CAIAC

Spring 2025

[WEEK 1]

Introduction to using gatpack

READINGS

Neural Networks
(3Blue1Brown, 2024) Hello World

(3Blue1Brown, 2024)

Neural Networks
3Blue1Brown (2024)
youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&feature=shared

Hello World
3Blue1Brown (2024)
youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&feature=shared

Notes:

Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

*Equal contributions

We investigate the optimal model size and number of tokens for training a transformer language model under a given compute budget. We find that current large language models are significantly undertrained, a consequence of the recent focus on scaling language models whilst keeping the amount of training data constant. By training over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens, we find that for compute-optimal training, the model size and the number of training tokens should be scaled equally: for every doubling of model size the number of training tokens should also be doubled. We test this hypothesis by training a predicted compute-optimal model, *Chinchilla*, that uses the same compute budget as *Gopher* but with 70B parameters and 4× more more data. *Chinchilla* uniformly and significantly outperforms *Gopher* (280B), GPT-3 (175B), Jurassic-1 (178B), and Megatron-Turing NLG (530B) on a large range of downstream evaluation tasks. This also means that *Chinchilla* uses substantially less compute for fine-tuning and inference, greatly facilitating downstream usage. As a highlight, *Chinchilla* reaches a state-of-the-art average accuracy of 67.5% on the MMLU benchmark, greater than a 7% improvement over *Gopher*.

1. Introduction

Recently a series of *Large Language Models* (LLMs) have been introduced (Brown et al., 2020; Lieber et al., 2021; Rae et al., 2021; Smith et al., 2022; Thoppilan et al., 2022), with the largest dense language models now having over 500 billion parameters. These large autoregressive transformers (Vaswani et al., 2017) have demonstrated impressive performance on many tasks using a variety of evaluation protocols such as zero-shot, few-shot, and fine-tuning.

The compute and energy cost for training large language models is substantial (Rae et al., 2021; Thoppilan et al., 2022) and rises with increasing model size. In practice, the allocated training compute budget is often known in advance: how many accelerators are available and for how long we want to use them. Since it is typically only feasible to train these large models once, accurately estimating the best model hyperparameters for a given compute budget is critical (Tay et al., 2021).

Kaplan et al. (2020) showed that there is a power law relationship between the number of parameters in an autoregressive language model (LM) and its performance. As a result, the field has been training larger and larger models, expecting performance improvements. One notable conclusion in Kaplan et al. (2020) is that large models should not be trained to their lowest possible loss to be compute optimal. Whilst we reach the same conclusion, we estimate that large models should be trained for many more training tokens than recommended by the authors. Specifically, given a 10× increase computational budget, they suggests that the size of the model should increase 5.5× while the number of training tokens should only increase 1.8×. Instead, we find that model size and the number of training tokens should be scaled in equal proportions.

Following Kaplan et al. (2020) and the training setup of GPT-3 (Brown et al., 2020), many of the recently trained large models have been trained for approximately 300 billion tokens (Table 1), in line with the approach of predominantly increasing model size when increasing compute.

Corresponding authors: {jordanhoffmann|sborgeaud|amensch|sifre}@deepmind.com

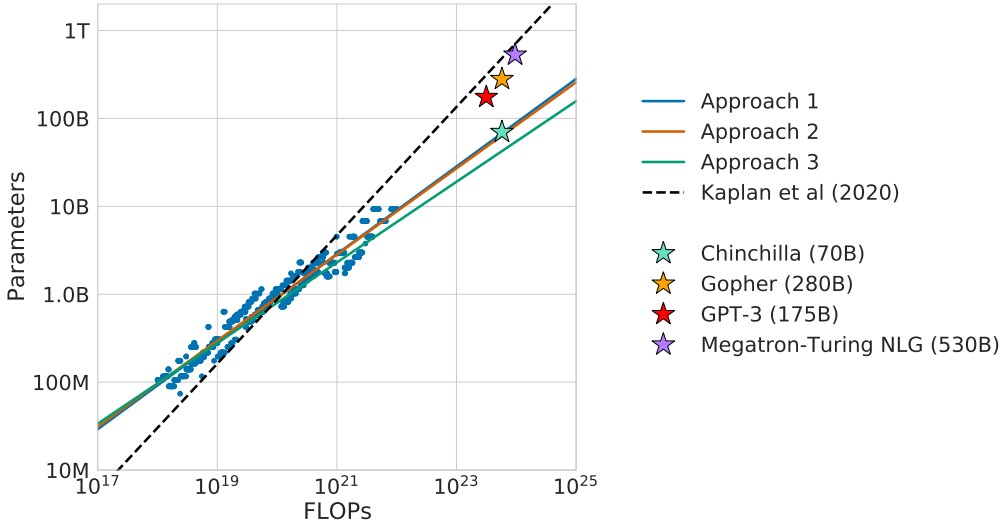


Figure 1 | **Overlaid predictions.** We overlay the predictions from our three different approaches, along with projections from Kaplan et al. (2020). We find that all three methods predict that current large models should be substantially smaller and therefore trained much longer than is currently done. In Figure A3, we show the results with the predicted optimal tokens plotted against the optimal number of parameters for fixed FLOP budgets. ***Chinchilla* outperforms *Gopher* and the other large models (see Section 4.2).**

In this work, we revisit the question: *Given a fixed FLOPs budget,*[1] *how should one trade-off model size and the number of training tokens?* To answer this question, we model the final pre-training loss[2] $L(N, D)$ as a function of the number of model parameters N, and the number of training tokens, D. Since the computational budget C is a deterministic function FLOPs(N, D) of the number of seen training tokens and model parameters, we are interested in minimizing L under the constraint FLOPs$(N, D) = C$:

$$N_{opt}(C), D_{opt}(C) = \underset{N,D \text{ s.t. } \text{FLOPs}(N,D)=C}{\operatorname{argmin}} L(N, D). \tag{1}$$

The functions $N_{opt}(C)$, and $D_{opt}(C)$ describe the optimal allocation of a computational budget C. We empirically estimate these functions based on the losses of over 400 models, ranging from under 70M to over 16B parameters, and trained on 5B to over 400B tokens – with each model configuration trained for several different training horizons. Our approach leads to considerably different results than that of Kaplan et al. (2020). We highlight our results in Figure 1 and how our approaches differ in Section 2.

Based on our estimated compute-optimal frontier, we predict that for the compute budget used to train *Gopher*, an optimal model should be 4 times smaller, while being training on 4 times more tokens. We verify this by training a more *compute-optimal* 70B model, called *Chinchilla*, on 1.4 trillion tokens. Not only does *Chinchilla* outperform its much larger counterpart, *Gopher*, but its reduced model size reduces inference cost considerably and greatly facilitates downstream uses on smaller hardware. The energy cost of a large language model is amortized through its usage for inference an fine-tuning. The benefits of a more optimally trained smaller model, therefore, extend beyond the immediate benefits of its improved performance.

[1] For example, knowing the number of accelerators and a target training duration.

[2] For simplicity, we perform our analysis on the smoothed training loss which is an unbiased estimate of the test loss, as we are in the infinite data regime (the number of training tokens is less than the number of tokens in the entire corpus).

Table 1 | **Current LLMs**. We show five of the current largest dense transformer models, their size, and the number of training tokens. Other than LaMDA (Thoppilan et al., 2022), most models are trained for approximately 300 billion tokens. We introduce *Chinchilla*, a substantially smaller model, trained for much longer than 300B tokens.

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

2. Related Work

Large language models. A variety of large language models have been introduced in the last few years. These include both dense transformer models (Brown et al., 2020; Lieber et al., 2021; Rae et al., 2021; Smith et al., 2022; Thoppilan et al., 2022) and mixture-of-expert (MoE) models (Du et al., 2021; Fedus et al., 2021; Zoph et al., 2022). The largest dense transformers have passed 500 billion parameters (Smith et al., 2022). The drive to train larger and larger models is clear—so far increasing the size of language models has been responsible for improving the state-of-the-art in many language modelling tasks. Nonetheless, large language models face several challenges, including their overwhelming computational requirements (the cost of training and inference increase with model size) (Rae et al., 2021; Thoppilan et al., 2022) and the need for acquiring more high-quality training data. In fact, in this work we find that larger, high quality datasets will play a key role in any further scaling of language models.

Modelling the scaling behavior. Understanding the scaling behaviour of language models and their transfer properties has been important in the development of recent large models (Hernandez et al., 2021; Kaplan et al., 2020). Kaplan et al. (2020) first showed a predictable relationship between model size and loss over many orders of magnitude. The authors investigate the question of choosing the optimal model size to train for a given compute budget. Similar to us, they address this question by training various models. Our work differs from Kaplan et al. (2020) in several important ways. First, the authors use a fixed number of training tokens and learning rate schedule for all models; this prevents them from modelling the impact of these hyperparameters on the loss. In contrast, we find that setting the learning rate schedule to approximately match the number of training tokens results in the best final loss regardless of model size—see Figure A1. For a fixed learning rate cosine schedule to 130B tokens, the intermediate loss estimates (for $D' << 130\text{B}$) are therefore overestimates of the loss of a model trained with a schedule length matching D'. Using these intermediate losses results in underestimating the effectiveness of training models on less data than 130B tokens, and eventually contributes to the conclusion that model size should increase faster than training data size as compute budget increases. In contrast, our analysis predicts that both quantities should scale at roughly the same rate. Secondly, we include models with up to 16B parameters, as we observe that there is slight curvature in the FLOP-loss frontier (see Appendix E)—in fact, the majority of the models used in our analysis have more than 500 million parameters, in contrast the majority of runs in Kaplan et al. (2020) are significantly smaller—many being less than 100M parameters.

Recently, Clark et al. (2022) specifically looked in to the scaling properties of Mixture of Expert

language models, showing that the scaling with number of experts diminishes as the model size increases—their approach models the loss as a function of two variables: the model size and the number of experts. However, the analysis is done with a fixed number of training tokens, as in Kaplan et al. (2020), potentially underestimating the improvements of branching.

Estimating hyperparameters for large models. The model size and the number of training tokens are not the only two parameters to chose when selecting a language model and a procedure to train it. Other important factors include learning rate, learning rate schedule, batch size, optimiser, and width-to-depth ratio. In this work, we focus on model size and the number of training steps, and we rely on existing work and provided experimental heuristics to determine the other necessary hyperparameters. Yang et al. (2021) investigates how to choose a variety of these parameters for training an autoregressive transformer, including the learning rate and batch size. McCandlish et al. (2018) finds only a weak dependence between optimal batch size and model size. Shallue et al. (2018); Zhang et al. (2019) suggest that using larger batch-sizes than those we use is possible. Levine et al. (2020) investigates the optimal depth-to-width ratio for a variety of standard model sizes. We use slightly less deep models than proposed as this translates to better wall-clock performance on our hardware.

Improved model architectures. Recently, various promising alternatives to traditional dense transformers have been proposed. For example, through the use of conditional computation large MoE models like the 1.7 trillion parameter Switch transformer (Fedus et al., 2021), the 1.2 Trillion parameter GLaM model (Du et al., 2021), and others (Artetxe et al., 2021; Zoph et al., 2022) are able to provide a large effective model size despite using relatively fewer training and inference FLOPs. However, for very large models the computational benefits of routed models seems to diminish (Clark et al., 2022). An orthogonal approach to improving language models is to augment transformers with explicit retrieval mechanisms, as done by Borgeaud et al. (2021); Guu et al. (2020); Lewis et al. (2020). This approach effectively increases the number of data tokens seen during training (by a factor of ~ 10 in Borgeaud et al. (2021)). This suggests that the performance of language models may be more dependant on the size of the training data than previously thought.

3. Estimating the optimal parameter/training tokens allocation

We present three different approaches to answer the question driving our research: *Given a fixed FLOPs budget, how should one trade-off model size and the number of training tokens?* In all three cases we start by training a range of models varying both model size and the number of training tokens and use the resulting training curves to fit an empirical estimator of how they should scale. We assume a power-law relationship between compute and model size as done in Clark et al. (2022); Kaplan et al. (2020), though future work may want to include potential curvature in this relationship for large model sizes. The resulting predictions are similar for all three methods and suggest that parameter count and number of training tokens should be increased equally with more compute[3]—with proportions reported in Table 2. This is in clear contrast to previous work on this topic and warrants further investigation.

[3]We compute FLOPs as described in Appendix F.

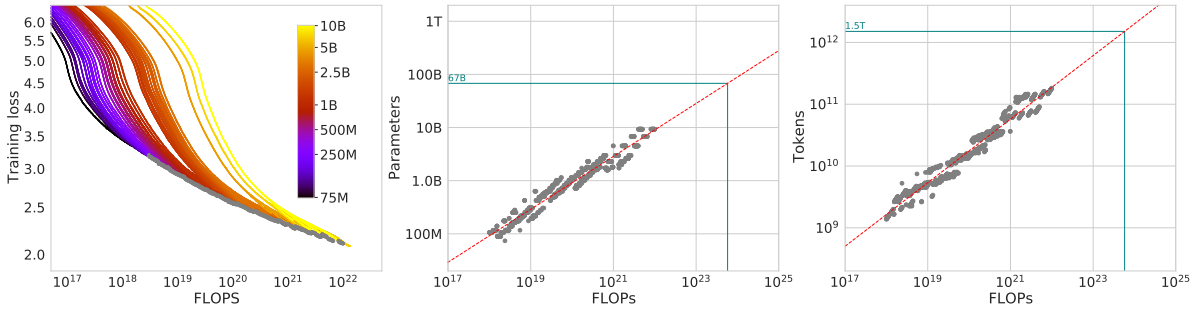


Figure 2 | **Training curve envelope.** On the **left** we show all of our different runs. We launched a range of model sizes going from 70M to 10B, each for four different cosine cycle lengths. From these curves, we extracted the envelope of minimal loss per FLOP, and we used these points to estimate the optimal model size (**center**) for a given compute budget and the optimal number of training tokens (**right**). In green, we show projections of optimal model size and training token count based on the number of FLOPs used to train *Gopher* (5.76×10^{23}).

3.1. Approach 1: Fix model sizes and vary number of training tokens

In our first approach we vary the number of training steps for a fixed family of models (ranging from 70M to over 10B parameters), training each model for 4 different number of training sequences. From these runs, we are able to directly extract an estimate of the minimum loss achieved for a given number of training FLOPs. Training details for this approach can be found in Appendix D.

For each parameter count N we train 4 different models, decaying the learning rate by a factor of 10× over a horizon (measured in number of training tokens) that ranges by a factor of 16×. Then, for each run, we smooth and then interpolate the training loss curve. From this, we obtain a continuous mapping from FLOP count to training loss for each run. Then, for each FLOP count, we determine which run achieves the lowest loss. Using these interpolants, we obtain a mapping from any FLOP count C, to the most efficient choice of model size N and number of training tokens D such that $\text{FLOPs}(N, D) = C$.[4] At 1500 logarithmically spaced FLOP values, we find which model size achieves the lowest loss of all models along with the required number of training tokens. Finally, we fit power laws to estimate the optimal model size and number of training tokens for any given amount of compute (see the center and right panels of Figure 2), obtaining a relationship $N_{opt} \propto C^a$ and $D_{opt} \propto C^b$. We find that $a = 0.50$ and $b = 0.50$—as summarized in Table 2. In Section D.4, we show a head-to-head comparison at 10^{21} FLOPs, using the model size recommended by our analysis and by the analysis of Kaplan et al. (2020)—using the model size we predict has a clear advantage.

3.2. Approach 2: IsoFLOP profiles

In our second approach we vary the model size[5] for a fixed set of 9 different training FLOP counts[6] (ranging from 6×10^{18} to 3×10^{21} FLOPs), and consider the final training loss for each point[7]. in contrast with Approach 1 that considered points (N, D, L) along the entire training runs. This allows us to directly answer the question: For a given FLOP budget, what is the optimal parameter count?

[4]Note that all selected points are within the last 15% of training. This suggests that when training a model over D tokens, we should pick a cosine cycle length that decays 10× over approximately D tokens—see further details in Appendix B.

[5]In approach 2, model size varies up to 16B as opposed to approach 1 where we only used models up to 10B.

[6]The number of training tokens is determined by the model size and training FLOPs.

[7]We set the cosine schedule length to match the number of tokens, which is optimal according to the analysis presented in Appendix B.

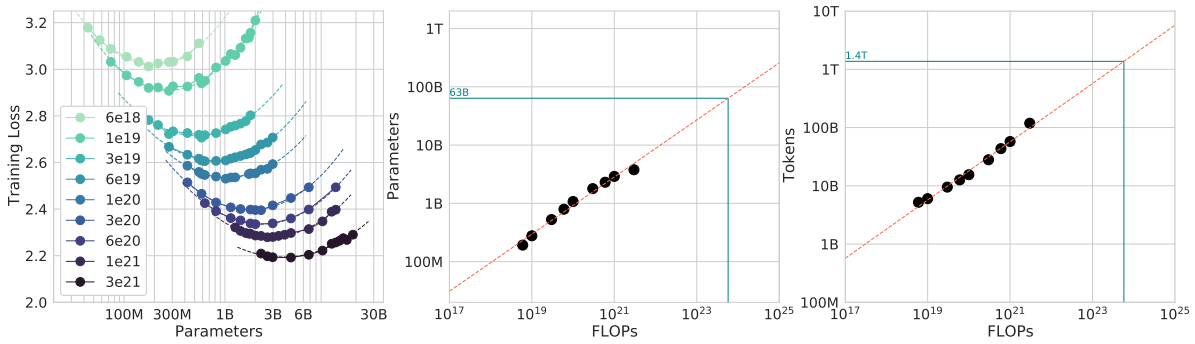


Figure 3 | **IsoFLOP curves.** For various model sizes, we choose the number of training tokens such that the final FLOPs is a constant. The cosine cycle length is set to match the target FLOP count. We find a clear valley in loss, meaning that for a given FLOP budget there is an optimal model to train (**left**). Using the location of these valleys, we project optimal model size and number of tokens for larger models (**center** and **right**). In green, we show the estimated number of parameters and tokens for an *optimal* model trained with the compute budget of *Gopher*.

For each FLOP budget, we plot the final loss (after smoothing) against the parameter count in Figure 3 (left). In all cases, we ensure that we have trained a diverse enough set of model sizes to see a clear minimum in the loss. We fit a parabola to each IsoFLOPs curve to directly estimate at what model size the minimum loss is achieved (Figure 3 (left)). As with the previous approach, we then fit a power law between FLOPs and loss-optimal model size and number of training tokens, shown in Figure 3 (center, right). Again, we fit exponents of the form $N_{opt} \propto C^a$ and $D_{opt} \propto C^b$ and we find that $a = 0.49$ and $b = 0.51$—as summarized in Table 2.

