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.

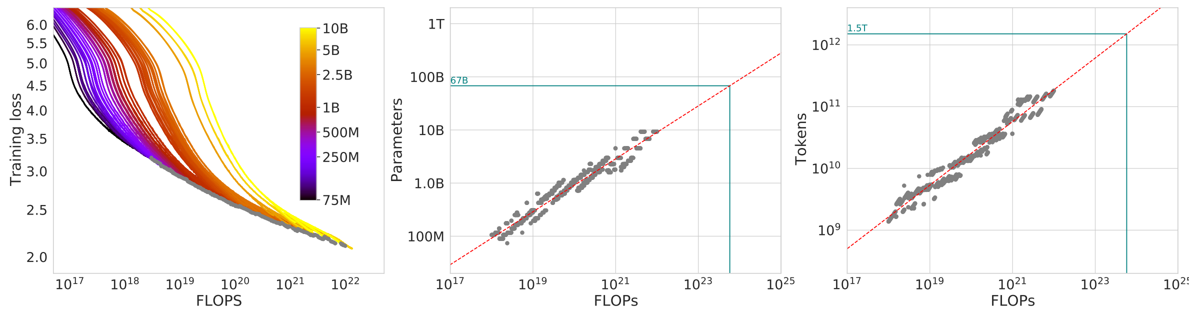

Figure 2 | **Training curve envelope.** On the **left** we show all of our different runs. We launched a range of model sizes going from 70M to 10B, each for four different cosine cycle lengths. From these curves, we extracted the envelope of minimal loss per FLOP, and we used these points to estimate 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.

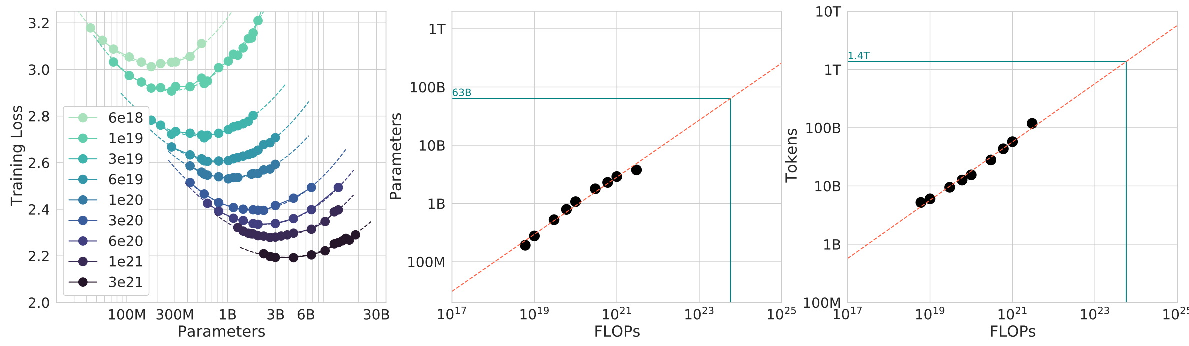

Figure 3 | **IsoFLOP curves.** For various model sizes, we choose the number of training tokens such that the final FLOPs is a constant. The cosine cycle length is set to match the target FLOP count. We find a clear valley in loss, meaning that for a given FLOP budget there is an optimal model to train (**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):

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

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.