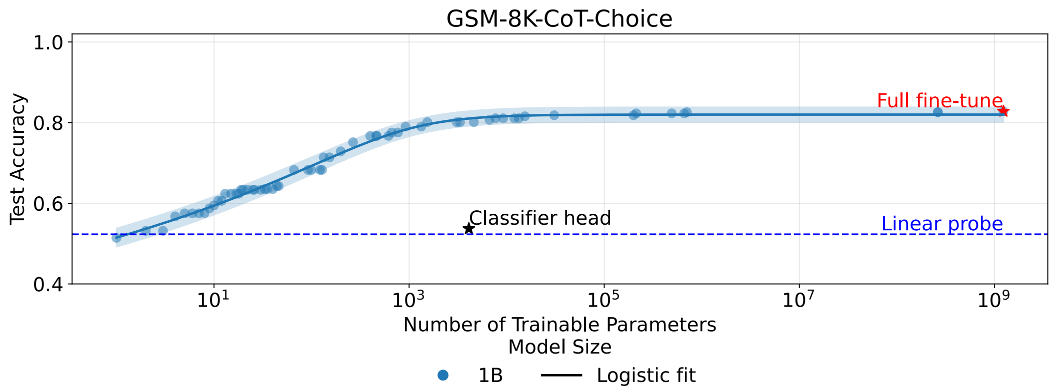


Figure 4: Llama-3.2-1B fine-tuned on GSM-8K-CoT-Choice with full fine-tune (red star), classifier head (black star), and linear probe (blue dashed line) baselines.

sequence is truncated to 256 tokens. During training we follow common instruction-tuning practice and compute loss only on the response: all prompt tokens are replaced with the ignore-index so gradients flow exclusively through the generated portion. The end-of-sequence token is reused as the pad token.

Base models and adaptation strategy. We fine-tune two pretrained Llama-3.2 models—1B and 3B parameters—using low-rank adaptation. Unless otherwise noted, LoRA modules (ranks 1, 2, 4, 8, 16, 256, 512) are inserted on every projection matrix in every transformer block.

To study how few parameters suffice for high-quality generation we treat the weights inside the LoRA matrices as an unordered pool and select an absolute number of them to update (1 to full rank). Selection is uniform at random over all LoRA weights; the mask is sampled once at the start of each run and held fixed for the duration of training. In the "tiny" regime (<10k trainable parameters) the original language-model head remains frozen to isolate the effect of the tiny adaptation; at larger budgets the head is fine-tuned together with the adapters.

Optimization and runtime. Every model is trained for exactly two passes over the 234k-example training split. Mixed-precision (bfloat16) training with FlashAttention-2 is employed throughout, and gradient check-pointing is enabled except in the smallest runs. Each experiment fits on a single NVIDIA H100 GPU; no multi-GPU or distributed training was employed.

Evaluation protocol. After fine-tuning we generate completions for the 805-example Alpaca evaluation subset using greedy decoding (temperature 0, max 256 tokens). Outputs are scored by an off-the-shelf preference model that compares each fine-tuned model's answer against the corresponding answer from the un-adapted base model of the same size. We report the win rate of the fine-tuned model over the 805 comparisons; ties and losses are counted separately but not shown in the main tables.

Summary of reported configurations. Model sizes: Llama 3.2 1B, Llama 3.2 3B

LoRA ranks: 1, 2, 4, 8, 16, 256, 512

Trainable-parameter budgets: exact counts spanning from just a few parameters to nearly full finetuning.

Epochs: 2 (all runs)