3.3. Approach 3: Fitting a parametric loss function

Lastly, we model all final losses from experiments in Approach 1 & 2 as a parametric function of model parameter count and the number of seen tokens. Following a classical risk decomposition (see Section D.2), we propose the following functional form

$$\hat{L}(N, D) \triangleq E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}. \tag{2}$$

The first term captures the loss for an ideal generative process on the data distribution, and should correspond to the entropy of natural text. The second term captures the fact that a perfectly trained transformer with N parameters underperforms the ideal generative process. The final term captures the fact that the transformer is not trained to convergence, as we only make a finite number of optimisation steps, on a sample of the dataset distribution.

Model fitting. To estimate (A, B, E, α, β), we minimize the Huber loss (Huber, 1964) between the predicted and observed log loss using the L-BFGS algorithm (Nocedal, 1980):

$$\min_{A,B,E,\alpha,\beta} \sum_{\text{Runs } i} \text{Huber}_\delta\left(\log \hat{L}(N_i, D_i) - \log L_i\right) \tag{3}$$

We account for possible local minima by selecting the best fit from a grid of initialisations. The Huber loss ($\delta = 10^{-3}$) is robust to outliers, which we find important for good predictive performance over held-out data points. Section D.2 details the fitting procedure and the loss decomposition.

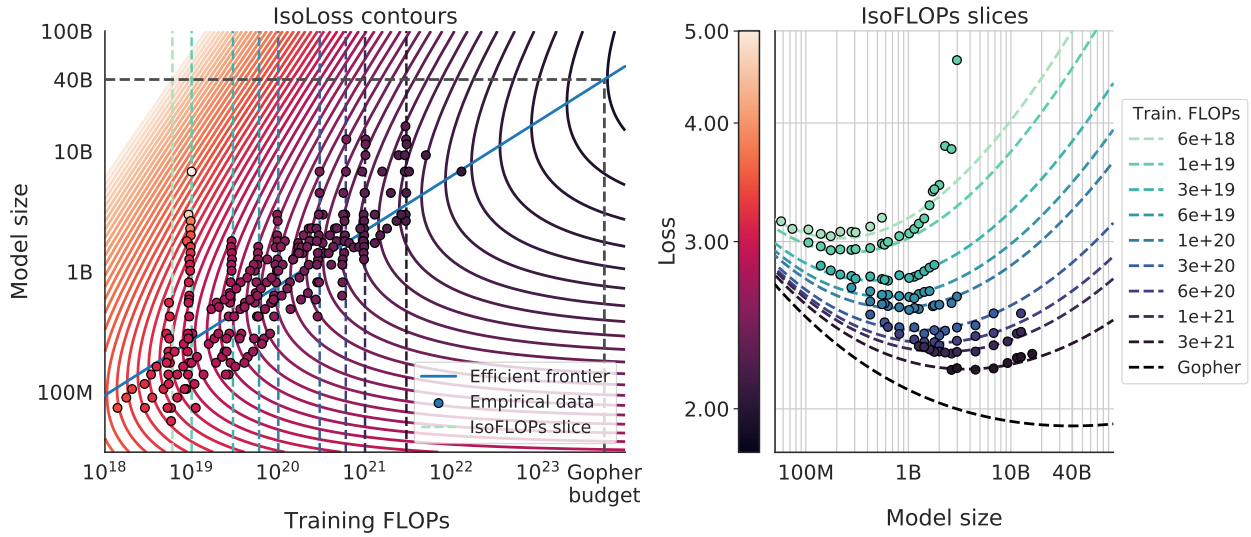


Figure 4 | **Parametric fit.** We fit a parametric modelling of the loss $\hat{L}(N, D)$ and display contour (**left**) and isoFLOP slices (**right**). For each isoFLOP slice, we include a corresponding dashed line in the left plot. In the left plot, we show the efficient frontier in blue, which is a line in log-log space. Specifically, the curve goes through each iso-loss contour at the point with the fewest FLOPs. We project the optimal model size given the *Gopher* FLOP budget to be 40B parameters.

Efficient frontier. We can approximate the functions N_{opt} and D_{opt} by minimizing the parametric loss $\hat{L}$ under the constraint FLOPs$(N, D) \approx 6ND$ (Kaplan et al., 2020). The resulting N_{opt} and D_{opt} balance the two terms in Equation (3) that depend on model size and data. By construction, they have a power-law form:

$$N_{opt}(C) = G\left(\frac{C}{6}\right)^{a}, \quad D_{opt}(C) = G^{-1}\left(\frac{C}{6}\right)^{b}, \quad \text{where} \quad G = \left(\frac{\alpha A}{\beta B}\right)^{\frac{1}{\alpha+\beta}}, \quad a = \frac{\beta}{\alpha+\beta}, \text{ and } b = \frac{\alpha}{\alpha+\beta}. \tag{4}$$

We show contours of the fitted function $\hat{L}$ in Figure 4 (left), and the closed-form efficient computational frontier in blue. From this approach, we find that $a = 0.46$ and $b = 0.54$—as summarized in Table 2.

3.4. Optimal model scaling

We find that the three approaches, despite using different fitting methodologies and different trained models, yield comparable predictions for the optimal scaling in parameters and tokens with FLOPs (shown in Table 2). All three approaches suggest that as compute budget increases, model size and the amount of training data should be increased in approximately equal proportions. The first and second approaches yield very similar predictions for optimal model sizes, as shown in Figure 1 and Figure A3. The third approach predicts even smaller models being optimal at larger compute budgets. We note that the observed points (L, N, D) for low training FLOPs ($C \leqslant 1e21$) have larger residuals $\|L - \hat{L}(N, D)\|_2^2$ than points with higher computational budgets. The fitted model places increased weight on the points with more FLOPs—automatically considering the low-computational budget points as outliers due to the Huber loss. As a consequence of the empirically observed negative curvature in the frontier $C \to N_{opt}$ (see Appendix E), this results in predicting a lower N_{opt} than the two other approaches.

In Table 3 we show the estimated number of FLOPs and tokens that would ensure that a model of a given size lies on the compute-optimal frontier. Our findings suggests that the current generation of

Table 2 | **Estimated parameter and data scaling with increased training compute.** The listed values are the exponents, a and b, on the relationship $N_{opt} \propto C^a$ and $D_{opt} \propto C^b$. Our analysis suggests a near equal scaling in parameters and data with increasing compute which is in clear contrast to previous work on the scaling of large models. The 10^{th} and 90^{th} percentiles are estimated via bootstrapping data (80% of the dataset is sampled 100 times) and are shown in parenthesis.

Approach	Coeff. a where $N_{opt} \propto C^a$	Coeff. b where $D_{opt} \propto C^b$
1. Minimum over training curves	0.50 (0.488, 0.502)	0.50 (0.501, 0.512)
2. IsoFLOP profiles	0.49 (0.462, 0.534)	0.51 (0.483, 0.529)
3. Parametric modelling of the loss	0.46 (0.454, 0.455)	0.54 (0.542, 0.543)
Kaplan et al. (2020)	0.73	0.27

Table 3 | **Estimated optimal training FLOPs and training tokens for various model sizes.** For various model sizes, we show the projections from Approach 1 of how many FLOPs and training tokens would be needed to train compute-optimal models. The estimates for Approach 2 & 3 are similar (shown in Section D.3)

Parameters	FLOPs	FLOPs (in *Gopher* unit)	Tokens
400 Million	1.92e+19	1/29,968	8.0 Billion
1 Billion	1.21e+20	1/4,761	20.2 Billion
10 Billion	1.23e+22	1/46	205.1 Billion
67 Billion	5.76e+23	1	1.5 Trillion
175 Billion	3.85e+24	6.7	3.7 Trillion
280 Billion	9.90e+24	17.2	5.9 Trillion
520 Billion	3.43e+25	59.5	11.0 Trillion
1 Trillion	1.27e+26	221.3	21.2 Trillion
10 Trillion	1.30e+28	22515.9	216.2 Trillion

large language models are considerably over-sized, given their respective compute budgets, as shown in Figure 1. For example, we find that a 175 billion parameter model should be trained with a compute budget of 4.41×10^{24} FLOPs and on over 4.2 trillion tokens. A 280 billion *Gopher*-like model is the optimal model to train given a compute budget of approximately 10^{25} FLOPs and should be trained on 6.8 trillion tokens. Unless one has a compute budget of 10^{26} FLOPs (over 250× the compute used to train *Gopher*), a 1 trillion parameter model is unlikely to be the optimal model to train. Furthermore, the amount of training data that is projected to be needed is far beyond what is currently used to train large models, and underscores the importance of dataset collection in addition to engineering improvements that allow for model scale. While there is significant uncertainty extrapolating out many orders of magnitude, our analysis clearly suggests that given the training compute budget for many current LLMs, smaller models should have been trained on more tokens to achieve the most performant model.

In Appendix C, we reproduce the IsoFLOP analysis on two additional datasets: C4 (Raffel et al., 2020a) and GitHub code (Rae et al., 2021). In both cases we reach the similar conclusion that model size and number of training tokens should be scaled in equal proportions.

4. *Chinchilla*

Based on our analysis in Section 3, the optimal model size for the *Gopher* compute budget is somewhere between 40 and 70 billion parameters. We test this hypothesis by training a model on the larger end of this range—70B parameters—for 1.4T tokens, due to both dataset and computational efficiency considerations. In this section we compare this model, which we call *Chinchilla*, to *Gopher* and other LLMs. Both *Chinchilla* and *Gopher* have been trained for the same number of FLOPs but differ in the size of the model and the number of training tokens.

While pre-training a large language model has a considerable compute cost, downstream fine-tuning and inference also make up substantial compute usage (Rae et al., 2021). Due to being 4× smaller than *Gopher*, both the memory footprint and inference cost of *Chinchilla* are also smaller.

4.1. Model and training details

The full set of hyperparameters used to train *Chinchilla* are given in Table 4. *Chinchilla* uses the same model architecture and training setup as *Gopher* with the exception of the differences listed below.

- We train *Chinchilla* on *MassiveText* (the same dataset as *Gopher*) but use a slightly different subset distribution (shown in Table A1) to account for the increased number of training tokens.
- We use AdamW (Loshchilov and Hutter, 2019) for *Chinchilla* rather than Adam (Kingma and Ba, 2014) as this improves the language modelling loss and the downstream task performance after finetuning.[8]
- We train *Chinchilla* with a slightly modified SentencePiece (Kudo and Richardson, 2018) tokenizer that does not apply NFKC normalisation. The vocabulary is very similar– 94.15% of tokens are the same as those used for training *Gopher*. We find that this particularly helps with the representation of mathematics and chemistry, for example.
- Whilst the forward and backward pass are computed in `bfloat16`, we store a `float32` copy of the weights in the distributed optimiser state (Rajbhandari et al., 2020). See *Lessons Learned* from Rae et al. (2021) for additional details.

In Appendix G we show the impact of the various optimiser related changes between *Chinchilla* and *Gopher*. All models in this analysis have been trained on TPUv3/TPUv4 (Jouppi et al., 2017) with JAX (Bradbury et al., 2018) and Haiku (Hennigan et al., 2020). We include a *Chinchilla* model card (Mitchell et al., 2019) in Table A8.

Model	Layers	Number Heads	Key/Value Size	d_{model}	Max LR	Batch Size
Gopher 280B	80	128	128	16,384	4×10^{-5}	3M → 6M
Chinchilla 70B	80	64	128	8,192	1×10^{-4}	1.5M → 3M

Table 4 | ***Chinchilla* architecture details.** We list the number of layers, the key/value size, the bottleneck activation size d_{model}, the maximum learning rate, and the training batch size (# tokens). The feed-forward size is always set to $4 \times d_{model}$. Note that we double the batch size midway through training for both *Chinchilla* and *Gopher*.

[8] Interestingly, a model trained with AdamW only passes the training performance of a model trained with Adam around 80% of the way through the cosine cycle, though the ending performance is notably better– see Figure A7

	# Tasks	Examples
Language Modelling	20	WikiText-103, The Pile: PG-19, arXiv, FreeLaw, ...
Reading Comprehension	3	RACE-m, RACE-h, LAMBADA
Question Answering	3	Natural Questions, TriviaQA, TruthfulQA
Common Sense	5	HellaSwag, Winogrande, PIQA, SIQA, BoolQ
MMLU	57	High School Chemistry, Astronomy, Clinical Knowledge, ...
BIG-bench	62	Causal Judgement, Epistemic Reasoning, Temporal Sequences, ...

Table 5 | **All evaluation tasks.** We evaluate *Chinchilla* on a collection of language modelling along with downstream tasks. We evaluate on largely the same tasks as in Rae et al. (2021), to allow for direct comparison.

4.2. Results

We perform an extensive evaluation of *Chinchilla*, comparing against various large language models. We evaluate on a large subset of the tasks presented in Rae et al. (2021), shown in Table 5. As the focus of this work is on optimal model scaling, we included a large representative subset, and introduce a few new evaluations to allow for better comparison to other existing large models. The evaluation details for all tasks are the same as described in Rae et al. (2021).

4.2.1. Language modelling

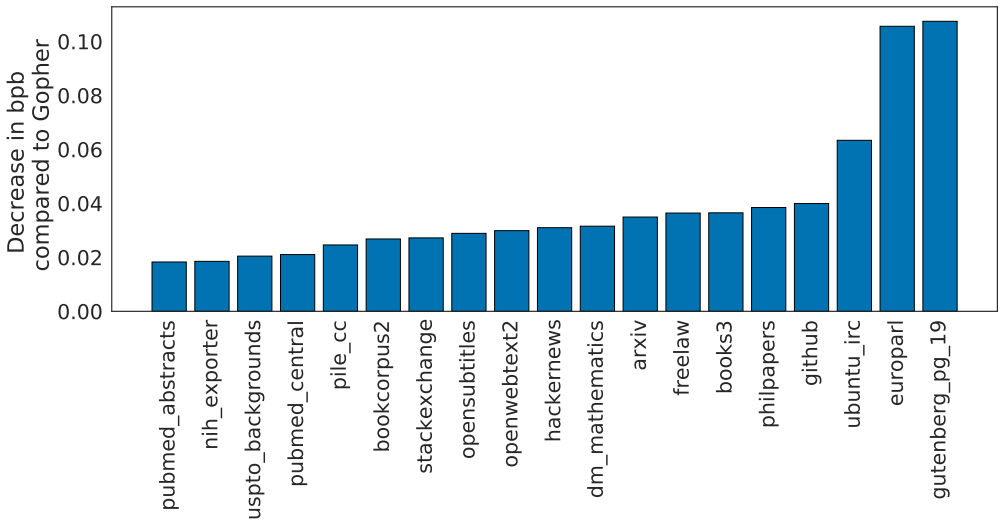


Figure 5 | **Pile Evaluation.** For the different evaluation sets in The Pile (Gao et al., 2020), we show the bits-per-byte (bpb) improvement (decrease) of *Chinchilla* compared to *Gopher*. On all subsets, *Chinchilla* outperforms *Gopher*.

Chinchilla significantly outperforms *Gopher* on all evaluation subsets of The Pile (Gao et al., 2020), as shown in Figure 5. Compared to Jurassic-1 (178B) Lieber et al. (2021), *Chinchilla* is more performant on all but two subsets– `dm_mathematics` and `ubuntu_irc`– see Table A5 for a raw bits-per-byte comparison. On Wikitext103 (Merity et al., 2017), *Chinchilla* achieves a perplexity of 7.16 compared to 7.75 for *Gopher*. Some caution is needed when comparing *Chinchilla* with *Gopher* on these language modelling benchmarks as *Chinchilla* is trained on 4× more data than *Gopher* and thus train/test set leakage may artificially enhance the results. We thus place more emphasis on other

Random	25.0%
Average human rater	34.5%
GPT-3 5-shot	43.9%
Gopher 5-shot	60.0%
***Chinchilla* 5-shot**	**67.6%**
Average human expert performance	*89.8%*
June 2022 Forecast	57.1%
June 2023 Forecast	63.4%

Table 6 | **Massive Multitask Language Understanding (MMLU).** We report the average 5-shot accuracy over 57 tasks with model and human accuracy comparisons taken from Hendrycks et al. (2020). We also include the average prediction for state of the art accuracy in June 2022/2023 made by 73 competitive human forecasters in Steinhardt (2021).

tasks for which leakage is less of a concern, such as MMLU (Hendrycks et al., 2020) and BIG-bench (BIG-bench collaboration, 2021) along with various closed-book question answering and common sense analyses.

4.2.2. MMLU

The Massive Multitask Language Understanding (MMLU) benchmark (Hendrycks et al., 2020) consists of a range of exam-like questions on academic subjects. In Table 6, we report *Chinchilla*'s average 5-shot performance on MMLU (the full breakdown of results is shown in Table A6). On this benchmark, *Chinchilla* significantly outperforms *Gopher* despite being much smaller, with an average accuracy of 67.6% (improving upon *Gopher* by 7.6%). Remarkably, *Chinchilla* even outperforms the expert forecast for June 2023 of 63.4% accuracy (see Table 6) (Steinhardt, 2021). Furthermore, *Chinchilla* achieves greater than 90% accuracy on 4 different individual tasks– `high_school_gov_and_politics`, `international_law`, `sociology`, and `us_foreign_policy`. To our knowledge, no other model has achieved greater than 90% accuracy on a subset.

In Figure 6, we show a comparison to *Gopher* broken down by task. Overall, we find that *Chinchilla* improves performance on the vast majority of tasks. On four tasks (`college_mathematics`, `econometrics`, `moral_scenarios`, and `formal_logic`) *Chinchilla* underperforms *Gopher*, and there is no change in performance on two tasks.

4.2.3. Reading comprehension

On the final word prediction dataset LAMBADA (Paperno et al., 2016), *Chinchilla* achieves 77.4% accuracy, compared to 74.5% accuracy from *Gopher* and 76.6% from MT-NLG 530B (see Table 7). On RACE-h and RACE-m (Lai et al., 2017), *Chinchilla* greatly outperforms *Gopher*, improving accuracy by more than 10% in both cases—see Table 7.

