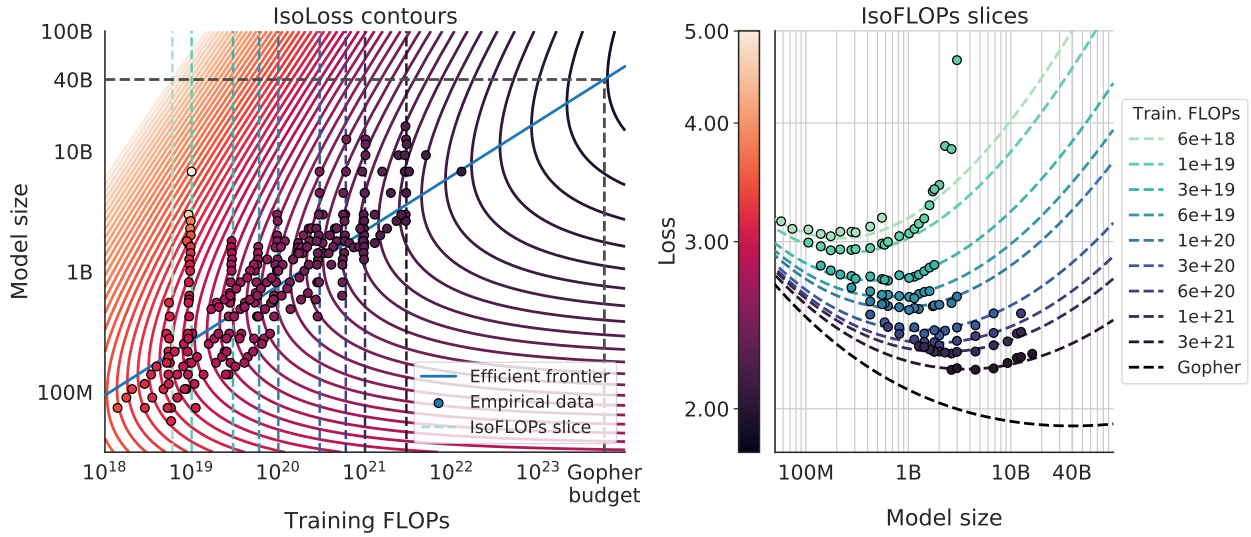

Figure 4 | **Parametric fit.** We fit a parametric modelling of the loss $\hat{L}(N, D)$ and display contour (**left**) and isoFLOP slices (**right**). For each isoFLOP slice, we include a corresponding dashed line in the left plot. In the left plot, we show the efficient frontier in blue, which is a 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 $4 \times d_{\text{model}}$. 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