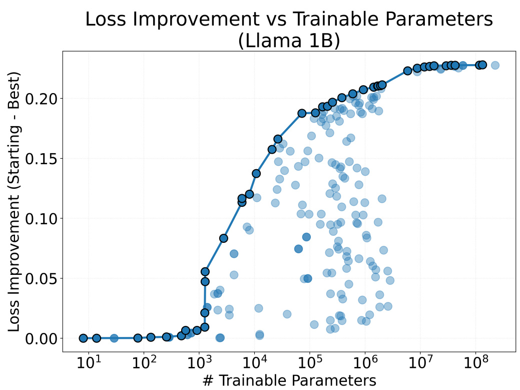
Sequence length: 256 tokens

Hardware: single NVIDIA H100 80 GB GPU per run

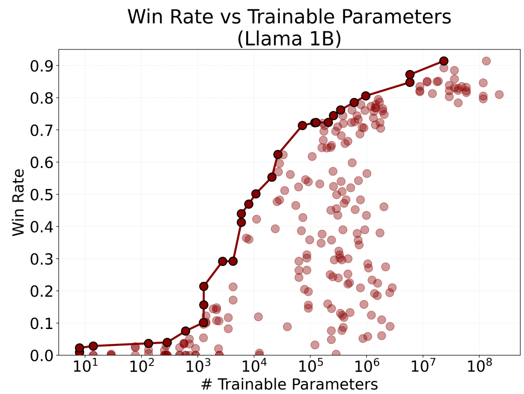
C Fine-tuning details

C.1 Target Modules

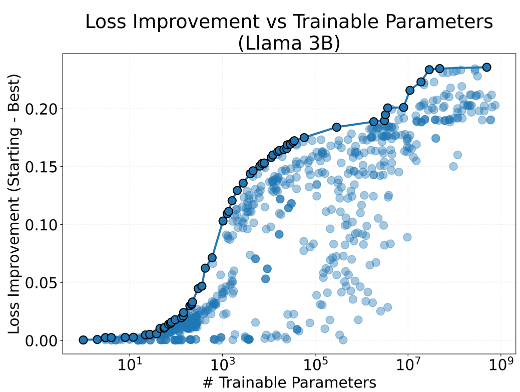
We investigate variations in which we adapt all weight matrices within the transformer block, Q, V or Q, V, K, O matrices in the self-attention layers, G, U, D matrices in the feed-forward network/MLP layers, K, Q, V, U, D matrices across the transformer block, and randomly selected subsets of weight matrices. We observe similar logistic scaling behavior of performance versus number of learnable parameters across all configurations of adapted weight matrices.



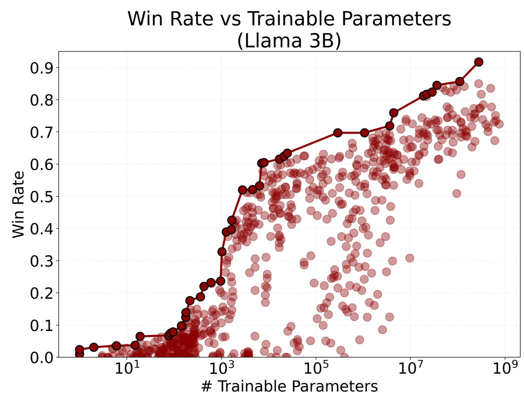
(a) 1B Pareto frontier: Loss improvement



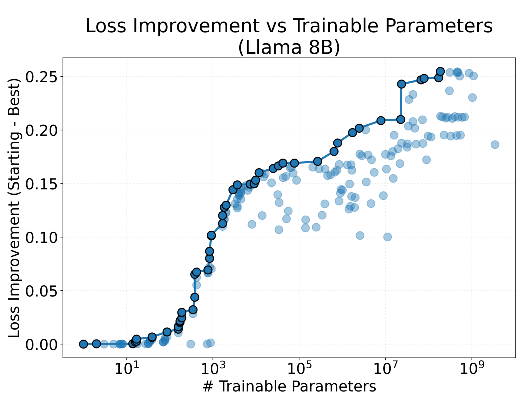
(b) 1B Pareto frontier: Win rate



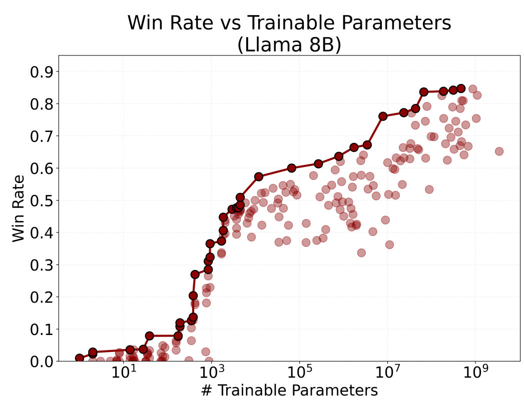
(c) 3B Pareto frontier: Loss improvement



(d) 3B Pareto frontier: Win rate



(e) 8B Pareto frontier: Loss improvement



(f) 8B Pareto frontier: Win rate

Figure 5: Comparison of loss improvement and win rate for Llama models fine-tuned on Alpaca and evaluated on AlpacaEval (1B, 3B, and 8B). Loss improvement is computed as each fine-tuned model's (blue points) improvement in the cross-entropy loss for autoregressive generation compared to its corresponding base model. Win rate denotes the proportion of responses generated a fine-tuned model (red points) that are preferred by the reward model over responses generated by the best full fine-tuned model. In each graph, single points denote the result of a single fine-tuned model with the specified number of trainable parameters. The Pareto frontier in each graph is denoted by the opaque points with black outline connected by a solid line.

When we refer to "trainable parameters," we exclusively mean scalar values within LoRA adapter matrices. For rank-r adaptation:

- Matrix A: $\mathbb{R}^{r\times d_{in}}$ (Kaiming uniform initialization)
- Matrix B: $\mathbb{R}^{d_{out}\times r}$ (zero initialization)
- Total parameters per module: $r \times (d_{in} + d_{out})$

C.2 Seed and hyperparameter variation

For each parameter budget, we run up to 5 random seeds with up to 50 Bayesian-Hyperband trials each over learning rate, batch size, and weight decay. In all experiments, we train until convergence (defined as at least 20 epochs with no improvement in validation accuracy), regardless of the number of epochs required.