4.2.4. BIG-bench

We analysed *Chinchilla* on the same set of BIG-bench tasks (BIG-bench collaboration, 2021) reported in Rae et al. (2021). Similar to what we observed in MMLU, *Chinchilla* outperforms *Gopher* on the vast majority of tasks (see Figure 7). We find that *Chinchilla* improves the average performance by 10.7%, reaching an accuracy of 65.1% versus 54.4% for *Gopher*. Of the 62 tasks we consider, *Chinchilla* performs worse than *Gopher* on only four—`crash_blossom`, `dark_humor_detection`,

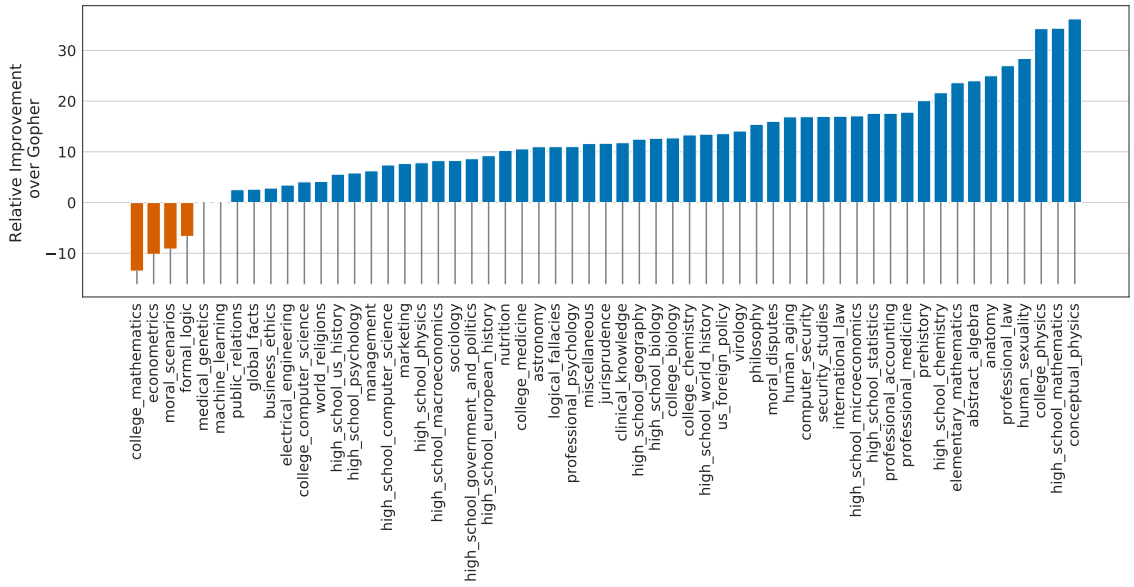


Figure 6 | **MMLU results compared to *Gopher*** We find that *Chinchilla* outperforms *Gopher* by 7.6% on average (see Table 6) in addition to performing better on 51/57 individual tasks, the same on 2/57, and worse on only 4/57 tasks.

	Chinchilla	*Gopher*	GPT-3	MT-NLG 530B
LAMBADA Zero-Shot	**77.4**	74.5	76.2	76.6
RACE-m Few-Shot	**86.8**	75.1	58.1	-
RACE-h Few-Shot	**82.3**	71.6	46.8	47.9

Table 7 | **Reading comprehension.** On RACE-h and RACE-m (Lai et al., 2017), *Chinchilla* considerably improves performance over *Gopher*. Note that GPT-3 and MT-NLG 530B use a different prompt format than we do on RACE-h/m, so results are not comparable to *Gopher* and *Chinchilla*. On LAMBADA (Paperno et al., 2016), *Chinchilla* outperforms both *Gopher* and MT-NLG 530B.

`mathematical_induction` and `logical_args`. Full accuracy results for *Chinchilla* can be found in Table A7.

4.2.5. Common sense

We evaluate *Chinchilla* on various common sense benchmarks: PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), Winogrande (Sakaguchi et al., 2020), HellaSwag (Zellers et al., 2019), and BoolQ (Clark et al., 2019). We find that *Chinchilla* outperforms both *Gopher* and GPT-3 on all tasks and outperforms MT-NLG 530B on all but one task—see Table 8.

On TruthfulQA (Lin et al., 2021), *Chinchilla* reaches 43.6%, 58.5%, and 66.7% accuracy with 0-shot, 5-shot, and 10-shot respectively. In comparison, *Gopher* achieved only 29.5% 0-shot and 43.7% 10-shot accuracy. In stark contrast with the findings of Lin et al. (2021), the large improvements (14.1% in 0-shot accuracy) achieved by Chinchilla suggest that better modelling of the pre-training data alone can lead to substantial improvements on this benchmark.

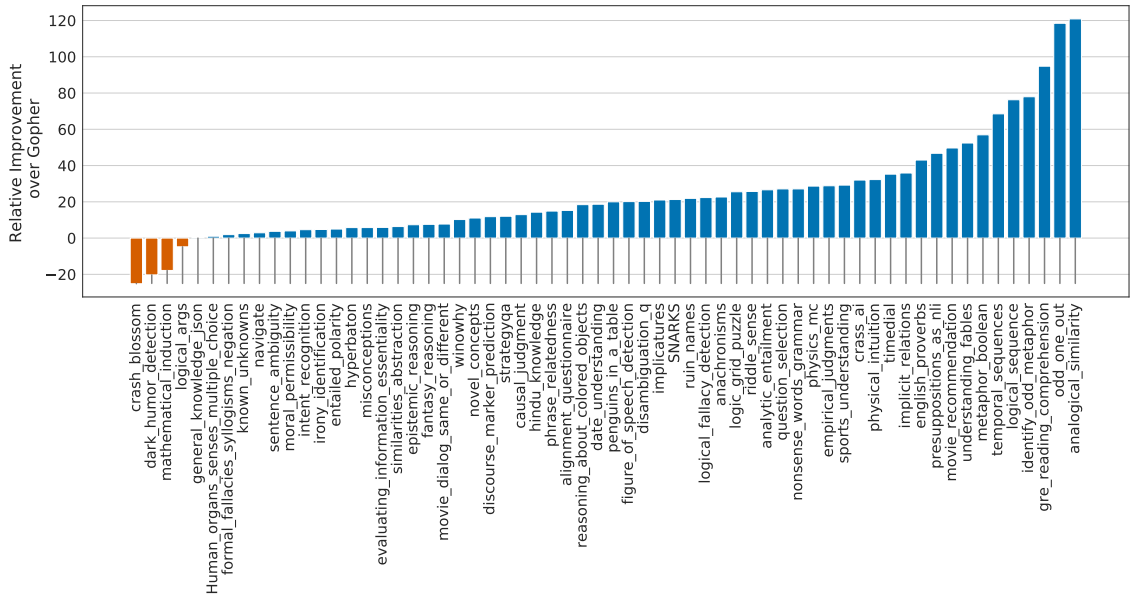


Figure 7 | **BIG-bench results compared to *Gopher*** *Chinchilla* out performs *Gopher* on all but four BIG-bench tasks considered. Full results are in Table A7.

4.2.6. Closed-book question answering

Results on closed-book question answering benchmarks are reported in Table 9. On the Natural Questions dataset (Kwiatkowski et al., 2019), *Chinchilla* achieves new closed-book SOTA accuracies: 31.5% 5-shot and 35.5% 64-shot, compared to 21% and 28% respectively, for *Gopher*. On TriviaQA (Joshi et al., 2017) we show results for both the filtered (previously used in retrieval and open-book work) and unfiltered set (previously used in large language model evaluations). In both cases, *Chinchilla* substantially out performs *Gopher*. On the filtered version, Chinchilla lags behind the open book SOTA (Izacard and Grave, 2020) by only 7.9%. On the unfiltered set, *Chinchilla* outperforms GPT-3—see Table 9.

4.2.7. Gender bias and toxicity

Large Language Models carry potential risks such as outputting offensive language, propagating social biases, and leaking private information (Bender et al., 2021; Weidinger et al., 2021). We expect *Chinchilla* to carry risks similar to *Gopher* because *Chinchilla* is trained on the same data,

	Chinchilla	*Gopher*	GPT-3	MT-NLG 530B	Supervised SOTA
HellaSWAG	**80.8%**	79.2%	78.9%	80.2%	93.9%
PIQA	81.8%	81.8%	81.0%	**82.0%**	90.1%
Winogrande	**74.9%**	70.1%	70.2%	73.0%	91.3%
SIQA	**51.3%**	50.6%	-	-	83.2%
BoolQ	**83.7%**	79.3%	60.5%	78.2%	91.4%

Table 8 | **Zero-shot comparison on Common Sense benchmarks.** We show a comparison between *Chinchilla*, *Gopher*, and MT-NLG 530B on various Common Sense benchmarks. We see that *Chinchilla* matches or outperforms *Gopher* and GPT-3 on all tasks. On all but one *Chinchilla* outperforms the much larger MT-NLG 530B model.

	Method	*Chinchilla*	*Gopher*	GPT-3	SOTA (open book)
Natural Questions (dev)	0-shot	16.6%	10.1%	14.6%	54.4%
	5-shot	31.5%	24.5%	-	
	64-shot	35.5%	28.2%	29.9%	
TriviaQA (unfiltered, test)	0-shot	67.0%	52.8%	64.3 %	-
	5-shot	73.2%	63.6%	-	
	64-shot	72.3%	61.3%	71.2%	
TriviaQA (filtered, dev)	0-shot	55.4%	43.5%	-	72.5%
	5-shot	64.1%	57.0%	-	
	64-shot	64.6%	57.2%	-	

Table 9 | **Closed-book question answering.** For Natural Questions (Kwiatkowski et al., 2019) and TriviaQA (Joshi et al., 2017), *Chinchilla* outperforms *Gopher* in all cases. On Natural Questions, *Chinchilla* outperforms GPT-3. On TriviaQA we show results on two different evaluation sets to allow for comparison to GPT-3 and to open book SOTA (FiD + Distillation (Izacard and Grave, 2020)).

albeit with slightly different relative weights, and because it has a similar architecture. Here, we examine gender bias (particularly gender and occupation bias) and generation of toxic language. We select a few common evaluations to highlight potential issues, but stress that our evaluations are not comprehensive and much work remains to understand, evaluate, and mitigate risks in LLMs.

Gender bias. As discussed in Rae et al. (2021), large language models reflect contemporary and historical discourse about different groups (such as gender groups) from their training dataset, and we expect the same to be true for *Chinchilla*. Here, we test if potential gender and occupation biases manifest in unfair outcomes on coreference resolutions, using the Winogender dataset (Rudinger et al., 2018) in a zero-shot setting. Winogender tests whether a model can correctly determine if a pronoun refers to different occupation words. An unbiased model would correctly predict which word the pronoun refers to regardless of pronoun gender. We follow the same setup as in Rae et al. (2021) (described further in Section H.3).

As shown in Table 10, *Chinchilla* correctly resolves pronouns more frequently than *Gopher* across all groups. Interestingly, the performance increase is considerably smaller for male pronouns (increase of 3.2%) than for female or neutral pronouns (increases of 8.3% and 9.2% respectively). We also consider *gotcha* examples, in which the correct pronoun resolution contradicts gender stereotypes (determined by labor statistics). Again, we see that *Chinchilla* resolves pronouns more accurately than *Gopher*. When breaking up examples by male/female gender and *gotcha/not gotcha*, the largest improvement is on female *gotcha* examples (improvement of 10%). Thus, though *Chinchilla* uniformly overcomes gender stereotypes for more coreference examples than *Gopher*, the rate of improvement is higher for some pronouns than others, suggesting that the improvements conferred by using a more compute-optimal model can be uneven.

Sample toxicity. Language models are capable of generating toxic language—including insults, hate speech, profanities and threats (Gehman et al., 2020; Rae et al., 2021). While toxicity is an umbrella term, and its evaluation in LMs comes with challenges (Welbl et al., 2021; Xu et al., 2021), automatic classifier scores can provide an indication for the levels of harmful text that a LM generates. Rae et al. (2021) found that improving language modelling loss by increasing the number of model parameters has only a negligible effect on toxic text generation (unprompted); here we analyze

	Chinchilla	*Gopher*
All	78.3%	71.4%
Male	71.2%	68.0%
Female	79.6%	71.3%
Neutral	84.2%	75.0%

	Chinchilla	*Gopher*
Male *gotcha*	62.5%	59.2%
Male *not gotcha*	80.0%	76.7%
Female *gotcha*	76.7%	66.7%
Female *not gotcha*	82.5%	75.8%

Table 10 | **Winogender results. Left:** *Chinchilla* consistently resolves pronouns better than *Gopher*. **Right:** *Chinchilla* performs better on examples which contradict gender stereotypes (*gotcha* examples). However, difference in performance across groups suggests *Chinchilla* exhibits bias.

whether the same holds true for a lower LM loss achieved via more compute-optimal training. Similar to the protocol of Rae et al. (2021), we generate 25,000 unprompted samples from *Chinchilla*, and compare their *PerspectiveAPI* toxicity score distribution to that of *Gopher*-generated samples. Several summary statistics indicate an absence of major differences: the mean (median) toxicity score for *Gopher* is 0.081 (0.064), compared to 0.087 (0.066) for *Chinchilla*, and the 95$^{\text{th}}$ percentile scores are 0.230 for *Gopher*, compared to 0.238 for *Chinchilla*. That is, the large majority of generated samples are classified as non-toxic, and the difference between the models is negligible. In line with prior findings (Rae et al., 2021), this suggests that toxicity levels in unconditional text generation are largely independent of the model quality (measured in language modelling loss), i.e. that better models of the training dataset are not necessarily more toxic.

5. Discussion & Conclusion

The trend so far in large language model training has been to increase the model size, often without increasing the number of training tokens. The largest dense transformer, MT-NLG 530B, is now over 3× larger than GPT-3's 170 billion parameters from just two years ago. However, this model, as well as the majority of existing large models, have all been trained for a comparable number of tokens—around 300 billion. While the desire to train these mega-models has led to substantial engineering innovation, we hypothesize that the race to train larger and larger models is resulting in models that are substantially underperforming compared to what could be achieved with the same compute budget.

We propose three predictive approaches towards optimally setting model size and training duration, based on the outcome of over 400 training runs. All three approaches predict that *Gopher* is substantially over-sized and estimate that for the same compute budget a smaller model trained on more data will perform better. We directly test this hypothesis by training *Chinchilla*, a 70B parameter model, and show that it outperforms *Gopher* and even larger models on nearly every measured evaluation task.

Whilst our method allows us to make predictions on how to scale large models when given additional compute, there are several limitations. Due to the cost of training large models, we only have two comparable training runs at large scale (*Chinchilla* and *Gopher*), and we do not have additional tests at intermediate scales. Furthermore, we assume that the efficient computational frontier can be described by a power-law relationship between the compute budget, model size, and number of training tokens. However, we observe some concavity in $\log(N_{opt})$ at high compute budgets (see Appendix E). This suggests that we may still be overestimating the optimal size of large models. Finally, the training runs for our analysis have all been trained on less than an epoch of data; future work may consider the multiple epoch regime. Despite these limitations, the comparison of *Chinchilla* to *Gopher* validates our performance predictions, that have thus enabled training a better (and more

lightweight) model at the same compute budget.

Though there has been significant recent work allowing larger and larger models to be trained, our analysis suggests an increased focus on dataset scaling is needed. Speculatively, we expect that scaling to larger and larger datasets is only beneficial when the data is high-quality. This calls for responsibly collecting larger datasets with a high focus on dataset quality. Larger datasets will require extra care to ensure train-test set overlap is properly accounted for, both in the language modelling loss but also with downstream tasks. Finally, training for trillions of tokens introduces many ethical and privacy concerns. Large datasets scraped from the web will contain toxic language, biases, and private information. With even larger datasets being used, the quantity (if not the frequency) of such information increases, which makes dataset introspection all the more important. *Chinchilla* does suffer from bias and toxicity but interestingly it seems less affected than *Gopher*. Better understanding how performance of large language models and toxicity interact is an important future research question.

While we have applied our methodology towards the training of auto-regressive language models, we expect that there is a similar trade-off between model size and the amount of data in other modalities. As training large models is very expensive, choosing the optimal model size and training steps beforehand is essential. The methods we propose are easy to reproduce in new settings.

6. Acknowledgements

We'd like to thank Jean-baptiste Alayrac, Kareem Ayoub, Chris Dyer, Nando de Freitas, Demis Hassabis, Geoffrey Irving, Koray Kavukcuoglu, Nate Kushman and Angeliki Lazaridou for useful comments on the manuscript. We'd like to thank Andy Brock, Irina Higgins, Michela Paganini, Francis Song, and other colleagues at DeepMind for helpful discussions. We are also very grateful to the JAX and XLA team for their support and assistance.

References

M. Artetxe, S. Bhosale, N. Goyal, T. Mihaylov, M. Ott, S. Shleifer, X. V. Lin, J. Du, S. Iyer, R. Pasunuru, G. Anantharaman, X. Li, S. Chen, H. Akin, M. Baines, L. Martin, X. Zhou, P. S. Koura, B. O'Horo, J. Wang, L. Zettlemoyer, M. Diab, Z. Kozareva, and V. Stoyanov. Efficient Large Scale Language Modeling with Mixtures of Experts. arXiv:2112.10684, 2021.

E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, pages 610–623, 2021.

BIG-bench collaboration. Beyond the imitation game: Measuring and extrapolating the capabilities of language models. In preparation, 2021. URL https://github.com/google/BIG-bench/.

Y. Bisk, R. Zellers, J. Gao, Y. Choi, et al. PIQA: Reasoning about physical commonsense in natural language. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 7432–7439, 2020.

S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. van den Driessche, J.-B. Lespiau, B. Damoc, A. Clark, D. de Las Casas, A. Guy, J. Menick, R. Ring, T. Hennigan, S. Huang, L. Maggiore, C. Jones, A. Cassirer, A. Brock, M. Paganini, G. Irving, O. Vinyals, S. Osindero, K. Simonyan, J. W. Rae, E. Elsen, and L. Sifre. Improving language models by retrieving from trillions of tokens. arXiv 2112.04426, 2021.

Emergent Abilities of Large Language Models

Jason Wei [1] *jasonwei@google.com*
Yi Tay [1] *yitay@google.com*
Rishi Bommasani [2] *nlprishi@stanford.edu*
Colin Raffel [3] *craffel@gmail.com*
Barret Zoph [1] *barretzoph@google.com*
Sebastian Borgeaud [4] *sborgeaud@deepmind.com*
Dani Yogatama [4] *dyogatama@deepmind.com*
Maarten Bosma [1] *bosma@google.com*
Denny Zhou [1] *dennyzhou@google.com*
Donald Metzler [1] *metzler@google.com*
Ed H. Chi [1] *edchi@google.com*
Tatsunori Hashimoto [2] *thashim@stanford.edu*
Oriol Vinyals [4] *vinyals@deepmind.com*
Percy Liang [2] *pliang@stanford.edu*
Jeff Dean [1] *jeff@google.com*
William Fedus [1] *liamfedus@google.com*

[1] *Google Research* [2] *Stanford University* [3] *UNC Chapel Hill* [4] *DeepMind*

Reviewed on OpenReview: *https://openreview.net/forum?id=yzkSU5zdwD*

Abstract

Scaling up language models has been shown to predictably improve performance and sample efficiency on a wide range of downstream tasks. This paper instead discusses an unpredictable phenomenon that we refer to as *emergent abilities* of large language models. We consider an ability to be emergent if it is not present in smaller models but is present in larger models. Thus, emergent abilities cannot be predicted simply by extrapolating the performance of smaller models. The existence of such emergence raises the question of whether additional scaling could potentially further expand the range of capabilities of language models.

1 Introduction

Language models have revolutionized natural language processing (NLP) in recent years. It is now well-known that increasing the scale of language models (e.g., training compute, model parameters, etc.) can lead to better performance and sample efficiency on a range of downstream NLP tasks (Devlin et al., 2019; Brown et al., 2020, *inter alia*). In many cases, the effect of scale on performance can often be methodologically predicted via scaling laws—for example, scaling curves for cross-entropy loss have been shown to empirically span more than seven orders of magnitude (Kaplan et al., 2020; Hoffmann et al., 2022). On the other hand, performance for certain downstream tasks counterintuitively does not appear to continuously improve as a function of scale, and such tasks cannot be predicted ahead of time (Ganguli et al., 2022).

In this paper, we will discuss the unpredictable phenomena of *emergent abilities* of large language models. Emergence as an idea has been long discussed in domains such as physics, biology, and computer science (Anderson, 1972; Hwang et al., 2012; Forrest, 1990; Corradini & O'Connor, 2010; Harper & Lewis, 2012, *inter*

alia). We will consider the following general definition of emergence, adapted from Steinhardt (2022) and rooted in a 1972 essay called "More Is Different" by Nobel prize-winning physicist Philip Anderson (Anderson, 1972):

> *Emergence is when quantitative changes in a system result in qualitative changes in behavior.*

Here we will explore emergence with respect to model scale, as measured by training compute and number of model parameters. Specifically, we define *emergent abilities of large language models* as abilities that are not present in smaller-scale models but are present in large-scale models; thus they cannot be predicted by simply extrapolating the performance improvements on smaller-scale models (§2).[1] We survey emergent abilities as observed in a range of prior work, categorizing them in settings such as few-shot prompting (§3) and augmented prompting strategies (§4). Emergence motivates future research on why such abilities are acquired and whether more scaling will lead to further emergent abilities, which we highlight as important questions for the field (§5).

2 Emergent Abilities Definition

As a broad concept, emergence is often used informally and can be reasonably interpreted in many different ways. In this paper, we will consider a focused definition of emergent abilities of large language models:

> *An ability is emergent if it is not present in smaller models but is present in larger models.*

Emergent abilities would not have been directly predicted by extrapolating a scaling law (i.e. consistent performance improvements) from small-scale models. When visualized via a scaling curve (x-axis: model scale, y-axis: performance), emergent abilities show a clear pattern—performance is near-random until a certain critical threshold of scale is reached, after which performance increases to substantially above random. This qualitative change is also known as a *phase transition*—a dramatic change in overall behavior that would not have been foreseen by examining smaller-scale systems (Huberman & Hogg, 1987).

Today's language models have been scaled primarily along three factors: amount of computation, number of model parameters, and training dataset size (Kaplan et al., 2020; Hoffmann et al., 2022). In this paper, we will analyze scaling curves by plotting the performance of different models where training compute for each model is measured in FLOPs on the x-axis (Hoffmann et al., 2022). Because language models trained with more compute tend to also have more parameters, we additionally show plots with number of model parameters as the x-axis in Appendix D (see Figure 11 and Figure 12, as well as Figure 4 and Figure 10). Using training FLOPs or model parameters as the x-axis produces curves with similar shapes due to the fact that most dense Transformer language model families have scaled training compute roughly proportionally with model parameters (Kaplan et al., 2020).

Training dataset size is also an important factor, but we do not plot capabilities against it because many language model families use a fixed number of training examples for all model sizes (Brown et al., 2020; Rae et al., 2021; Chowdhery et al., 2022). Although we focus on training computation and model size here, there is not a single proxy that adequately captures all aspects of scale. For example, Chinchilla (Hoffmann et al., 2022) has one-fourth as many parameters as Gopher (Rae et al., 2021) but uses similar training compute; and sparse mixture-of-expert models have more parameters per training/inference compute than dense models (Fedus et al., 2021; Du et al., 2021). Overall, it may be wise to view emergence as a function of many correlated variables. For example, later in Figure 4 we will also plot emergence as a function of WikiText103 perplexity (Merity et al., 2016), which happens to closely correlate with training computation for Gopher/ Chinchilla (though this correlation may not hold in the long-run).

Note that the scale at which an ability is first observed to emerge depends on a number of factors and is not an immutable property of the ability. For instance, emergence may occur with less training compute

[1] This survey focuses on pre-trained Transformer language models. Emergent abilities in NLP more broadly, however, could go back to Miller et al. (2004), Liang (2005), or earlier.

or fewer model parameters for models trained on higher-quality data. Conversely, emergent abilities also crucially depend on other factors such as not being limited by the amount of data, its quality, or the number of parameters in the model. Today's language models are likely not trained optimally (Hoffmann et al., 2022), and our understanding of how to best train models will evolve over time. Our goal in this paper is not to characterize or claim that a specific scale is required to observe emergent abilities, but rather, we aim to discuss examples of emergent behavior in prior work.

3 Few-Shot Prompted Tasks

We first discuss emergent abilities in the *prompting* paradigm, as popularized by GPT-3 (Brown et al., 2020).[2] In prompting, a pre-trained language model is given a prompt (e.g. a natural language instruction) of a task and completes the response without any further training or gradient updates to its parameters. Brown et al. (2020) proposed *few-shot prompting*, which includes a few input-output examples in the model's context (input) as a preamble before asking the model to perform the task for an unseen inference-time example. An example prompt is shown in Figure 1.

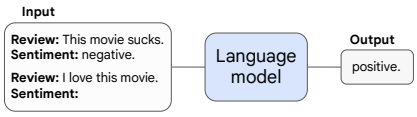


Figure 1: Example of an input and output for few-shot prompting.

The ability to perform a task via few-shot prompting is emergent when a model has random performance until a certain scale, after which performance increases to well-above random. Figure 2 shows eight such emergent abilities spanning five language model families from various work.

BIG-Bench. Figure 2A–D depicts four emergent few-shot prompted tasks from BIG-Bench, a crowd-sourced suite of over 200 benchmarks for language model evaluation (BIG-Bench, 2022). Figure 2A shows an arithmetic benchmark that tests 3-digit addition and subtraction, as well as 2-digit multiplication. GPT-3 and LaMDA (Thoppilan et al., 2022) have close-to-zero performance for several orders of magnitude of training compute, before performance jumps to sharply above random at $2 \cdot 10^{22}$ training FLOPs (13B parameters) for GPT-3, and 10^{23} training FLOPs (68B parameters) for LaMDA. Similar emergent behavior also occurs at around the same model scale for other tasks, such as transliterating from the International Phonetic Alphabet (Figure 2B), recovering a word from its scrambled letters (Figure 2C), and Persian question-answering (Figure 2D). Even more emergent abilities from BIG-Bench are given in Appendix E.

TruthfulQA. Figure 2E shows few-shot prompted performance on the TruthfulQA benchmark, which measures the ability to answer questions truthfully (Lin et al., 2021). This benchmark is adversarially curated against GPT-3 models, which do not perform above random, even when scaled to the largest model size. Small Gopher models also do not perform above random until scaled up to the largest model of $5 \cdot 10^{23}$ training FLOPs (280B parameters), for which performance jumps to more than 20% above random (Rae et al., 2021).

Grounded conceptual mappings. Figure 2F shows the task of grounded conceptual mappings, where language models must learn to map a conceptual domain, such as a cardinal direction, represented in a textual grid world (Patel & Pavlick, 2022). Again, performance only jumps to above random using the largest GPT-3 model.

Multi-task language understanding. Figure 2G shows the Massive Multi-task Language Understanding (MMLU) benchmark, which aggregates 57 tests covering a range of topics including math, history, law, and more (Hendrycks et al., 2021a). For GPT-3, Gopher, and Chinchilla, models of $\sim 10^{22}$ training FLOPs ($\sim$10B parameters) or smaller do not perform better than guessing on average over all the topics, scaling up to 3–5 $\cdot 10^{23}$ training FLOPs (70B–280B parameters) enables performance to substantially surpass random. This result is striking because it could imply that the ability to solve knowledge-based questions spanning a large collection of topics might require scaling up past this threshold (for dense language models without retrieval or access to external memory).

[2]Though GPT-3 popularized prompting, the task setup has existed since before GPT-3 (Trinh & Le, 2018; McCann et al., 2018; Radford et al., 2019; Raffel et al., 2020).

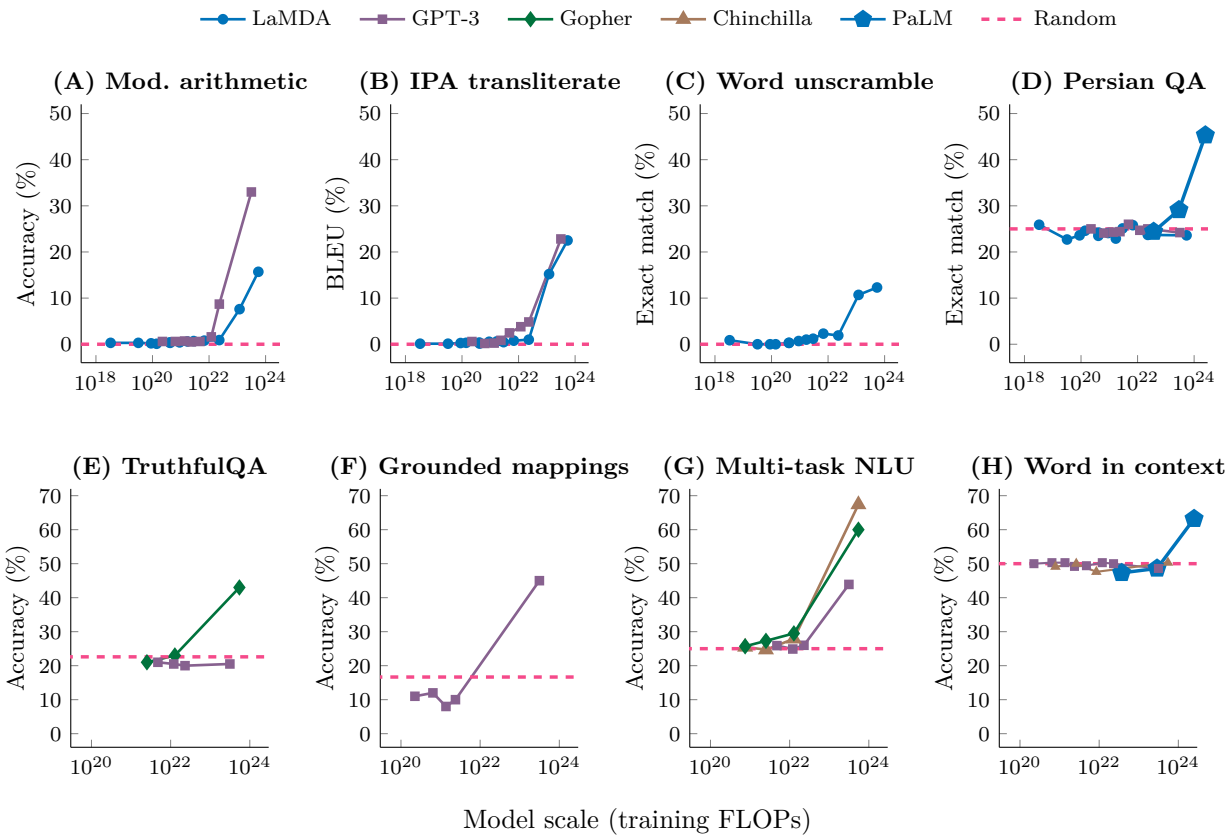


Figure 2: Eight examples of emergence in the few-shot prompting setting. Each point is a separate model. The ability to perform a task via few-shot prompting is emergent when a language model achieves random performance until a certain scale, after which performance significantly increases to well-above random. Note that models that used more training compute also typically have more parameters—hence, we show an analogous figure with number of model parameters instead of training FLOPs as the x-axis in Figure 11. A–D: BIG-Bench (2022), 2-shot. E: Lin et al. (2021) and Rae et al. (2021). F: Patel & Pavlick (2022). G: Hendrycks et al. (2021a), Rae et al. (2021), and Hoffmann et al. (2022). H: Brown et al. (2020), Hoffmann et al. (2022), and Chowdhery et al. (2022) on the WiC benchmark (Pilehvar & Camacho-Collados, 2019).

Word in Context. Finally, Figure 2H shows the Word in Context (WiC) benchmark (Pilehvar & Camacho-Collados, 2019), which is a semantic understanding benchmark. Notably, GPT-3 and Chinchilla fail to achieve one-shot performance of better than random, even when scaled to their largest model size of $\sim 5 \cdot 10^{23}$ FLOPs. Although these results so far may suggest that scaling alone may not enable models to solve WiC, above-random performance eventually emerged when PaLM was scaled to $2.5 \cdot 10^{24}$ FLOPs (540B parameters), which was much larger than GPT-3 and Chinchilla.

4 Augmented Prompting Strategies

Although few-shot prompting is perhaps currently the most common way of interacting with large language models, recent work has proposed several other prompting and finetuning strategies to further augment the abilities of language models. If a technique shows no improvement or is harmful when compared to the baseline of not using the technique until applied to a model of a large-enough scale, we also consider the technique an emergent ability.

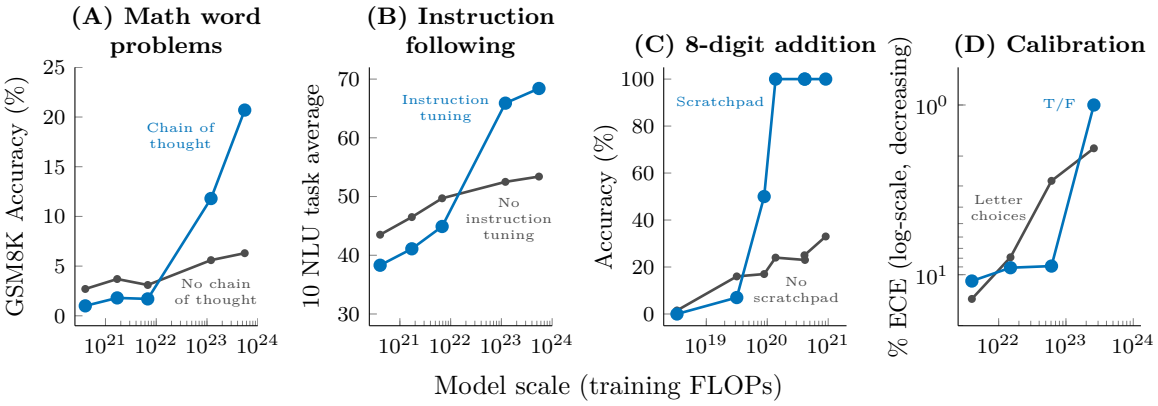


Figure 3: Specialized prompting or finetuning methods can be emergent in that they do not have a positive effect until a certain model scale. A: Wei et al. (2022b). B: Wei et al. (2022a). C: Nye et al. (2021). D: Kadavath et al. (2022). An analogous figure with number of parameters on the x-axis instead of training FLOPs is given in Figure 12. The model shown in A-C is LaMDA (Thoppilan et al., 2022), and the model shown in D is from Anthropic.

Multi-step reasoning. Reasoning tasks, especially those involving multiple steps, have been challenging for language models and NLP models more broadly (Rae et al., 2021; Bommasani et al., 2021; Nye et al., 2021). A recent prompting strategy called chain-of-thought prompting enables language models to solve such problems by guiding them to produce a sequence of intermediate steps before giving the final answer (Cobbe et al., 2021; Wei et al., 2022b; Suzgun et al., 2022). As shown in Figure 3A, chain of thought prompting only surpasses standard prompting without intermediate steps when scaled to 10^{23} training FLOPs (∼100B parameters). A similar emergence in performance gain was also observed when augmenting few-shot prompting with explanations that came after the final answer (Lampinen et al., 2022).

Instruction following. Another growing line of work aims to better enable language models to perform new tasks simply by reading instructions describing the task (without few-shot exemplars). By finetuning on a mixture of tasks phrased as instructions, language models have been shown to respond appropriately to instructions describing an unseen task (Ouyang et al., 2022; Wei et al., 2022a; Sanh et al., 2022; Chung et al., 2022). As shown in Figure 3B, Wei et al. (2022a) found that this instruction-finetuning technique hurts performance for models of $7 \cdot 10^{21}$ training FLOPs (8B parameters) or smaller, and only improves performance when scaled to 10^{23} training FLOPs (∼100B parameters) (though Sanh et al. (2022) found shortly after that this instruction-following behavior could be also induced by finetuning smaller encoder-decoder T5 models).

Program execution. Consider computational tasks involving multiple steps, such as adding large numbers or executing computer programs. Nye et al. (2021) show that finetuning language models to predict intermediate outputs ("scratchpad") enables them to successfully execute such multi-step computations. As shown in Figure 3C, on 8-digit addition, using a scratchpad only helps for models of $\sim 9 \cdot 10^{19}$ training FLOPs (40M parameters) or larger.

Model calibration. Finally, an important direction for deployment of language models studies is *calibration*, which measures whether models can predict which questions they will be able to answer correctly. Kadavath et al. (2022) compared two ways of measuring calibration: a True/False technique, where models first propose answers and then evaluate the probability "P(True)" that their answers are correct, and more-standard methods of calibration, which use the probability of the correct answer compared with other answer options. As shown in Figure 3D, the superiority of the True/False technique only emerges when scaled to the largest model scale of $\sim 3 \cdot 10^{23}$ training FLOPs (52B parameters).

Table 1: List of emergent abilities of large language models and the scale (both training FLOPs and number of model parameters) at which the abilities emerge.