Search spaces. Parameter search ranges are calibrated based on the number of trainable parameters:

- Learning rate: $[10^{-6}, 1]$ (log-uniform spacing)
- Batch size: $\{4, 8, 16, 32, 64, 128, 256\}$ (discrete)
- Weight decay: $[0, 0.1]$ (uniform sampling)

C.3 Multiple-choice scoring

For multiple-choice tasks, we aggregate probabilities across formatting variations:

$$P(\text{label}) = \sum_i P(\text{token}_i), \tag{1}$$

where $\text{token}_i \in$ {"A", " A", "a", " a"} for label A (and similarly for all other labels). This approach ensures robustness to minor formatting variations in the model's output while maintaining evaluation consistency.

To reduce the dependence of the results on the model's ability to format its responses properly, we calculate loss and accuracy directly using the logits for tokens corresponding to each label class. We take the softmax of the model's output vector for the final token position to obtain probabilities on the outputs for each token in the model's vocabulary, and for each label class, we sum the probabilities of differently formatted, single-token variations of each label (e.g., "A": ("A", " A", "a", " a") for label class A). We use the same set of variations for all label classes (e.g., "Correct": ("Correct", " Correct", "correct", " correct"), "Incorrect": ("Incorrect", " Incorrect", "incorrect", " incorrect") with sets of labels for each class incorporating the same variations on spacing and capitalization).

C.4 Bayesian optimization directly over LoRA adapter weights (ultra-low parameter regimes)

For experiments in which fewer than 20 parameters are trained, training until convergence can require more than 1000 training epochs, and training dynamics can become unpredictable with discontinuities observed in the loss. The combination of these factors makes it difficult to identify when training has converged or to explore sufficient hyperparameter configurations to effectively resolve the true elicitation Pareto frontier.

To circumvent this issue, in addition to training with gradient descent, in the regime of fewer than ~20-25 trainable parameters, we also perform Bayesian optimization with a Gaussian process surrogate model and expected improvement acquisition function (GPyTorch with default Matérn kernel) directly on the learnable adapter weights within the LoRA matrices; parameter selection is performed using the same random sampling procedure as in the gradient-descent-based fine-tuning setting. This approach mitigates optimization instabilities at extreme sparsity levels.

For the Gaussian process, parameter bounds are empirically calibrated through preliminary gradient descent experiments:

- $k \leq 2 : [-10, 10], [-5, 5]$
- $3 \leq k \leq 10 : [-5, 5], [-3, 3]$
- $11 \leq k \leq 25 : [-3, 3], [-1, 1],$

where k is the number of trainable parameters. The Gaussian process optimizer directly searches this k-dimensional space, evaluating model performance for each parameter configuration. The same bounds are used for all trainable weights (uniform bounding hypercube). We initially uniformly randomly sample from within the bounding hypercube a set of candidate vectors that contains three times as many vectors as the number of trainable weights. Each vector has dimension equal to the number of trainable weights, with each entry in a vector corresponding to an individual trainable weight.

For each candidate vector, we replace the values of the trainable weights in the original LoRA matrices with the corresponding candidate entries and evaluate on the entire train and validation sets. We use the initial set of candidate vectors with evaluated accuracies to seed the Gaussian process optimization. We iterate over all candidate vectors, generating up to 400 additional, new candidate vectors with the Gaussian process optimizer, and take the performance from the candidate vector with highest validation accuracy as the final performance (with weights of the final trained parameters replaced with that candidate vector's entries).

This process entirely eliminates the need for hyperparameter search and significantly reduces training time and overhead, as only the forward pass must be computed for optimization. In this very-few parameter limit, we find that performance from models trained with Bayesian optimization matches or exceeds the performance of the best performing hyperparameter configurations, enabling us to efficiently sample the elicitation frontier.

Due to the fact that Bayesian optimization does not employ training epochs, however, we are unable to calculate prequential minimum description lengths for the resulting fine-tuned models. Instead, we calculate prequential MDL only for models fine-tuned using gradient descent.

C.5 Computational Requirements

- Hardware: Single NVIDIA H100 80 GB GPU
- Runtime: 10 minutes to 36 hours depending on parameters trained and total training epochs
- Memory: 20-60 GB depending on model size and batch size
- Total experiments: ~3500 individual runs

D Dataset details and fine-tuning prompts

D.1 GSM-8K-CoT-Choice

GSM-8K-CoT-Choice was created by adapting ~1000 questions from the GSM-8K benchmark [24] to a binary classification task. For each question, four chain-of-thought reasoning solution attempts and answers were generated with Claude-3.5-Sonnet. In each example, the model is presented with a question, a solution attempt, and a final answer and asked to identify whether the solution attempt and final answer are correct. The dataset is split evenly between correct and incorrect solutions, with two of each per question. Model-generated solution attempts were rigorously validated by a human and additional language model to ensure accurate labeling.