	Emergent scale			
	Train. FLOPs	Params.	Model	Reference
Few-shot prompting abilities				
• Addition/subtraction (3 digit)	2.3E+22	13B	GPT-3	Brown et al. (2020)
• Addition/subtraction (4-5 digit)	3.1E+23	175B		
• MMLU Benchmark (57 topic avg.)	3.1E+23	175B	GPT-3	Hendrycks et al. (2021a)
• Toxicity classification (CivilComments)	1.3E+22	7.1B	Gopher	Rae et al. (2021)
• Truthfulness (Truthful QA)	5.0E+23	280B		
• MMLU Benchmark (26 topics)	5.0E+23	280B		
• Grounded conceptual mappings	3.1E+23	175B	GPT-3	Patel & Pavlick (2022)
• MMLU Benchmark (30 topics)	5.0E+23	70B	Chinchilla	Hoffmann et al. (2022)
• Word in Context (WiC) benchmark	2.5E+24	540B	PaLM	Chowdhery et al. (2022)
• Many BIG-Bench tasks (see Appendix E)	Many	Many	Many	BIG-Bench (2022)
Augmented prompting abilities				
• Instruction following (finetuning)	1.3E+23	68B	FLAN	Wei et al. (2022a)
• Scratchpad: 8-digit addition (finetuning)	8.9E+19	40M	LaMDA	Nye et al. (2021)
• Using open-book knowledge for fact checking	1.3E+22	7.1B	Gopher	Rae et al. (2021)
• Chain-of-thought: Math word problems	1.3E+23	68B	LaMDA	Wei et al. (2022b)
• Chain-of-thought: StrategyQA	2.9E+23	62B	PaLM	Chowdhery et al. (2022)
• Differentiable search index	3.3E+22	11B	T5	Tay et al. (2022b)
• Self-consistency decoding	1.3E+23	68B	LaMDA	Wang et al. (2022b)
• Leveraging explanations in prompting	5.0E+23	280B	Gopher	Lampinen et al. (2022)
• Least-to-most prompting	3.1E+23	175B	GPT-3	Zhou et al. (2022)
• Zero-shot chain-of-thought reasoning	3.1E+23	175B	GPT-3	Kojima et al. (2022)
• Calibration via P(True)	2.6E+23	52B	Anthropic	Kadavath et al. (2022)
• Multilingual chain-of-thought reasoning	2.9E+23	62B	PaLM	Shi et al. (2022)
• Ask me anything prompting	1.4E+22	6B	EleutherAI	Arora et al. (2022)

5 Discussion

We have seen that a range of abilities—in the few-shot prompting setup or otherwise—have thus far only been observed when evaluated on a sufficiently large language model. Hence, their emergence cannot be predicted by simply extrapolating performance on smaller-scale models. Emergent few-shot prompted tasks are also unpredictable in the sense that these tasks are not explicitly included in pre-training, and we likely do not know the full scope of few-shot prompted tasks that language models can perform. This raises the question of whether further scaling could potentially endow even-larger language models with new emergent abilities. Tasks that language models cannot currently do are prime candidates for future emergence; for instance, there are dozens of tasks in BIG-Bench for which even the largest GPT-3 and PaLM models do not achieve above-random performance (see Appendix E.4).

The ability for scale to unpredictably enable new techniques is not just theoretical. Consider the Word in Context (WiC) benchmark (Pilehvar & Camacho-Collados, 2019) shown in Figure 2H, as a historical example. Here, scaling GPT-3 to around $3 \cdot 10^{23}$ training FLOPs (175B parameters) failed to unlock above-random one-shot prompting performance.[3] Regarding this negative result, Brown et al. (2020) cited the model architecture of GPT-3 or the use of an autoregressive language modeling objective (rather than using a denoising training objective) as potential reasons, and suggested training a model of comparable size with bidirectional architecture as a remedy. However, later work found that further scaling a decoder-only language model was actually enough to enable above-random performance on this task. As is shown in Figure 2H, scaling PaLM (Chowdhery et al., 2022) from $3 \cdot 10^{23}$ training FLOPs (62B parameters) to $3 \cdot 10^{24}$ training

[3]GPT-3 does achieve slightly above-random performance on the dev set with few-shot instead of one-shot prompting (∼55%), but this above-random performance did not appear to be a result of scale and did not hold on the test set server.

FLOPs (540B parameters) led to a significant jump in performance, without the significant architectural changes suggested by Brown et al. (2020).

5.1 Potential explanations of emergence

Although there are dozens of examples of emergent abilities, there are currently few compelling explanations for why such abilities emerge in the way they do. For certain tasks, there may be natural intuitions for why emergence requires a model larger than a particular threshold scale. For instance, if a multi-step reasoning task requires l steps of sequential computation, this might require a model with a depth of at least $O(l)$ layers. It is also reasonable to assume that more parameters and more training enable better memorization that could be helpful for tasks requiring world knowledge.[4] As an example, good performance on closed-book question-answering may require a model with enough parameters to capture the compressed knowledge base itself (though language model-based compressors can have higher compression ratios than conventional compressors (Bellard, 2021)).

It is also important to consider the evaluation metrics used to measure emergent abilities (BIG-Bench, 2022). For instance, using exact string match as the evaluation metric for long-sequence targets may disguise compounding incremental improvements as emergence. Similar logic may apply for multi-step or arithmetic reasoning problems, where models are only scored on whether they get the final answer to a multi-step problem correct, without any credit given to partially correct solutions. However, the jump in final answer accuracy does not explain why the quality of intermediate steps suddenly emerges to above random, and using evaluation metrics that do not give partial credit are at best an incomplete explanation, because emergent abilities are still observed on many classification tasks (e.g., the tasks in Figure 2D–H).

As an alternative evaluation, we measure cross-entropy loss, which is used in scaling laws for pre-training, for the six emergent BIG-Bench tasks, as detailed in Appendix A. This analysis follows the same experimental setup from BIG-Bench (2022) and affirms their conclusions for the six emergent tasks we consider. Namely, cross-entropy loss improves even for small model scales where the downstream metrics (exact match, BLEU, and accuracy) are close to random and do not improve, which shows that improvements in the log-likelihood of the target sequence can be masked by such downstream metrics. However, this analysis does not explain why downstream metrics are emergent or enable us to predict the scale at which emergence occurs. Overall, more work is needed to tease apart what enables scale to unlock emergent abilities.

5.2 Beyond scaling

Although we may observe an emergent ability to occur at a certain scale, it is possible that the ability could be later achieved at a smaller scale—in other words, model scale is not the singular factor for unlocking an emergent ability. As the science of training large language models progresses, certain abilities may be unlocked for smaller models with new architectures, higher-quality data, or improved training procedures. For example, there are 14 BIG-Bench tasks[5] for which LaMDA 137B and GPT-3 175B models perform at near-random, but PaLM 62B in fact achieves above-random performance, despite having fewer model parameters and training FLOPs. While there is not an empirical study ablating every difference between PaLM 62B and prior models (the computational cost would be too high), potential reasons for the better performance of PaLM could include high-quality training data (e.g., more multilingual and code data than LaMDA) and architectural differences (e.g., split digit-encodings; see Section 2 in Chowdhery et al. (2022)). Another potentially way of unlocking emergence is through a different pre-training objective—it was shown in Tay et al. (2022c) that a computationally-efficient continued pre-training stage on a mixture-of-denoisers objective (Tay et al., 2022a) enabled emergent performance on several BIG-Bench tasks.

Moreover, once an ability is discovered, further research may make the ability available for smaller scale models. Consider the nascent direction of enabling language models to follow natural language instructions describing a task (Wei et al., 2022a; Sanh et al., 2022; Ouyang et al., 2022, *inter alia*). Although Wei et al. (2022a) initially found that instruction-based finetuning only worked for 68B parameter or larger decoder-only

[4] Though note that encoding world knowledge in parameters is just one approach; there are others (e.g., Guu et al., 2020; Borgeaud et al., 2021).

[5] These tasks are enumerated in Appendix F.

models, Sanh et al. (2022) induced similar behavior in a 11B model with an encoder-decoder architecture, which typically has higher performance after finetuning than decoder-only architectures (Wang et al., 2022a). As another example, Ouyang et al. (2022) proposed a finetuning and reinforcement learning from human feedback approach for the InstructGPT models, which enabled a 1.3B model to outperform much larger models in human-rater evaluations on a broad set of use cases.

There has also been work on improving the general few-shot prompting abilities of language models (Gao et al., 2021; Schick & Schütze, 2021, *inter alia*). Theoretical and interpretability research (Wei et al., 2021a; Saunshi et al., 2021) on why a language modeling objective facilitates certain downstream behavior could in turn have implications on how to enable emergence beyond simply scaling. For instance, certain features of pre-training data (e.g., long-range coherence, having many rare classes) have also been shown to correlate with emergent few-shot prompting and could potentially enable it in smaller models (Xie et al., 2022; Chan et al., 2022), and few-shot learning can require certain model architectures in some scenarios (Chan et al., 2022). Computational linguistics work has further shown how threshold frequencies of training data can activate emergent syntactic rule-learning when model parameters and training FLOPs are held constant (Wei et al., 2021b), which has even been shown to have striking "aha" moments similar to those in the psycholinguistics literature (Abend et al., 2017; Zhang et al., 2021). As we continue to train language models, lowering the scale threshold for emergent abilities will become more important for making research on such abilities to available to the community more broadly (Bommasani et al., 2021; Ganguli et al., 2022; Liang et al., 2022).

Naturally, there are limitations to a program consisting only of increasing scale (training compute, model parameters, and dataset size). For instance, scaling may eventually be bottle-necked by hardware constraints, and some abilities may not have emerged at this point. Other abilities may never emerge—for instance, tasks that are far out of the distribution of even a very large training dataset might not ever achieve any significant performance. Finally, an ability could emerge and then plateau; in other words, there is no guarantee that scaling enables an ability to reach the desired level.

5.3 Another view of emergence

While scale (e.g., training FLOPs or model parameters) has been highly correlated with language model performance on many downstream metrics so far, scale need not be the only lens to view emergent abilities. For example, the emergence of task-specific abilities can be analyzed as a function of the language model's perplexity on a general text corpus such as WikiText103 (Merity et al., 2016). Figure 4 shows such a plot with WikiText103 perplexity of the language model on the x-axis and performance on the MMLU benchmark on the y-axis, side-by-side with plots of training FLOPs and model parameters on the x-axis.

Because WikiText103 perplexity and training FLOPs happen to be highly correlated for the models considered here (Gopher and Chinchilla), the plots of emergent abilities look similar for both. However, this correlation between WikiText103 perplexity and scale may not hold in the future as new techniques beyond vanilla dense Transformer models are developed (e.g., retrieval-augmented models may have strong WikiText103 perplexity with less training compute and fewer model parameters (Borgeaud et al., 2021)). Also note that using WikiText103 perplexity to compare across model families can be complicated due to factors such as differences in training data composition. Overall, emergent abilities should probably be viewed as a function of many correlated variables.

5.4 Emergent risks

Importantly, similar to how emergent abilities have been observed in the few-shot prompting setting without explicitly being included in pre-training, risks could also emerge (Bommasani et al., 2021; Steinhardt, 2021; Ganguli et al., 2022). For instance, societal risks of large language models such as truthfulness, bias, and toxicity are a growing area of research (Weidinger et al., 2021). Such risks are important considerations whether or not they can be precisely characterized as "emergent" based on the definition in §2, and, in some scenarios, do increase with model scale (see the Inverse Scaling Prize[6]). Since work on emergent abilities

[6] https://github.com/inverse-scaling/prize

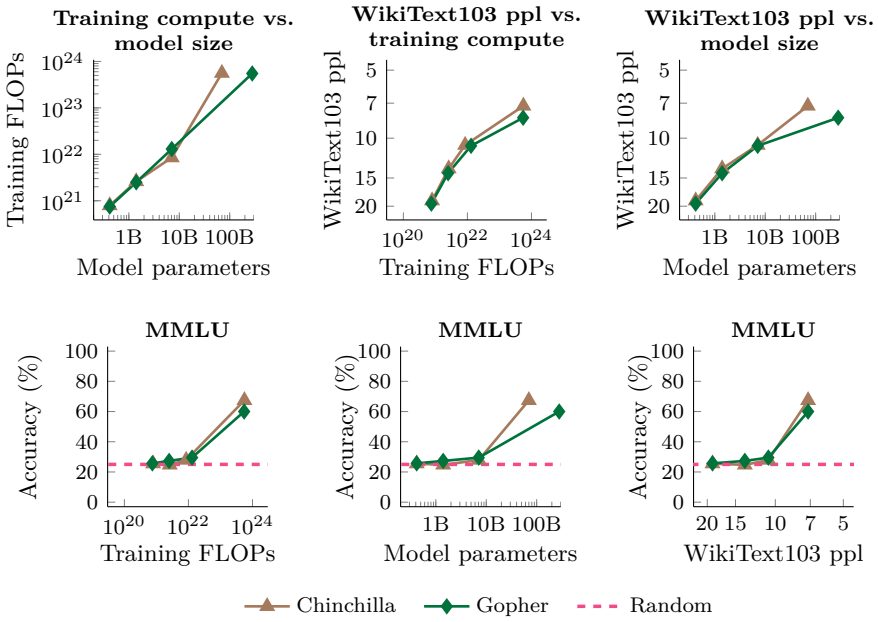


Figure 4: Top row: the relationships between training FLOPs, model parameters, and perplexity (ppl) on WikiText103 (Merity et al., 2016) for Chinchilla and Gopher. Bottom row: Overall performance on the massively multi-task language understanding benchmark (MMLU; Hendrycks et al., 2021a) as a function of training FLOPs, model parameters, and WikiText103 perplexity.

incentivizes scaling language models, it is important to be aware of risks that increase with model scale even if they are not emergent.

Here, we summarize several prior findings on the relationship between specific social risks and model scale. On WinoGender (Rudinger et al., 2017), which measures gender bias in occupations such as "nurse" or "electrician," scaling has improved performance so far (Du et al., 2021; Chowdhery et al., 2022), though BIG-Bench (2022) found in BBQ bias benchmark (Parrish et al., 2022) that bias can increase with scaling for ambiguous contexts. As for toxicity, Askell et al. (2021) found that while larger language models could produce more toxic responses from the RealToxicityPrompts dataset (Gehman et al., 2020), this behavior could be mitigated by giving models prompts with examples of being "helpful, harmless, and honest." For extracting training data from language models, larger models were found to be more likely to memorize training data (Carlini et al., 2021; 2022), though deduplication methods have been proposed and can simultaneously reduce memorization while improving performance (Kandpal et al., 2022; Lee et al., 2022a). The TruthfulQA benchmark (Lin et al., 2021) showed that GPT-3 models were more likely to mimic human falsehoods as they got larger, though Rae et al. (2021) later showed on a multiple-choice version that scaling Gopher to 280B enabled emergent performance substantially better than random.

Beyond the above, emergent risks also include phenomena that might only exist in future language models or that have not yet been characterized in current language models. Some such behaviors, as discussed in detail in Hendrycks et al. (2021b), could be backdoor vulnerabilities, inadvertent deception, or harmful content synthesis. Approaches involving data filtering, forecasting, governance, and automatically discovering harmful behaviors have been proposed for discovering and mitigating emergent risks (Bender et al., 2021; Weidinger et al., 2021; Steinhardt, 2021; Ganguli et al., 2022; Perez et al., 2022, *inter alia*). For a more detailed discussion of the risks of large language models, including emergent risks, see Bender et al. (2021); Steinhardt (2021); Bommasani et al. (2021); Ganguli et al. (2022).

5.5 Sociological changes

Finally, the emergent abilities discussed here focus on model behavior and are just one of several types of emergence in NLP (Manning et al., 2020; Teehan et al., 2022). Another notable type of qualitative change is sociological, in which increasing scale has shifted how the community views and uses language models. For instance, NLP has historically focused on task-specific models (Jurafsky & Martin, 2009). Recently, scaling has led to an explosion in research on and development of models that are "general purpose" in that they are single models that aim to perform a range of tasks not explicitly encoded in the training data (e.g., GPT-3, Chinchilla, and PaLM) (Manning, 2022).

One key set of results in the emergent sociological shift towards general-purpose models is when scaling enables a few-shot prompted general-purpose model to outperform prior state of the art held by finetuned task-specific models. As a few examples, GPT-3 175B achieved new state of the art on the TriviaQA and PiQA question-answering benchmarks (Brown et al., 2020); PaLM 540B achieved new state of the art on three arithmetic reasoning benchmarks (Chowdhery et al., 2022); and the multimodal Flamingo 80B model achieved new state of the art on six visual question answering benchmarks (Alayrac et al., 2022). In all of these cases, state-of-the-art performance was achieved by few-shot prompting a language model of unprecendented scale (scaling curves for these examples are shown in Appendix Figure 13). These abilities are not necessarily emergent since they have smooth, predictable scaling curves—however, they do underscore an emergent sociological shift towards general-purpose models in the NLP community.

The ability for general-purpose models to perform unseen tasks given only a few examples has also led to many new applications of language models outside the NLP research community. For instance, language models have been used via prompting to translate natural language instructions into actions executable by robots (Ahn et al., 2022; Huang et al., 2022), interact with users (Coenen et al., 2021; Wu et al., 2021; 2022a; Lee et al., 2022b), and facilitate multi-modal reasoning (Zeng et al., 2022; Alayrac et al., 2022). Large language models have also been deployed in the real-world both in products, such as GitHub CoPilot,[7] and directly as services themselves, such as OpenAI's GPT-3 API.[8]

5.6 Directions for future work

Future work on emergent abilities could involve train more-capable language models, as well as methods for better enabling language models to perform tasks. Some potential directions include but are not limited to the following.

Further model scaling. Further scaling up models has so far appeared to increase the capabilities of language models, and is a straightforward direction for future work. However, simply scaling up language models is computationally expensive and requires solving substantial hardware challenges, and so other approaches will likely play a key role in the future of the emergent abilities of large language models.

Improved model architectures and training. Improving model architecture and training procedures may facilitate high-quality models with emergent abilities while mitigating computational cost. One direction is using sparse mixture-of-experts architectures (Lepikhin et al., 2021; Fedus et al., 2021; Artetxe et al., 2021; Zoph et al., 2022), which scale up the number of parameters in a model while maintaining constant computational costs for an input. Other directions for better computational efficiency could involve variable amounts of compute for different inputs (Graves, 2016; Dehghani et al., 2018), using more localized learning strategies than backpropagation through all weights in a neural network (Jaderberg et al., 2017), and augmenting models with external memory (Guu et al., 2020; Borgeaud et al., 2021; Wu et al., 2022b, *inter alia*). These nascent directions have already shown promise in many settings but have not yet seen widespread adoption, which will likely require further work.

Data scaling. Training long enough on a large-enough dataset has been shown to be key for the ability of language models to acquire syntactic, semantic, and other world knowledge (Zhang et al., 2021; Wei et al., 2021b; Razeghi et al., 2022). Recently, Hoffmann et al. (2022) argued that prior work (Kaplan et al., 2020)

[7] https://copilot.github.com/
[8] https://beta.openai.com/docs/introduction

underestimated the amount of training data needed to train a compute-optimal model, underscoring the importance of training data. Collecting large datasets so that models can be trained for longer could allow a greater range of emergent abilities under a fixed model size constraint.