D.1.1 Dataset generation procedure

We use the following procedure to generate the dataset:

1. **Question selection:** Sample 1,000 problems from GSM-8K training and validation sets
2. **Solution generation:** For each problem, generate 4 chain-of-thought solutions using Claude-3.5-Sonnet with structured JSON output (2 correct examples, 2 incorrect examples)
3. **Answer validation:**
    - Correct solutions: Verify exact string match with ground-truth answer
    - Incorrect solutions: Ensure no match with ground-truth answer
4. **Quality control:** Manual verification of flagged examples where automated validation fails
5. **Balance Verification:** Confirm exactly 2 correct and 2 incorrect solutions per problem

D.2 Dataset statistics

- Total examples: 4,000 (1,000 questions $\times$ 4 solutions each)
- Class balance: 50% correct, 50% incorrect
- Token length: 99% of examples < 384 tokens (using Llama tokenizer)

D.3 Fine-tuning Prompts

Systematic evaluation revealed minimal sensitivity to prompt variations when instructions are present (<0.5% performance difference). Without instructions, performance degrades by approximately 2%, motivating our standardized prompt templates.

D.3.1 GSM-8K-CoT-Choice

```
'''
You will be given a math problem, a step-by-step solution attempt,
and a final answer.
Evaluate the correctness of the solution attempt and answer to the
problem.
Respond with ONLY 'Correct' or 'Incorrect.'
```

````
Problem: {problem}
Solution Attempt: {solution attempt}
Answer: {answer}
Respond with ONLY 'Correct' or 'Incorrect'. Response:
```
````

D.3.2 ARC-Easy & ARC-Challenge

````
```
You will be given a question and a series of answer choices labeled
A, B, C, D, etc.
Select the correct answer from the choices.
Respond with ONLY the uppercase letter of the correct answer (A-E).
Question: {question}
Answer choices:
A: {choice A}
B: {choice B}
... {other choices}
Response:
```
````

D.3.3 BoolQ

````
```
You will be given a passage and a question.
Determine the answer to the question based on the information in the
passage.
Respond with ONLY 'Yes' or 'No.'
Passage: {passage}
Question: {question}
Response:
```
````

D.3.4 Alpaca

````
```
Below is an instruction that describes a task{, paired with an input
that provides further context}.
Write a response that appropriately completes the request.

### Instruction:
{instruction}

### Input:
{input}

### Response:
```
````

E Preliminaries

Notation. Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$ denote a downstream task dataset of size n. We denote by θ_0 the parameters of the pretrained base model and by $\theta_{p,s,h}$ the LoRA-adapted parameters ontained by training exactly p adapter weights with random seed s and hyperparameters h. We write

$$\mathrm{Acc}(\theta) \; = \; \frac{1}{|\mathcal{D}_{\mathrm{val}}|} \sum_{x,y \in \mathcal{D}_{\mathrm{val}}} \mathbf{1}\{\hat{y}_\theta(x) = y\} \tag{2}$$

for the validation accuracy of the model θ. We further define the zero-shot baseline $\mathrm{Acc}_0 = \mathrm{Acc}(\theta_0)$ and the accuracy of the fully fine-tuned model $\mathrm{Acc}_{\mathrm{full}}$.

Parameter Budget Selection. We select parameter budgets $\{p_1, ..., p_k\}$ on a logarithmic scale to ensure adequate resolution across orders of magnitude.
````

Pareto Frontier Extraction. To characterize the trade-off between parameter budget and performance, we choose a set of budgets $p \in \{p_1, \ldots, p_K\}$. For each budget p, we perform:

- $S = 3 - 5$ independent random seeds s.
- For each (p, s), up to 50 hyperparameter trials h via Bayesian optimization with Hyperband [25].
- Record the maximal validation accuracy

$$f(p) \;=\; \max_{s,h} \; \mathrm{Acc}\big(\theta_{p,s,h}\big). \tag{3}$$

The set of points $\big\{(p,\; f(p))\big\}_{p=1}^{K}$ defines the empirical Pareto frontier. We fit a generalized logistic in $\log p$ for its flexibility in modeling diverse S-shaped growth patterns:

$$f(p) \;\approx\; \mathrm{Acc}_{\infty} \;-\; \frac{\mathrm{Acc}_{\infty} - \mathrm{Acc}_0}{\big(1 + \exp\left(-a - b\,\log p\right)\big)^{\nu}}, \tag{4}$$

using nonlinear least squares to recover $(\mathrm{Acc}_0, \mathrm{Acc}_{\infty}, a, b, \nu)$. This formulation accommodates asymmetric growth curves observed across different tasks.

Bootstrap Confidence Intervals. We employ nonparametric bootstrap with 10^4-10^5 samples:

1. For each iteration, remove one frontier point uniformly at random
2. Fit logistic curve to remaining points
3. Compute 95% confidence intervals from empirical distribution of fitted parameters

Empirical Metrics. We evaluate two additional quantities on the frontier models θ_{p,s^*,h^*} achieving $f(p)$:

1. *Gap closure*:
$$\mathrm{GC}(p) = \frac{f(p) - \mathrm{Acc}_0}{\mathrm{Acc}_{\mathrm{full}} - \mathrm{Acc}_0} \;\in [0,1].$$
2. *Online MDL* [2, 26]:
$$\mathrm{MDL}(p) = \sum_{i=1}^{n} \ell\big(\theta_{p,s^*,h^*};\, x_i, y_i\big),$$
where ℓ is the per-example training loss, which is updated after each example (or batch) has been processed, and the sum is taken over the first epoch only, such that each train set example has been seen exactly once by the model.

F Analysis of alternative fitting functions

F.1 Alternative fitting functions

We analyze four alternative fitting functions in addition to a generalized logistic: saturating exponential, power law, (piecewise) broken power law, and smooth broken power law (power law multiplied with a logistic window to maintain differentiability).

Generalized logistic:

$$y = y_{\min} + \frac{y_{\max} - y_{\min}}{\left(1 + e^{-k(x - x_0)}\right)^{\nu}} \tag{5}$$

Saturating exponential:

$$y = y_{\max} - Ae^{-B(x - x_0)} \tag{6}$$

Power law:

$$y = ax^{b} \tag{7}$$

Broken power law:

$$y = \begin{cases} a + bx & \text{for } x \leq x_k \\ a + bx_k + c(x - x_k)^d & \text{for } x > x_k \end{cases} \tag{8}$$

Smooth broken power law:

$$y = \begin{cases} a + bx & \text{for } x << x_k \\ a + bx_k + c(x - x_k)^d & \text{for } x >> x_k \end{cases} \tag{9}$$

F.2 Model comparison metrics

We evaluate model fit using:

- Akaike information criterion (AIC): $2k - 2\ln \hat{L}$
- Bayesian information criterion (BIC): $k \ln n - 2\ln \hat{L}$
- Mean square error (MSE): $\frac{1}{n}\sum_i (y_i - \hat{y}_i)^2$,

where k is the number of fitting parameters (not trainable parameters in this instance), n is the number of data points (i.e., the number of Pareto frontier points used for generating the fitted curve), and $\hat{L}$ is the maximized likelihood (sum of squared errors). Figure 6 shows the results of curve fitting for all three Llama model sizes (1B, 3B, and 8B) fine-tuned on GSM-8K-CoT-Choice with corresponding AIC and BIC for each fit.

G Results from additional datasets

G.1 Alpaca

Figure 5 shows results from fine-tuning Llama-3.2-1B and Llama-3.2-3B on the Alpaca instruction-tuning dataset.

G.2 BoolQ

Figure 5 shows results from fine-tuning Llama-3.2-1B and Llama-3.2-3B on BoolQ. The logistic scaling of accuracy with increasing trainable parameters remains consistent in BoolQ, with both models requiring ~100 parameters to achieve 50% performance gap closure between zero-shot accuracy and full fine-tune accuracy.

G.3 Dataset Coverage Limitations

BoolQ and Alpaca experiments include only Llama-3.2-1B and 3B models due to computational constraints. The consistent logistic scaling observed across completed experiments suggests similar patterns would emerge for the 8B model, though empirical validation remains future work.

H Results across hyperparameter and seed variations

Because we fine-tune with numbers of trainable parameters spanning several orders of magnitude, we optimize hyperparameters at each parameter budget for each seed, performing up to 50 trials of Bayesian optimization with Hyperband. Figure 8 shows results for hyperparameter optimization trials for Llama-3.1-8B fine-tuned on ARC-Challenge (all seeds). Hyperparameter optimization is required for each parameter budget as hyperparameters (learning rate, in particular) can differ significantly depending on the sparsity of trainable parameters. For the highest performing seeds, as shown in Figure 2, the highest performing runs for each seed demonstrate little variability in final validation accuracy and largely fall within the 95% confidence interval of the fit to the Pareto frontier.

I Construction of Prequential MDL and RDA

Because the true Minimum Description Length is uncomputable, Perez et al. [17] propose to upper bound it with the data's MDL obtained by prequential coding:

$$\mathcal{L}(y_{1:N} \mid x_{1:N}, f) \; < \; \mathcal{L}(y_{1:N} \mid x_{1:N}), \tag{10}$$

which states that a model that uses a domain-relevant capability to generate a label obtains a more compact explanation of the label-generation process, which enables it to succinctly model the dataset itself.

We can alternatively view this label-generation learning process as a communication protocol for sending the labels for the dataset from someone who has them to someone who does not. Alice, who has the labels, wants to share them with Bob, who does not, and they want to find the smallest file Alice needs to send to Bob so he can have them. Alice and Bob share the same base model M and learning algorithm A (which contains all information Bob needs to know to replicate Alice's training on his end), update the model after each label, and code that label with its instantaneous cross-entropy loss. This "online" code length converges in expectation to the true MDL when the learner (i.e., model) is Bayes-optimal.