Better techniques for and understanding of prompting. Although few-shot prompting (Brown et al., 2020) is simple and effective, general improvements to prompting may further expand the abilities of language models. For instance, simple modifications such as calibrating output probabilities (Zhao et al., 2021; Holtzman et al., 2021) or using a noisy channel (Min et al., 2022a) have improved performance on a range of tasks. Augmenting few-shot exemplars with intermediate steps (Reynolds & McDonell, 2021; Nye et al., 2021; Wei et al., 2022b) has also enabled models to perform multi-step reasoning tasks not possible in the standard prompting formulation from Brown et al. (2020). Moreover, better exploration of what makes prompting successful (Wei et al., 2021a; Xie et al., 2022; Min et al., 2022b; Olsson et al., 2022) could lead to insights on how to elicit emergent abilities at a smaller model scale. Sufficient understanding of why models work generally lags the development and popularization of techniques such as few-shot prompting, and it is also likely that the best practices for prompting will change as more-powerful models are developed over time.

Frontier tasks. Although language models can perform a wide range of tasks, there are still many tasks that even the largest language models to date cannot perform with above-random accuracy. Dozens of such tasks from BIG-Bench are enumerated in Appendix E.4; these tasks often involve abstract reasoning (e.g., playing Chess, challenging math, etc). Future research could potentially investigate why these abilities have not yet emerged, and how to enable models to perform these tasks. Looking forward, another growing direction could be multilingual emergence; results on multilingual BIG-Bench tasks indicate that both model scale and training data play a role in emergence (e.g., Figure 2D shows that both using PaLM's training dataset and scaling to 62B parameters is required for question-answering in Persian). Other frontier tasks could include prompting in multiple modalities (Alayrac et al., 2022; Ramesh et al., 2022).

Understanding emergence. Beyond research on unlocking further emergence, an open question for future research is how and why emergent abilities occur in large language models. This paper conducted initial analyses regarding scaling of the cross-entropy loss on BIG-Bench (Appendix A.1), different metrics for generative tasks (Appendix A.2), and which types of tasks emergence occurs (Appendix A.3 and Appendix B). These analyses did not provide complete answers to why emergence occurs or how to predict it. Future research could potentially analyze emergence in new ways (e.g., analyze the relationship between emergent tasks and similar data in training; create a synthetic task that requires multiple compositional sub-tasks and evaluate how each of those sub-tasks improve with scale and unlock emergence when combined). Overall, understanding emergence is an important direction because it could potentially allow us predict what abilities future models may have, as well as provide new insights into how to train more-capable language models.

6 Conclusions

We have discussed emergent abilities of language models, for which meaningful performance has only been thus far observed at a certain computational scale. Emergent abilities can span a variety of language models, task types, and experimental scenarios. Such abilities are a recently discovered outcome of scaling up language models, and the questions of how they emerge and whether more scaling will enable further emergent abilities seem to be important future research directions for the field of NLP.

Broader Impact Statement

In this paper, we surveyed results in the existing literature, without proposing new methods or models. As discussed in (§5), emergent abilities are unpredictable in several ways, and include emergent risks (§5.4). We believe these phenomena warrant careful study and raise important questions for the field.

Acknowledgments

We thank Charles Sutton, Slav Petrov, Douglas Eck, Jason Freidenfelds, Jascha Sohl-Dickstein, Ethan Dyer, Dale Schuurmans, and Xavier Garcia for useful discussions and feedback on the manuscript.

GROKKING: GENERALIZATION BEYOND OVERFITTING ON SMALL ALGORITHMIC DATASETS

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin
OpenAI

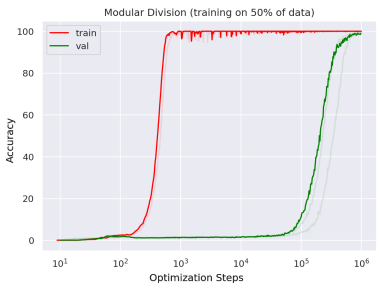
Vedant Misra*
Google

ABSTRACT

In this paper we propose to study generalization of neural networks on small algorithmically generated datasets. In this setting, questions about data efficiency, memorization, generalization, and speed of learning can be studied in great detail. In some situations we show that neural networks learn through a process of "grokking" a pattern in the data, improving generalization performance from random chance level to perfect generalization, and that this improvement in generalization can happen well past the point of overfitting. We also study generalization as a function of dataset size and find that smaller datasets require increasing amounts of optimization for generalization. We argue that these datasets provide a fertile ground for studying a poorly understood aspect of deep learning: generalization of overparametrized neural networks beyond memorization of the finite training dataset.

1 INTRODUCTION

The generalization of overparameterized neural networks has long been a source of interest to the machine learning community since it defies intuitions derived from classical learning theory. In this paper we show that training networks on small algorithmically generated datasets can reliably exhibit unusual generalization patterns, clearly decoupled from performance on the training set, in a significantly more pronounced way than such effects manifest on datasets derived from natural data (see Figure 1, left, for an example). Such experiments can be quickly reproduced on a single GPU, and this makes them convenient testbeds for theories of generalization.



★	a	b	c	d	e
a	a	d	?	c	d
b	c	d	d	a	c
c	?	e	d	b	d
d	a	?	?	b	c
e	b	b	c	?	a

Figure 1: **Left**. Grokking: A dramatic example of generalization far after overfitting on an algorithmic dataset. We train on the binary operation of division mod 97 with 50% of the data in the training set. Each of the 97 residues is presented to the network as a separate symbol, similar to the representation in the figure to the right. The red curves show training accuracy and the green ones show validation accuracy. Training accuracy becomes close to perfect at $< 10^3$ optimization steps, but it takes close to 10^6 steps for validation accuracy to reach that level, and we see very little evidence of any generalization until 10^5 steps. **Center**. Training time required to reach 99% validation accuracy increases rapidly as the training data fraction decreases. **Right**. An example of a small binary operation table. We invite the reader to make their guesses as to which elements are missing.

*Vedant was at OpenAI at the time of this work

The datasets we consider are binary operation tables of the form $a \circ b = c$ where a, b, c are discrete symbols with no internal structure, and $\circ$ is a binary operation. Examples of binary operations include addition, composition of permutations, and bivariate polynomials. Training a neural network on a proper subset of all possible equations then amounts to filling in the blanks of the binary op table, much like solving a Sudoku puzzle. An example is shown on the right in Figure 1. Since we use distinct abstract symbols for all distinct elements a, b, c involved in the equations, the network is not made aware of any internal structure of the elements, and has to learn about their properties only from their interactions with other elements. For example the network doesn't see numbers in decimal notation, or permutations in line notation.

Our contributions are as follows:

- We show that neural networks are capable of generalizing to the empty slots in a variety of binary op tables.
- We show that, long after severely overfitting, validation accuracy sometimes suddenly begins to increase from chance level toward perfect generalization. We call this phenomenon 'grokking'. An example is shown in Figure 1.
- We present the data efficiency curves for a variety of binary operations.
- We show empirically that the amount of optimization required for generalization quickly increases as the dataset size decreases.
- We compare various optimization details to measure their impact on data efficiency. We find that weight decay is particularly effective at improving generalization on the tasks we study.
- We visualize the symbol embeddings learned by these networks and find that they sometimes uncover recognizable structure of the mathematical objects represented by the symbols.

2 METHOD

All of our experiments used a small transformer trained on datasets of equations of the form $a \circ b = c$, where each of "a", "$\circ$", "b", "$=$", and "c" is a separate token. Details of the operations studied, the architecture, training hyperparameters and tokenization can be found in Appendix A.1.

3 EXPERIMENTS

3.1 GENERALIZATION BEYOND OVERFITTING

Deep learning practitioners are used to seeing small improvements in validation accuracy after validation loss stops decreasing. A double descent of validation loss has been documented in some circumstances, but is considered unusual among practitioners Nakkiran et al. (2019); Belkin et al. (2018); d'Ascoli et al. (2020). On the small algorithmic datasets that we study, improved generalization after initial overfitting occurs for a range of models, optimizers, and dataset sizes, and in some cases these effects are extremely pronounced. A typical example is shown for modular division in Figure 1. There we see that validation accuracy starts increasing beyond chance level only after 1000 times more optimization steps than are required for training accuracy to get close to optimal. In Figure 4 the training/validation losses are also plotted and we see the double descent of the validation loss.

We found these behaviors to be typical for all the binary operations for dataset sizes that were close to the minimal dataset size for which the network generalized within the allotted optimization budget. For larger dataset sizes, the training and validation curves tend to track each other more closely.

3.1.1 LEARNING TIME CURVES

In a typical supervised learning problem, decreasing the amount of training data decreases the converged generalization performance of the model when the optimization procedure is capable of interpolating the training data. In our setting, we observe a different phenomenon: while the converged performance stays constant at 100% within a range of training dataset sizes, the optimization time required to achieve that performance grows quicky as the dataset size is decreased.

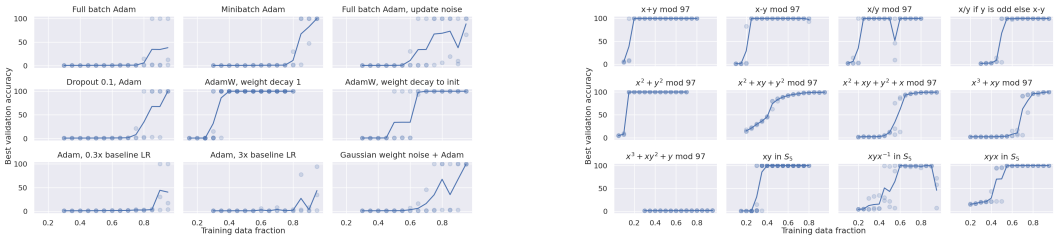


Figure 2: **Left**. Different optimization algorithms lead to different amounts of generalization within an optimization budget of 10^5 steps for the problem of learning the product in the abstract group S_5. Weight decay improves generalization the most, but some generalization happens even with full batch optimizers and models without weight or activation noise at high percentages of training data. Suboptimal choice hyperparameters severely limit generalization. Not shown: training accuracy reaches 100% after 10^3-10^4 updates for all optimization methods. **Right**. Best validation accuracy achieved after 10^5 steps on a variety of algorithmic datasets, averaged over 3 seeds. Generalization happens at higher percentages of data for intuitively more complicated and less symmetrical operations.

Figure 1 (center) shows median number of optimization steps until validation performance first reaches 99% for the product in abstract group S_5. In the vicinity of 25-30% of data, a decrease of 1% of training data leads to an increase of 40-50% in median time to generalization. While the number of steps until validation accuracy $> 99\%$ grows quickly as dataset size decreases, the number of steps until the train accuracy first reaches 99% generally trends down as dataset size decreases and stays in the range of 10^3-10^4 optimization steps. We've observed a similar pattern of exponential increase in optimization time until reaching generalization as dataset size decreases on all the algorithmic tasks for which we could get the networks to generalize.

3.2 Grokking on a variety of problems

We've measured the mean accuracy across three runs for training datasets consisting of different fractions of all available equations for a variety of binary operations listed in Appendix A.1.1. The results are presented in Figure 2 (right).

Since the operands are presented to the neural network as unrelated abstract symbols, the operations $x+y \pmod{p-1}$ and $x*y \pmod{p}$ with a prime number p and non-zero x, y are indistinguishable from the neural network's perspective (and similarly $x-y \pmod{p-1}$ and $x/y \pmod{p}$). This is because every nonzero residue modulo a prime can be represented as a power of a primitive root. This representation shows the equivalence (up to renaming of symbols) of modular addition modulo $p-1$ and modular multiplication modulo p. We see in Figure 2 (right) that $x-y$ and x/y indeed take about the same amount of data for generalization to occur.

Some of the operations listed in Figure 2 (right) are symmetric with respect to the order of the operands ($x+y$, $x*y$, x^2+y^2 and x^2+xy+y^2). Such operations tend to require less data for generalization than closely related non-symmetrical counterparts ($x-y$, x/y, x^2+xy+y^2+x). We believe this effect might be partially architecture-dependent, since it's easy for a transformer to learn a symmetric function of the operands by ignoring positional embedding.

Some operations (for example $x^3+xy^2+y \pmod{97}$) didn't lead to generalization within the allowed optimization budget at any percentage of data up to 95%. The converged models effectively just memorized the training dataset without finding any real patterns in the data. To such a model, the data is effectively random.

The operation $[x/y \pmod{p}$ if y is odd, otherwise $x-y \pmod{p}]$ requires the network to learn a mix of several simple operations - in particular the role of x has to be interpreted as a residue in the additive group when it's paired with an even y, and as a residue in the multiplicative group when it's paired with an odd y. This shows that generalization can happen even for operations that are not cleanly interpretable via group or ring operations.

3.3 Ablations and Tricks

We've tried various forms of regularization to see what can induce networks to generalize better on our datasets. Here we present the data efficiency curves on a particular dataset S_5 for a variety of interventions: full-batch gradient descent, stochastic gradient descent, large or small learning rates,

residual dropout Srivastava et al. (2014), weight decay Loshchilov & Hutter (2017) and gradient noise Neelakantan et al. (2015). The results are shown in Figure 2 (left).

We find that adding weight decay has a very large effect on data efficiency, more than halving the amount of samples needed compared to most other interventions. We found that weight decay towards the initialization of the network is also effective, but not quite as effective as weight decay towards the origin. This makes us believe that the prior, that approximately zero weights are suitable for small algorithmic tasks, explains part, but not all of the superior performance of weight decay. Adding some noise to the optimization process (e.g. gradient noise from using minibatches, Gaussian noise applied to weights before or after computing the gradients) is beneficial for generalization, consistent with the idea that such noise might induce the optimization to find flatter minima that generalize better. We found that learning rate had to be tuned in a relatively narrow window for the generalization to happen (within 1 order of magnitude).

3.4 Qualitative Visualization of Embeddings

In order to gain some insight into networks that generalize, we visualized the matrix of the output layer for the case of modular addition and S_5. In Figure 3 we show t-SNE plots of the row vectors. For some networks we find clear reflections of the structure of the underlying mathematical objects in the plots. For example the circular topology of modular addition is shown with a 'number line' formed by adding 8 to each element. The structure is more apparent in networks that were optimized with weight decay.

4 Discussion

We have seen that in the datasets we studied, small algorithmic binary operation tables, effects such as double descent or late generalization, and improvements to generalization from interventions like weight decay can be striking. This suggests that these datasets could be a good place to investigate aspects of generalization. For example, we plan to test whether various proposed measures of minima flatness correlate with generalization in our setting.

We have also seen that visualizing the embedding spaces of these neural networks can show natural kinds of structure, for example in problems of modular arithmetic the topology of the embeddings tends to be circles or cylinders. We also see that the network tends to idiosyncratically organize the embeddings by various residues. Whilst the properties of these mathematical objects are familiar to us, we speculate that such visualizations could one day be a useful way to gain intuitions about novel mathematical objects.

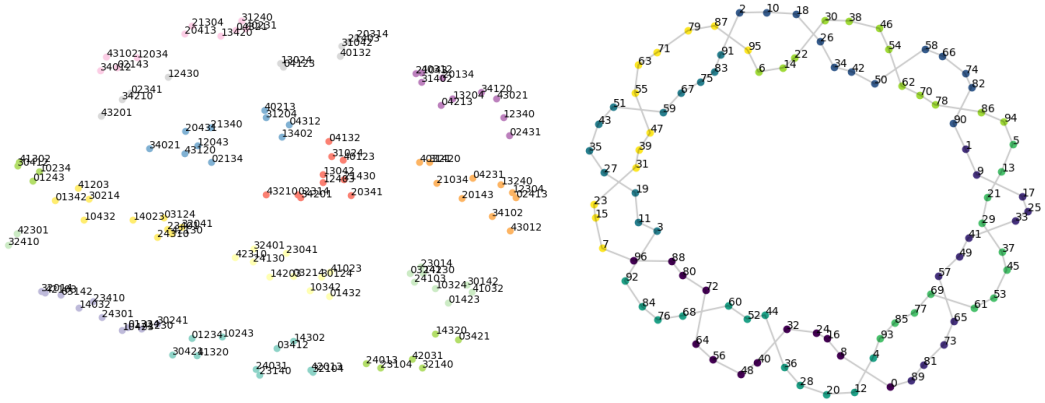


Figure 3: **Left**. t-SNE projection of the output layer weights from a network trained on S_5. We see clusters of permutations, and each cluster is a coset of the subgroup $\langle (0,3)(1,4), (1,2)(3,4) \rangle$ or one of its conjugates. **Right**. t-SNE projection of the output layer weights from a network trained on modular addition. The lines show the result of adding 8 to each element. The colors show the residue of each element modulo 8.

In addition, we document an interesting phenomenon, where the number of optimization steps needed to reach a given level of performance increases quickly as we reduce the size of the training dataset. Since this represents a way trade compute for performance on smaller amounts of data, it would be useful to investigate in future work whether the effect is also present for other datasets.

REFERENCES

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning practice and the bias-variance trade-off. *arXiv preprint arXiv:1812.11118*, 2018.

Stéphane d'Ascoli, Levent Sagun, and Giulio Biroli. Triple descent and the two kinds of overfitting: Where & why do they appear? *arXiv preprint arXiv:2006.03509*, 2020.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. *Advances in neural information processing systems*, 28: 1828–1836, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural computation*, 9(1):1–42, 1997.

Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. *arXiv preprint arXiv:1912.02178*, 2019.

Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016. URL `http://arxiv.org/abs/1609.04836`.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*, 2019.

Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

Scott Reed and Nando De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*, 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural turing machines-revised. *arXiv preprint arXiv:1505.00521*, 2015.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

A Appendix

A.1 Additional experimental details

A.1.1 Binary operations

The following are the binary operations that we have tried (for a prime number $p = 97$):

$x \circ y = x + y \pmod p$ for $0 \leq x, y < p$

$x \circ y = x - y \pmod p$ for $0 \leq x, y < p$

$x \circ y = x/y \pmod p$ for $0 \leq x < p, 0 < y < p$

$x \circ y = [x/y \pmod p$ if y is odd, otherwise $x - y \pmod p]$ for $0 \leq x, y < p$

$x \circ y = x^2 + y^2 \pmod p$ for $0 \leq x, y < p$

$x \circ y = x^2 + xy + y^2 \pmod p$ for $0 \leq x, y < p$

$x \circ y = x^2 + xy + y^2 + x \pmod p$ for $0 \leq x, y < p$

$x \circ y = x^3 + xy \pmod p$ for $0 \leq x, y < p$

$x \circ y = x^3 + xy^2 + y \pmod p$ for $0 \leq x, y < p$

$x \circ y = x \cdot y$ for $x, y \in S_5$

$x \circ y = x \cdot y \cdot x^{-1}$ for $x, y \in S_5$

$x \circ y = x \cdot y \cdot x$ for $x, y \in S_5$

For each binary operation we constructed a dataset of equations of the form $\langle x \rangle \langle op \rangle \langle y \rangle \langle = \rangle \langle x \circ y \rangle$, where $\langle a \rangle$ stands for the token corresponding to element a.

For each training run, we chose a fraction of all available equations at random and declared them to be the training set, with the rest of equations being the validation set.

A.1.2 Model and optimization

We trained a standard decoder-only transformer Vaswani et al. (2017) with causal attention masking, and calculated loss and accuracy only on the answer part of the equation. For all experiments we used a transformer with 2 layers, width 128, and 4 attention heads, with a total of about $4 \cdot 10^5$ non-embedding parameters.

We have tuned optimization hyperparameters by running experiments on modular addition and product in S_5. For final configuration of hyperparameters we have chosen a balance of performance we saw on S_5 and simplicity (for example we chose not to anneal the learning rate for the experiments in the paper even though it performed better in some situations). For most experiments we used AdamW optimizer with learning rate 10^{-3}, weight decay 1, $\beta_1 = 0.9$, $\beta_2 = 0.98$, linear learning rate warmup over the first 10 updates, minibatch size 512 or half of training dataset size (whichever was smaller) and optimization budget of 10^5 gradient updates.

In section 3.3 we have also tried the following variants (listed in the reading order for Figure 2 left):

- Adam optimizer with full batch (i.e. exact gradient of the loss on the whole training dataset)
- Adam optimizer

- Adam optimizer with full batch and Gaussian noise added to the update direction for each parameter ($W \leftarrow W + \text{lr} \cdot (\Delta W + \epsilon)$, where ϵ is sampled from unit Gaussian, ΔW is the standard Adam weight update, and lr is the learning rate)
- Adam optimizer on model with residual dropout 0.1 added
- AdamW optimizer with weight decay 1 (default setting in most other experiments)
- AdamW optimizer with weight decay 1 towards the initialization instead of the origin
- Adam optimizer with learning rate $3 \cdot 10^{-4}$
- Adam optimizer with learning rate $3 \cdot 10^{-3}$
- Adam optimizer on model with Gaussian weight noise of standard deviation 0.01 (i.e. each parameter W replaced by $W + 0.01 \cdot \epsilon$ in the model, with ϵ sampled from unit Gaussian).

For experiments reported in Section 3.1.1 we increased the optimization budget to $5 \cdot 10^5$ optimization steps in order to capture the increase of time to perfect generalization better.

For the experiments reported in Section 3.1 we increased the optimization budget to 10^6, and used Adam optimizer with no weight decay, for emphasizing how late into the optimization process the generalization can begin.

We've repeated each experiment for each dataset size with 3 random seeds, with the exception of experiments in section 3.1.1, where we've aggregated results over 7 random seeds.

A.2 Additional Figures

In Figure 4 we show the loss curves that correspond to the accuracy curves in Figure 1.

In Figure 5 we show an example of a binary operation table that the network can actually solve.

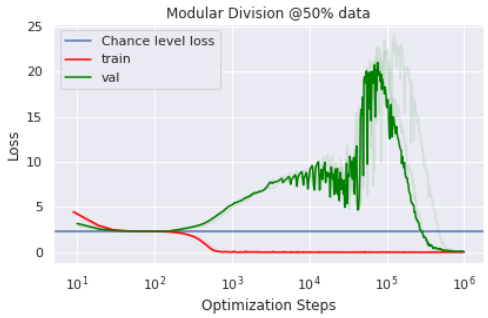


Figure 4: The loss curves for modular division, train and validation. We see the validation loss increases from 10^2 to about 10^5 optimization steps before it begins a second descent.

A.3 Related Work

In this paper we study training and generalization dynamics on small simple algorithmic datasets. In the past, algorithmic datasets have been used to probe the capability of neural networks to perform symbolic and algorithmic reasoning. For example the tasks of copying, reversing, and sorting randomly generated sequences, and performing arithmetic operations of multi-digit numbers, have been used as standard benchmarks for sequence-to-sequence models Graves et al. (2014), Weston et al. (2014) Kaiser & Sutskever (2015) Reed & De Freitas (2015), Grefenstette et al. (2015), Zaremba & Sutskever (2015), Graves (2016), Dehghani et al. (2018). Typically in these works however the emphasis is on the performance in the unlimited data regime, with generalization often studied with respect to input sequence length. Some papers study the sample complexity on algorithmic tasks Reed & De Freitas (2015), but mostly focus on the impact of architectural choices. In contrast we study the phenomenon of generalization in data-limited regime, with an emphasis on phenomena that we believe to be architecture-agnostic.

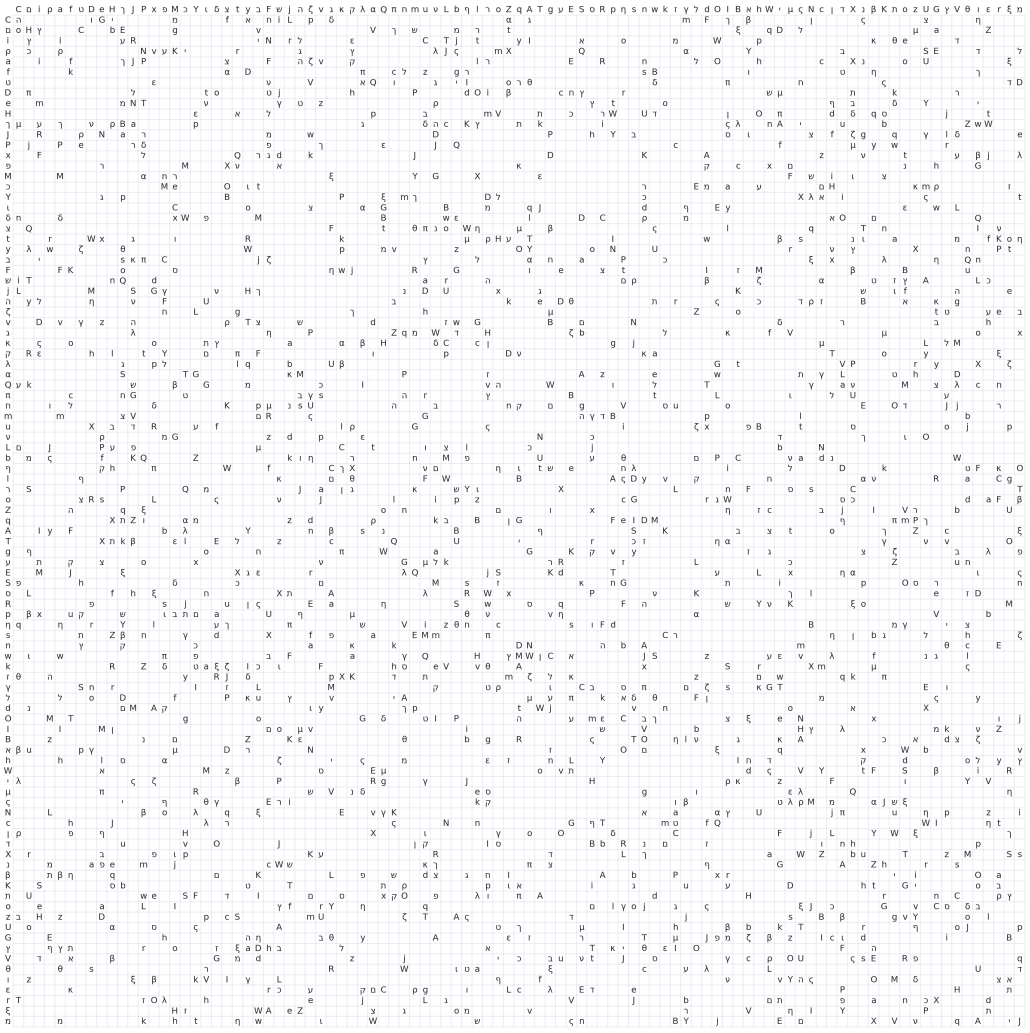


Figure 5: One of the binary operation tables presented to the networks that the network can perfectly fill in. Each symbol is represented as a letter in English, Hebrew, or Greek alphabet for reader's convenience. We invite the reader to guess which operation is represented here.

Algorithmically generated reasoning datasets like bAbI Weston et al. (2015) encourage work on studying generalization in data-limited regime. Most results on such datasets however focus on a point estimate of performance of a particular architecture or training technique, whereas our main interest is in pointing out the change in generalization past the point where a particular architecture can memorize the training data completely.

Neelakantan et al. (2015) has a "grok-like" learning curve on an algorithmic task, but it is related to optimization difficulty, whereas our phenomenon is specifically about generalization.

In Saxton et al. (2019) they study generalization on procedurally generated math problems such as arithmetic and differentiation, but for the most part these tasks are more involved than the simple binary op problems we have studied and as such do not lend themselves to observing the kinds of phenomena we describe in this paper, since they would require an extremely large number of samples to master.

In Jiang et al. (2019) they studied a large number of generalization or complexity measures on convolutional neural networks to see which, if any, are predictive of generalization performance. They find that flatness based measures that aim to quantify the sensitivity of the trained neural network to parameter perturbations are the most predictive. We conjectured that the grokking phenomena

we report in this work may be due to the noise from SGD driving the optimization to flatter/simpler solutions that generalize better and hope to investigate in future work whether any of these measures are predictive of grokking.

Zhang et al. (2016) finds that neural networks of sizes typically used in deep learning can interpolate arbitrary training data, and yet generalize when trained with semantically meaningful labels using appropriate optimization procedures. Our work shows a related phenomenon where neural networks can interpolate a small algorithmic training dataset without generalizing, but start generalizing when trained with SGD for longer.

Nakkiran et al. (2019); Belkin et al. (2018) focus on the phenomenon of double descent in loss as a function of model and optimization procedure capacity. They find that the classical U-shaped validation loss curve is followed in some settings (including neural network training) by a second descent of loss that starts around the minimal capacity that is needed to interpolate any training data. We observe a second descent in validation loss (though not accuracy) as a function of the amount of training in some of our experiments, and it happens past the point of interpolating the training data. We believe that the phenomenon we describe might be distinct from the double descent phenomena described in Nakkiran et al. (2019); Belkin et al. (2018) because we observe the second descent in loss far past the first time the training loss becomes very small (tens of thousands of epochs in some of our experiments), and we don't observe a non-monotonic behavior of accuracy. The setting of small algorithmic datasets that we study also provides a smaller, more tractable playground for studying subtle generalization phenomena than natural datasets studied in Nakkiran et al. (2019).

A.4 Generalization with memorizing several outliers

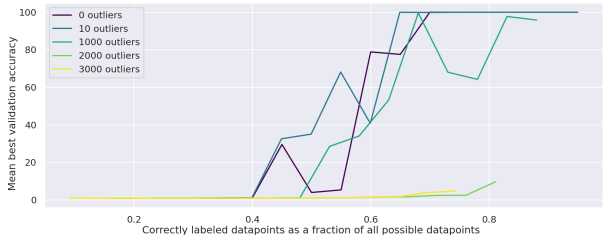


Figure 6: Effect on data efficiency of introducing $k \in [0, 10, 100, 1000, 2000, 3000]$ outliers (examples with random labels) into the training data. Small number of outliers doesn't noticeably impact generalization performance, but a large number hinders it significantly.

In this section we show data efficiency curves for a modified version of a binary op by introducing k outliers to the training dataset. More precisely, at the beginning of the experiment we randomly sample k equations from the training set and replace their answers with answers to other k equations randomly sampled from the training data. The rest of the equations in the training data and all the equations in the validation data are kept as before.

In this situation one could imagine one of the following scenarios unfolding. If the model class of neural networks optimized and regularized as before was not large enough to interpolate such "noisy" dataset, one could imagine the procedure converging to a solution that generalizes well, but denoises the training data (i.e. predicts $c = a \circ b$ as an answer even for the outlier equations $a, b \to c'$ with $c' \neq c$). On the other extreme it could be that the optimization procedure can find networks that interpolate the data, but the resulting models don't generalize, because they are forced to represent a considerably more complicated function than before (a simple function + k exceptions encoded in the training data).

In our experiments we find that the first option doesn't happen - all experiments reach 100% training accuracy at some point, and this point is not considerably affected by changing the number of outliers k. The second phenomenon happens in a range of training data percentages and number of outliers k - increasing k decreases the range of training data percentages for which the optimization procedure converges to models that generalize. However the effect of introducing a small number of outliers (up to 1000) is not very pronounced - see Figure 6. We interpret this as additional evidence that the capacity of the network and optimization procedure is well beyond the capacity needed for

memorizing all the labels on the training data, and that generalization happening at all requires a non-trivial explanation.

A.5 GENERALIZATION MEASURES

We believe it is useful to explore how predictive common generalization measures are of generalization on small algorithmic datasets presented in this paper. In a preliminary investigation we found that sharpness Hochreiter & Schmidhuber (1997) of the minimum found by a trained network measure seems to be predictive of generalization on one of these datasets. We trained multiple networks with different initialization seeds for a fixed number a steps on the $S5$ composition objective, until approximately half of them achieved high validation accuracy. We then used the method described in Keskar et al. (2016) to calculate the sharpness approximation value, ϕ. We found that the validation accuracy and the ϕ score across our trained networks had Spearman correlation coefficient of -0.79548 (significant with $p < 0.000014$). This is suggestive that grokking may only happen after the network's parameters are in flatter regions of the loss landscape. It would be valuable for future work to explore this hypothesis, as well as test other generalization measures.

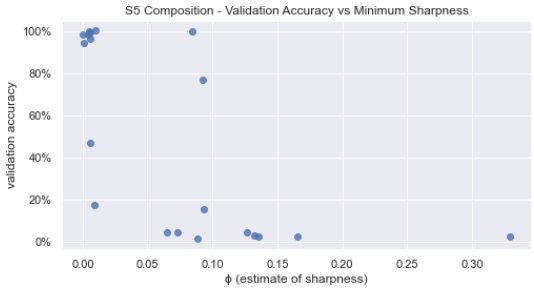


Figure 7: Networks trained on the $S5$ composition objective appear to only grok in relatively flat regions of the loss landscape.

What will GPT-2030 look like?

by **Jacob Steinhardt** 7th Jun 2023

Forecasts (Specific Predictions) | Scaling Laws | AI | Curated

GPT-4 surprised many people with its abilities at coding, creative brainstorming, letter-writing, and other skills. Surprises in machine learning are not restricted to GPT-4: I was previously surprised by Minerva's mathematical abilities, as were many competitive forecasters.

How can we be less surprised by developments in machine learning? Our brains often implicitly make a zeroth-order forecast: looking at the current state of the art, and adding on improvements that "feel reasonable". But what "seems reasonable" is prone to cognitive bias, and will underestimate progress in a fast-moving field like ML. A more effective approach is first-order forecasting: quantifying the historical rate of progress and extrapolating it forward, while also considering reasons for possible slowdowns or speedups.[1]

In this post, I'll use this approach to forecast the properties of large pretrained ML systems in 2030. I'll refer throughout to "GPT_{2030}", a hypothetical system that has the capabilities, computational resources, and inference speed that we'd project for large language models in 2030 (but which was likely trained on other modalities as well, such as images). To forecast GPT_{2030}'s properties, I consulted a variety of sources, including empirical scaling laws, projections of future compute and data availability, velocity of improvement on specific benchmarks, empirical inference speed of current systems, and possible future improvements in parallelism.

GPT_{2030}'s capabilities turn out to be surprising (to me at least). In particular, GPT_{2030} will enjoy a number of significant advantages over current systems[2], as well as (in at least some important respects) current human workers:

1. GPT_{2030} will likely be superhuman at various specific tasks, including coding, hacking, and math, and potentially protein engineering (Section 1).
2. GPT_{2030} can "work" and "think" quickly: I estimate it will be 5x as fast as humans as measured by words processed per minute [*range: 0.5x-20x*][3], and that this could be increased to 125x by paying 5x more per FLOP (Section 2).
3. GPT_{2030} can be copied arbitrarily and run in parallel. The organization that trains GPT_{2030} would have enough compute to run many parallel copies: I estimate enough to perform 1.8 million years of work when adjusted to human working speeds [*range: 0.4M-10M years*] (Section 3). Given the 5x speed-up in the previous point, this work could be done in 2.4 months.
4. GPT_{2030}'s copies can share knowledge due to having identical model weights, allowing for rapid parallel learning: I estimate 2,500 human-equivalent years of learning in 1 day (Section 4).
5. GPT_{2030} will be trained on additional modalities beyond text and images, possibly including counterintuitive modalities such as molecular structures, network traffic, low-

level machine code, astronomical images, and brain scans. It may therefore possess a strong intuitive grasp of domains where we have limited experience, including forming concepts that we do not have (Section 5).

These capabilities would, at minimum, accelerate many areas of research while also creating serious vectors for misuse (Section 6). Regarding misuse, GPT_{2030}'s programming abilities, parallelization, and speed would make it a potent cyberoffensive threat. Additionally, its rapid parallel learning could be turned towards human behavior and thus used to manipulate and misinform with the benefit of thousands of "years" of practice.

On acceleration, a main bottleneck will be autonomy. In a domain like mathematics research where work can be checked automatically, I'd predict that GPT_{2030} will outcompete most professional mathematicians. In machine learning, I'd predict that GPT_{2030} will independently execute experiments and generates plots and write-ups, but that graduate students and research scientists will provide direction and evaluate results. In both cases, GPT_{2030} will be an integral part of the research process.

My forecast of GPT_{2030}'s properties are not intuitive from looking at today's systems, and they may be wrong, since there is significant uncertainty about how ML will look in 2030. However, properties (1.-5.) above are my median bet, and whatever GPT_{2030} is like, I doubt it will be "GPT-4 but a bit better".

If I'm right, then whatever the impacts of AI are, they won't be small. We should be preparing for those impacts now, asking what will happen at the largest scales (on the order of $1T, 10M lives, or significant disruptions to social processes). It's better to be surprised now, rather than in 7 years when the system is already being rolled out.

1. Specific Capabilities

I expect GPT_{2030} to have superhuman coding, hacking, and mathematical abilities. I also expect it to be superhuman in its ability to read and process large corpora for patterns and insights and to recall facts. Finally, since AlphaFold and AlphaZero had superhuman abilities in protein engineering and game-playing, GPT_{2030} could as well, for instance if it was trained multimodally on similar data to the AlphaFold/AlphaZero models.