Perez et al. [17] introduce Rissanen Data Analysis to replace MDL with a block-wise code that retrains the model (calculating loss and updating gradients) only after a batch of examples has been seen, until all examples

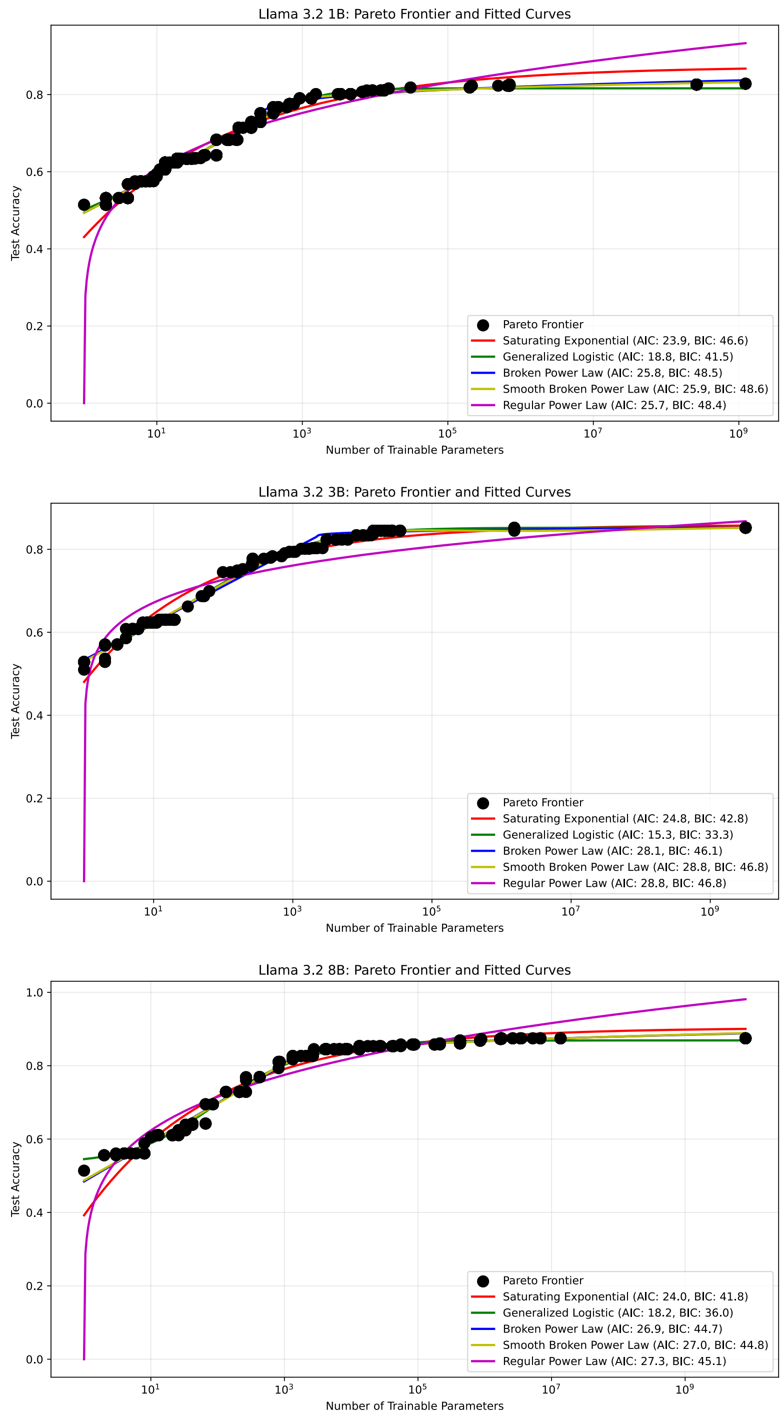


Figure 6: Fitting of curves with corresponding AIC and BIC for 1B, 3B, and 8B Llama models fine-tuned on GSM-8K-CoT-Choice. For all models, AIC and BIC are consistently lowest for fits to a generalized logistic function.