Programming. GPT-4 outperformed a strong human baseline on LeetCode problems posed after its training cutoff (Bubeck et al. 2023, Table 2), and passed the mock interview for several major tech companies (Figure 1.5). The velocity of improvement remains high, with a 19% jump from GPT-3 to 4. On the more challenging CodeForces competition, GPT-4 does less well, but AlphaCode is on par with the median CodeForces competitor. On the even more challenging APPS dataset, Parsel further outperforms AlphaCode (7.8%->25.5%). Looking forward, the forecasting platform Metaculus gives a median year of 2027° for 80% on APPS, which would exceed all but the very best humans.[4]

Hacking. I expect hacking to improve with general coding ability, plus ML models can scour large codebases for vulnerabilities much more scalably and conscientiously than humans.

ChatGPT has already been used to generate exploits, including polymorphic malware, which is typically considered to be an advanced offensive capability.

Math. Minerva achieved 50% accuracy on a competition math benchmark (MATH), which is better than most human competitors. The velocity of progress is high (>30% in 1 year), and there is significant low-hanging fruit via autoformalization, reducing arithmetic errors, improving chain-of-thought, and better data[5]. Metaculus predicts 92% on MATH by 2025°, and gives a median year of 2028° for AI winning a gold medal at the International Math Olympiad, on par with the best high school students in the world. I personally expect GPT_{2030} to be better than most professional mathematicians at proving well-posed theorems.[6]

Information processing. Factual recall and processing large corpora are natural consequences of language models' memorization capabilities and large context windows. Empirically, GPT-4 achieves 86% accuracy on MMLU, a broad suite of standardized exams including the bar exam, MCAT, and college math, physics, biochemistry, and philosophy; even accounting for likely train-test contamination, this probably exceeds the breadth of knowledge of any living human. Regarding large corpora, Zhong et al. (2023) used GPT-3 to construct a system that discovered and described several previously unknown patterns in large text datasets, and scaling trends on a related task in Bills et al. (2023) suggest that models will soon be superhuman. Both of these works exploit the large context windows of LLMs, which are now over 100,000 tokens and growing.

More generally, **ML models have a different skill profile than humans**, since humans and ML were adapted to very different data sources (evolution vs. massive internet data). At the point that models are human-level at tasks such as video recognition, they will likely be superhuman at many other tasks (such as math, programming, and hacking). Furthermore, additional strong capabilities will likely emerge over time due to larger models and better data, and there is no strong reason to expect model capabilities to "level out" at or below human-level. While it is possible that current deep learning approaches will fall short of human-level capabilities in some domains, it is also possible that they will surpass them, perhaps significantly, especially in domains such as math that humans are not evolutionarily specialized for.

2. Inference Speed

(*Thanks to Lev McKinney for running the performance benchmarks for this section.*)

To study the speed of ML models, we'll measure how quickly ML models generate text, benchmarking against the human thinking rate of 380 words per minute (Korba (2016), see also Appendix A). Using OpenAI's chat completions API, we estimate that gpt-3.5-turbo can generate 1200 words per minute (wpm), while gpt-4 generates 370 wpm, as of early April 2023. Smaller open source models like pythia-12b achieve at least 1350 wpm with out-of-the-box tools on an A100 GPU, and twice this appears possible with further optimization.

Thus, if we consider OpenAI models as of April, we are either at roughly 3x human speed, or equal to human speed. I predict that models will have faster inference speed in the future, as there are strong commercial and practical pressures towards speeding up inference. Indeed, in

the week leading up to this post, GPT-4's speed already increased to around 540wpm (12 tokens/second), according to Fabien Roger's tracking data; this illustrates that there is continuing room and appetite for improvement.

My median forecast is that models will have **5x the words/minute of humans** (range: [0.5x, 20x]), as that is roughly where there would be diminishing practical benefits to further increases, though there are considerations pointing to both higher or lower numbers. I provide a detailed list of these considerations in Appendix A, as well as comparisons of speeds across model scales and full details of the experiments above.

Importantly, **the speed of an ML model is not fixed**. Models' serial inference speed can be increased by k^2 at a cost of a k-fold reduction in throughput (in other words, k^3 parallel copies of a model can be replaced with a single model that is k^2 times faster). This can be done via a parallel tiling scheme that theoretically works even for large values of k^2, likely at least 100 and possibly more. Thus, a model that is 5x human speed could be sped up to 125x human speed by setting $k=5$.

An important caveat is that speed is not necessarily matched by quality: as discussed in Section 1, GPT_{2030} will have a different skill profile than humans, failing at some tasks we find easy and mastering some tasks we find difficult. We should therefore not think of GPT_{2030} as a "sped-up human", but as a "sped-up worker" with a potentially counterintuitive skill profile.

Nevertheless, considering speed-ups is still informative, especially when they are large. For language models with a 125x speed-up, cognitive actions that take us a day could be completed in minutes, assuming they were within GPT_{2030}'s skill profile. Using the earlier example of hacking, exploits or attacks that are slow for us to generate could be created quickly by ML systems.

3. Throughput and Parallel Copies

Models can be copied arbitrarily subject to available compute and memory. This allows them to quickly do any work that can be effectively parallelized. In addition, once one model is fine-tuned to be particularly effective, the change could be immediately propagated to other instances. Models could also be distilled for specialized tasks and thus run faster and more cheaply.

There will likely be enough resources to run many copies of a model once it has been trained. This is because training a model requires running many parallel copies of it, and whatever organization trained the model will still have those resources at deployment time. We can therefore lower bound the number of copies by estimating training costs.

As an example of this logic, the cost of training GPT-3 was enough to run it for 9×10^{11} forward passes. To put that into human-equivalent terms, humans think at 380 words per minute (see Appendix A) and one word is 1.33 tokens on average, so 9×10^{11} forward passes corresponds to ~3400 years of work at human speed. Therefore, the organization could run 3400 parallel copies of the model for a full year at human working-speeds, or the same number of copies for 2.4 months at 5x human speed.

Let's next project this same "training overhang" (ratio of training to inference cost) for future models. It should be larger: the main reason is that training overhang is roughly proportional to dataset size, and datasets are increasing over time. This trend will be slowed as we run out of naturally-occuring language data, but new modalities as well as synthetic or self-generated data will still push it forward.[7] In Appendix B, I consider these factors in detail to project forward to 2030. I forecast that models in 2030 will be trained with enough resources to perform **1,800,000 years of work** adjusted to human speed [*range: 400k-10M*].

Note that Cotra (2020) ° and Davidson (2023) estimate similar quantities and arrive at larger numbers than me; I'd guess the main difference is how I model the effect of running out of natural language data.

The projection above is somewhat conservative, since models may be run on more resources than they were trained on if the organization buys additional compute. A quick ballpark estimate suggests that GPT-4 was trained on about 0.01% of all computational resources in the world, although I expect future training runs to use up a larger share of total world compute and therefore have less room to scale up further after training. Still, an organization could possibly increase the number of copies they run by another order of magnitude if they had strong reasons to do so.

4. Knowledge Sharing

(*Thanks to Geoff Hinton who first made this argument to me.*)

Different copies of a model can share parameter updates. For instance, ChatGPT could be deployed to millions of users, learn something from each interaction, and then propagate gradient updates to a central server where they are averaged together and applied to all copies of the model. In this way, ChatGPT could observe more about human nature in an hour than humans do in a lifetime (1 million hours = 114 years). Parallel learning may be one of the most important advantages models have, as it means they can rapidly learn any missing skills.

The rate of parallel learning depends on how many copies of a model are running at once, how quickly they can acquire data, and whether the data can be efficiently utilized in parallel. On the last point, even extreme parallelization should not harm learning efficiency much, as batch sizes in the millions are routine in practice, and the gradient noise scale (McCandlish et al., 2018) predicts minimal degradation in learning performance below a certain "critical batch size". We'll therefore focus on parallel copies and data acquisition.

I will provide two estimates that both suggest it would be feasible to have at least ~1 million copies of a model learning in parallel at human speed. This corresponds to **2500 human-equivalent years of learning per day**, since 1 million days = 2500 years.

The first estimate uses the numbers from Section 3, which concluded that the cost of training a model is enough to simulate models for 1.8M years of work (adjusted to human speed). Assuming that the training run itself lasted for less than 1.2 years (Sevilla et al., 2022), this means the organization that trained the model has enough GPUs to run 1.5M copies at human speed.

The second estimate considers the market share of the organization deploying the model. For example, if there are 1 million users querying the model at a time, then the organization necessarily has the resources to serve 1 million copies of the model. As a ballpark, ChatGPT had 100 million users as of May 2023 (not all active at once), and 13 million active users/day as of January 2023. I'd assume the typical user is requesting a few minutes worth of model-generated text, so the January number probably only implies around 0.05 million person-days of text each day. However, it seems fairly plausible that future ChatGPT-style models would 20x this, reaching 250 million active users/day or more and hence 1 million person-days of data each day. As a point of comparison, Facebook has 2 billion daily active users.

5. Modalities, Tools, and Actuators

Historically, GPT-style models have primarily been trained on text and code, and had limited capacity to interact with the outside world except via chat dialog. However, this is rapidly changing, as models are being trained on additional modalities such as images, are being trained to use tools, and are starting to interface with physical actuators. Moreover, models will not be restricted to anthropocentric modalities such as text, natural images, video, and speech---they will likely also be trained on unfamiliar modalities such as network traffic, astronomical images, or other massive data sources.

Tools. Recently-released models use external tools, as seen with ChatGPT plugins as well as Schick et al. (2023), Yao et al. (2022), and Gao et al. (2022). Text combined with tool use is sufficient to write code that gets executed, convince humans to take actions on their behalf, make API calls, make transactions, and potentially execute cyberattacks. Tool use is economically useful, so there will be strong incentives to further develop this capability.

> ChatGPT is reactive: user says X, ChatGPT responds with Y. Risks exist but are bounded. Soon it will be tempting to have proactive systems - an assistant that will answer emails for you, take actions on your behalf, etc. Risks will then be much higher.
>
> — Percy Liang (@percyliang) February 27, 2023

New modalities. There are now large open-source vision-language models such as OpenFlamingo, and on the commercial side, GPT-4 and Flamingo were both trained on vision and text data. Researchers are also experimenting with more exotic pairs of modalities such as proteins and language (Guo et al., 2023).

We should expect the modalities of large pretrained models to continue to expand, for two reasons. First, economically, it is useful to pair language with less familiar modalities (such as proteins) so that users can benefit from explanations and efficiently make edits. This predicts multimodal training with proteins, biomedical data, CAD models, and any other modality associated with a major economic sector.

Second, we are starting to run out of language data, so model developers will search for new types of data to continue benefiting from scale. Aside from the traditional text and videos, some of the largest existing sources of data are astronomical data (will soon be at exabytes

per day) and genomic data (around 0.1 exabytes/day). It is plausible that these and other massive data sources will be leveraged for training GPT_{2030}.

The use of exotic modalities means that GPT_{2030} might have unintuitive capabilities. It might understand stars and genes much better than we do, even while it struggles with basic physical tasks. This could lead to surprises, such as designing novel proteins, that we would not have expected based on GPT_{2030}'s level of "general" intelligence. When thinking about the impacts of GPT_{2030}, it will be important to consider specific superhuman capabilities it might possess due to these exotic data sources.

Actuators. Models are also beginning to use physical actuators: ChatGPT has already been used for robot control and OpenAI is investing in a humanoid robotics company. However, it is much more expensive to collect data in physical domains than digital domains, and humans are also more evolutionarily adapted to physical domains (so the bar for ML models to compete with us is higher). Compared to digital tools, I'd therefore expect mastery of physical actuators to occur more slowly, and I'm unsure if we should expect it by 2030. Quantitatively, I'd assign 40% probability to there being a general-purpose model in 2030 that is able to autonomously assemble a scale-replica Ferrari as defined in this Metaculus question°.

6. Implications of GPT-2030

We'll next analyze what a system like GPT_{2030} would mean for society. A system with GPT_{2030}'s characteristics would, at minimum, significantly accelerate some areas of research, while also possessing powerful capacities for misuse.

I'll start by framing some general strengths and limitations of GPT_{2030}, then use this as a lens to analyze both acceleration and misuse.

Strengths. GPT_{2030} represents a large, highly adaptable, high-throughput workforce. Recall that GPT_{2030} could do 1.8 million years of work[8] across parallel copies, where each copy is run at 5x human speed. This means we could simulate 1.8 million agents working for a year each in 2.4 months. As discussed above, we could also instead run 1/5 as many copies at 125x human speed, so we could simulate 360,000 agents working for a year each in *3 days*.

Limitations. There are three obstacles to utilizing this digital workforce: skill profile, experiment cost, and autonomy. On the first, GPT_{2030} will have a different skill profile from humans that makes it worse at some tasks (but better at others). On the second, simulated workers still need to interface with the world to collect data, which has its own time and compute costs. Finally, on autonomy, models today can only generate a few thousand tokens in a chain-of-thought before getting "stuck", entering a state where they no longer produce high-quality output. We'd need significant increases in reliability before delegating complex tasks to models. I expect reliability to increase, but not without limit: my (very rough) guess is that GPT_{2030} will be able to run for several human-equivalent days before having to be reset or steered by external feedback. If models run at a 5x speed-up, that means they need human oversight every several hours.

Therefore, the tasks that GPT_{2030} would most impact are tasks that:

1. Leverage skills that GPT_{2030} is strong at relative to humans.
2. Only require external empirical data that can be readily and quickly collected (as opposed to costly physical experiments).
3. Can be a priori decomposed into subtasks that can be performed reliably, or that have clear and automatable feedback metrics to help steer the model.

Acceleration. One task that readily meets all three criteria is mathematics research. On the first, GPT_{2030} will likely have superhuman mathematical capabilities (Section 1). On the second and third, math can be done purely by thinking and writing, and we know when a theorem has been proved. There are furthermore not that many mathematicians in total in the world (e.g. only 3,000 in the US) so GPT_{2030} could simulate 10x or more the annual output of mathematicians every few days.

Significant parts of ML research also meet the criteria above. GPT_{2030} would be superhuman at programming, which includes implementing and running experiments. I'd guess it will also be good at presenting and explaining the results of experiments, given that GPT-4 is good at explaining complex topics in an accessible way (and there is significant market demand for this). Therefore, ML research might reduce to thinking up good experiments to run and interfacing with high-quality (but potentially unreliable) write-ups of the results. In 2030, grad students might therefore have the same resources as a professor with several strong students would have today.

Parts of social science could also be significantly accelerated. There are many papers where the majority of the work is chasing down, categorizing, and labeling scientifically interesting sources of data and extracting important patterns—see Acemoglu et al. (2001) or Webb (2020) for representative examples. This satisfies requirement (3.) because categorization and labeling can be decomposed into simple subtasks, and it satisfies requirement (2.) as long as the data is available on the internet, or could be collected through an online survey.

Misuse. Beyond acceleration, there would be serious risks of misuse. The most direct case is cyberoffensive hacking capabilities. Inspecting a specific target for a specific style of vulnerability could likely be done reliably, and it is easy to check if an exploit succeeds (subject to being able to interact with the code), so requirement (3.) is doubly satisfied. On (2.), GPT_{2030} would need to interact with target systems to know if the exploit works, which imposes some cost, but not enough to be a significant bottleneck. Moreover, the model could locally design and test exploits on open source code as a source of training data, so it could become very good at hacking before needing to interact with any external systems. Thus, GPT_{2030} could rapidly execute sophisticated cyberattacks against large numbers of targets in parallel.

A second source of misuse is manipulation. If GPT_{2030} interacts with millions of users at once, then it gains more experience about human interaction in an hour than a human does in their lifetime (1 million hours = 114 years). If it used these interactions to learn about manipulation, then it could obtain manipulation skills that are far greater than humans—as an analogy, con artists are good at tricking victims because they've practiced on hundreds of

people before, and GPT_{2030} could scale this up by several orders of magnitude. It could therefore be very good at manipulating users in one-on-one conversation, or at writing news articles to sway public opinion.

Thus in summary, GPT_{2030} could automate almost all mathematics research as well as important parts of other research areas, and it could be a powerful vector of misuse regarding both cyberattacks and persuasion/manipulation. Much of its impact would be limited by "oversight bottlenecks", so if it could run autonomously for long periods of time then its impact may be larger still.

Thanks to Louise Verkin for transcribing this post to Ghost format, and Lev McKinney for running empirical benchmark experiments. Thanks to Karena Cai, Michael Webb, Leo Aschenbrenner, Anca Dragan, Roger Grosse, Lev McKinney, Ruiqi Zhong, Sam Bowman, Tatsunori Hashimoto, Percy Liang, Tom Davidson, and others for providing feedback on drafts of this post.

Appendix: Runtime and Training Estimates for Future Models

A. Words per minute

First we'll estimate the word per minute of humans and of current models. Then we'll extrapolate from current models to future models.

For humans, there are five numbers we could measure: talking speed, reading speed, listening speed, and both "elliptic" and "extended" thinking speed. Regarding the first three, Rayner and Clifton (2009) say that reading speed is 300 words per minute[9] and speaking is 160 words per minute[10], and that listening can be done 2-3 times faster than speaking (so ~400 words per minute)[11]. For thinking speed, we need to distinguish between "elliptic" and "extended" thought—it turns out that we think in flashes of words rather than complete sentences, and if we extend these flashes to full sentences we get very different word counts (~10x different). Korba (2016) find that elliptic thought is 380 words per minute while extended thought is ~4200 words per minute. Since most of these numbers cluster in the 300-400 wpm range, I'll use **380 words per minute** as my estimate of human thinking speed. Using the 4:3 token to word ratio suggested by OpenAI, this comes out to **500 tokens per minute**.[12]

(*Thanks to Lev McKinney for running the evaluations in the following paragraphs.*)
Next, let's consider current models. We queried gpt-3.5-turbo and gpt-4, as well as several open source models from EleutherAI, to benchmark their inference speed. We did this by querying the models to count from 1 to n, where n ranged from 100 to 1900 inclusive in increments of 100. Since numbers contain more than one token, we cut the model off when it reached n tokens generated, and measured the time elapsed. We then ran a linear regression with a bias term to account for latency in order to estimate the asymptotic number of tokens per second.

Further Reading

A short introduction to machine learning
Ngo (2021)
www.alignmentforum.org/posts/qE73pqxAZmeACsAdF/a-short-introduction-to-machine-learning

Machine Learning for Humans, Part 2.1: Supervised Learning
Maini and Sabri (2017)
medium.com/@v_maini/supervised-learning-740383a2feab