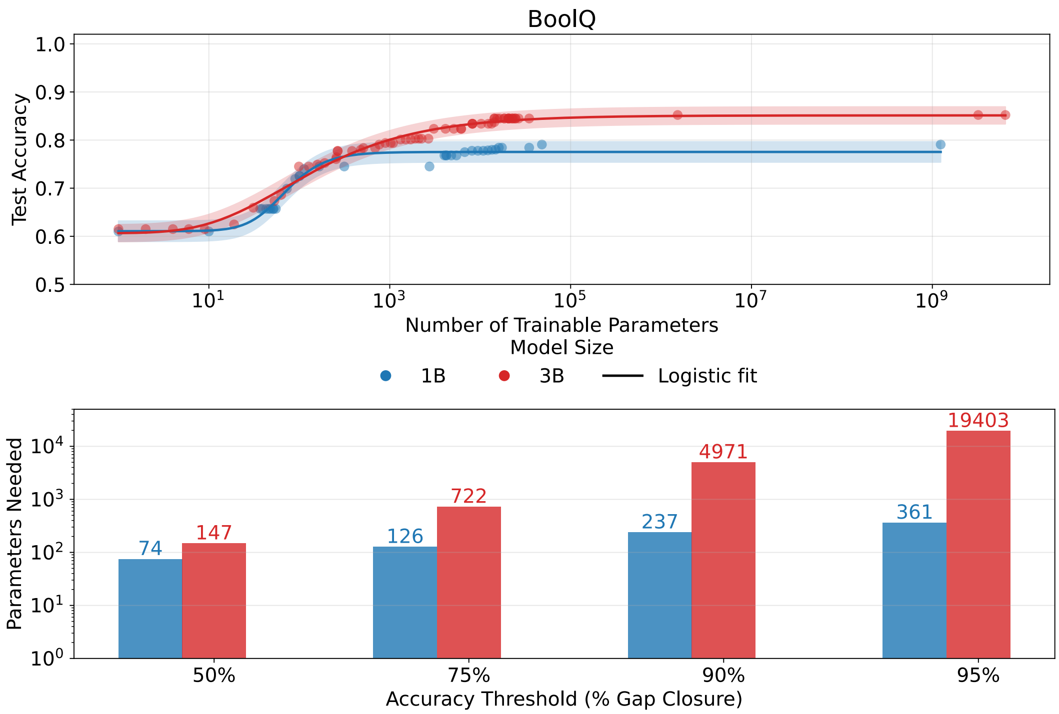


Figure 7: **Fine-tuning ~100 parameters in Llama-3.2-1B and Llama-3.2-3B recovers 50% of the performance gap on BoolQ.** 2σ standard errors are shown as shaded regions.

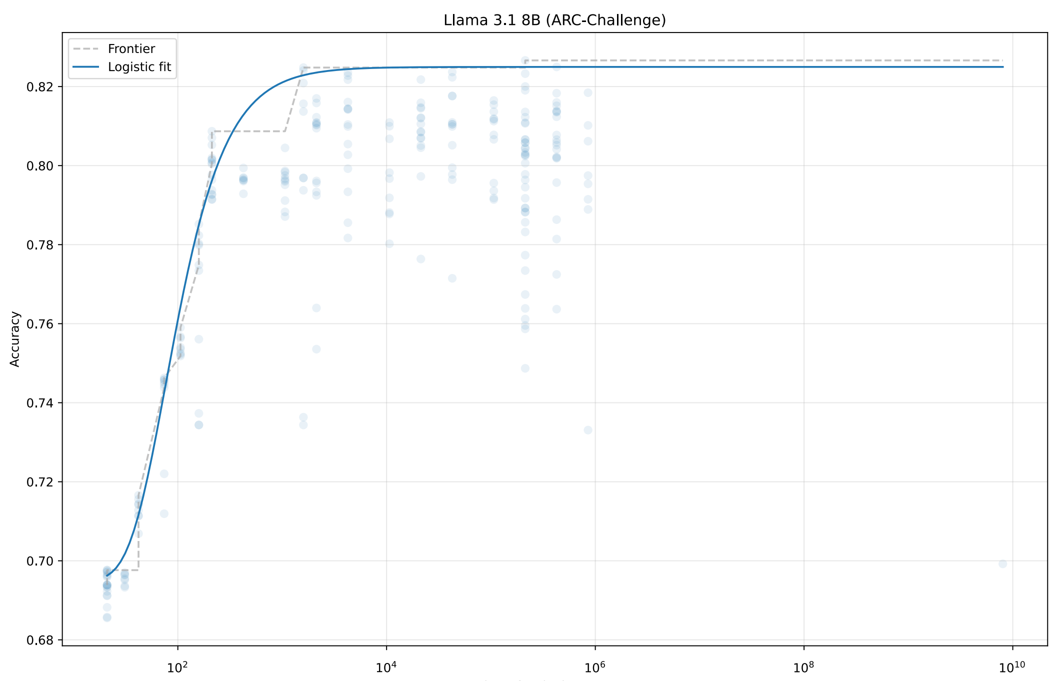


Figure 8: Llama-3.1-8B fine-tuned on ARC-Challenge. Each semi-transparent point indicates a fine-tuned model with a unique set of hyperparameters and random seed. The logistic fit to the Pareto frontier is denoted by a solid blue line, and a gray dashed line connects the points that define the Pareto frontier.

in the dataset have been seen a single time. This still constitutes an upper bound on the minimum information Alice must send because each batch is encoded with a less recent model than in the prequential setting.

Canonically, RDA either keeps the model architecture and optimizer fixed or charges $\log_2 M$ additional bits if an ensemble of M learners is allowed, averaging codelengths over five random seeds. We adopt the same formulation.

We additionally ask the question, "How much do you have to specify to the model to get it to demonstrate knowledge it already has?". An intuitive way of measuring how much guiding a model needs to be able to access (some fraction of) a pre-existing capability is by counting the number of parameters that must be updated for it to be able to achieve a certain performance threshold on a task that requires the skill. In our elicitation framework, we employ MDL to measure how efficiently a model can encode task labels during its initial exposure to the training data. We compute MDL using the first epoch of training from a randomly initialized LoRA adapter, not from a converged model, as this provides a measure of how quickly a model can develop an efficient task representation given minimal parameter updates.

Comparing MDL and final test accuracy for a given model then provides information about how well a model can quickly develop a compact description of the task, as well as information about the model's ultimate asymptotic performance when given effectively unlimited parameter updates. For most tasks, MDL compression tracks accuracy improvements closely (Figure 3). However, divergences between MDL and accuracy curves (as seen for Llama-3.2-1B on ARC-Challenge) provide diagnostic information: when accuracy improves without commensurate MDL compression, the model may be learning rather than eliciting.

We can then recast our fine-tuning elicitation setting as a communication protocol between Alice and Bob:

Table 3: **Casting PEFT elicitation as an Alice $\rightarrow$ Bob communication game.**

Element	**Description**
Alice & Bob	Identical copies of pretrained base model M_0
Shared knowledge	Base model M_0, learning algorithm A, dataset inputs $\{x_1, ..., x_n\}$
Alice's goal	Communicate dataset labels $\{y_1, ..., y_n\}$ to Bob
Communication (message)	(i) A one-shot header describing number of trainable parameters k to initialize the LoRA adapters θ_k with, parameter selection seed s, hyperparameters h; + (ii) block-wise stream of label codes
Instructions for accessing capability f	The adapter update $\Delta\theta_k$ containing k learned weights that Bob obtains as he trains his copy of M_0

All header terms are paid once, before the first label is transmitted, and the final summation term in the message is the usual prequential code computed only on the labels, as the inputs x and backbone M_0 are shared *a priori*, following Perez et al. [17].

Our MDL calculation proceeds as follows:

1. **Initialization:** Given parameter budget k, randomly select k parameters from LoRA adapters using seed s
2. **Hyperparameter selection:** Through Bayesian optimization, identify hyperparameters h that maximize validation accuracy
3. **Prequential MDL computation:** For the optimal (s, h) configuration, compute prequential loss during the first epoch only

Following the prequential MDL formulation [17, 27–29], we encode each example sequentially using the model trained only on previously observed examples:

$$\mathcal{L}_{\text{preq}} = -\log P(\mathcal{D}|\hat{\theta}) = -\sum_{t=1}^{T} \log(y_t|x_t; \theta_{t-1}), \tag{11}$$

in which t indexes gradient steps in the first epoch, ensuring each encoded data point is unseen at encoding time. Practically, this is done by initializing the pretrained model, presenting each training example (or batch) exactly once, performing a gradient step immediately after an example (batch) has been seen, and accumulating the per-example cross-entropy losses over all examples (batches). This directly measures the incremental compression efficiency of a model during fine-tuning.

Intuitive Interpretation: Consider MDL as measuring how efficiently a model can "compress" its understanding of a task within its first pass over the training dataset. If updating merely 10 parameters enables the model to compress task labels by hundreds of bits within the first epoch alone, this suggests the model already possesses a structured representation of the task—the parameters merely serve as a "key" to unlock this latent knowledge. Conversely, if compression improves only marginally until thousands of parameters are modified, the capability may likely require learning from scratch rather than elicitation, as it cannot be easily unlocked or accessed.

Algorithm 1 Prequential MDL Computation.

Input: Base model M_0, dataset $\mathcal{D} = \{x_i, y_i\}_i^N$, parameter budget k
Initialize model M_0 with LoRA adapters with k trainable parameters, $\Delta\theta_k$, (using seed s)
Initialize MDL, $\mathcal{L}_k = 0$
for each batch B in epoch 1 **do**
  Compute loss $\ell_B = \mathrm{CrossEntropy}(M_0(\Delta\theta_k, B), \mathrm{labels}(B))$
  $\mathcal{L}_k \mathrel{+}= \ell_B \times |B|$
  Update trainable parameters $\Delta\theta_k$ via gradient descent
end for
Return $\mathcal{L}_k$

Interpretation for elicitation: In scenarios with high compression and low trainable parameter counts, the model rapidly develops an efficient encoding scheme with a limited set of locations where it can make changes, suggesting it leverages pre-existing structured representations. In scenarios with low compression, the model struggles to compress the label stream efficiently, indicating that the capability is not easily accessible or utilized—potentially because it is absent. This differs from standard fine-tuning analysis because we measure compression during initial learning, not post-convergence, and explicitly account for the information cost of specifying which parameters to train. The resulting compression rate then reveals how much "latent structure" the model can immediately access.

The MDL compression $\Delta_k = \bar{\mathcal{L}}_p^{(0)} - \bar{\mathcal{L}}^{(k)}$ is therefore a direct, information-theoretic measure of how helpful the latent capability is when k parameters are available to specify how to access it. In the language of Alice and Bob, if granting Alice access to f shortens the MDL, then f must convey information that was previously missing. The magnitude of the MDL drop, measured in bits, quantifies the intrinsic value of the capability to the task.

In our setting, every capability is presumed to be latent in the pretrained (base) model M_0. A small adapter $\Delta\theta_k$ with only k trainable weights serves as a pointer to that capability; training enables the model to develop this pointer, as $\Delta\theta_k$ plays the role of invoking f. If a short message ($\Delta\theta_k$) lets the model compress the subsequent label stream by hundreds of bits, it is highly probable that the original base model already contained a highly structured representation of the task. Conversely, if the MDL shrinks slowly or not at all until k becomes large, the capability was likely absent and must be learned rather than elicited.

J Data and Code Availability

GSM-8K-CoT-Choice dataset and evaluation scripts will be released with publication.

- LoRA implementation with random masking
- Bayesian optimization for ultra-low parameter regimes
- MDL computation utilities
- Frontier extraction and fitting procedures