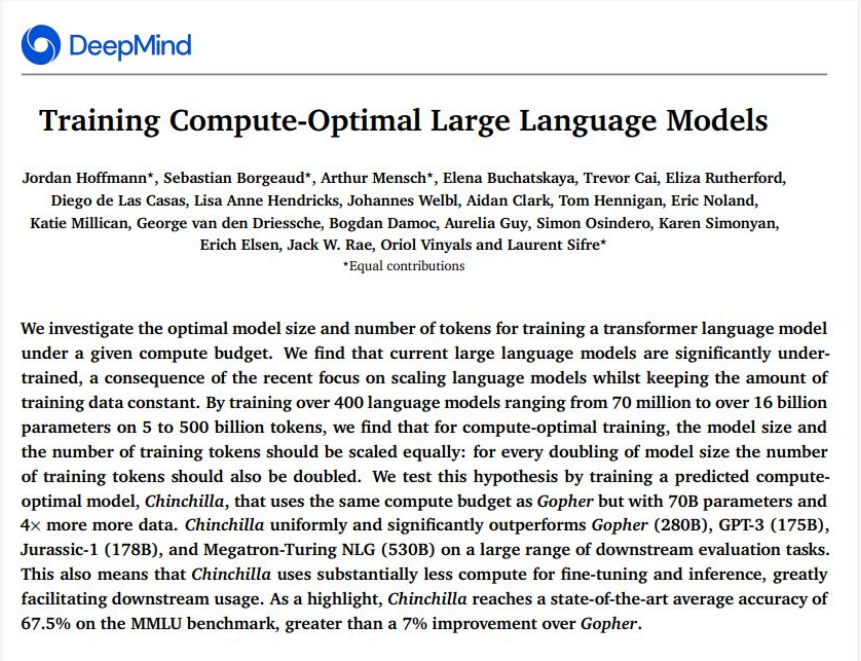
Training Compute-Optimal Large Language Models

Hoffmann et al. (2022) - the Chinchilla paper

Core idea: under compute-optimal training, the model size and number of training tokens should be scaled linearly



DeepMind

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 under-trained, 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*.

James Dill, Michael Fu

In 2022, frontier dense LLMs were mostly huge and undertrained

Most flagship dense models kept scaling parameter count while still training on roughly ~300B tokens.

Model	Params	Tokens
GPT-3	175B	300B
Jurassic-1	178B	300B
Gopher	280B	300B
MT-NLG	530B	270B
Chinchilla	**70B**	**1.4T**

Why this matters

- Training FLOPs are usually fixed in advance by hardware and time.
- Inference cost scales with model size, so a smaller optimal model is doubly attractive.
- If the frontier is wrong, billion-dollar scaling decisions can be misallocated.

Paper's opening claim: current large language models are significantly undertrained.

PROBLEM

How should a fixed training compute budget be allocated?

Choose model size N and training tokens D to minimize loss L(N,D), subject to a fixed compute budget C.

Optimization view

Interpretation

- Bigger N lowers loss, but it makes every token update more expensive.
- More D lowers loss, but only if the model still has capacity to use those tokens.
- The right answer is a trade-off, not "make N as large as possible."

Operational consequence: inference cost grows with N, not D. If a smaller model wins at equal training FLOPs, it is also cheaper to deploy.

Kaplan vs. Chinchilla: two very different scaling recommendations

Earlier view (Kaplan et al. 2020)

- Implication: spend most additional compute on parameters.
- This view encouraged 175B-530B dense models trained on ~300B tokens.

This paper's view (Hoffmann et al. 2022)

- Implication: many huge models were over-sized for their compute budget.
- Spend more training compute on tokens; keep the deployed model smaller.

Central claim to remember: as compute grows, parameters and data should scale almost together.

Why did Hoffmann et al. think the old frontier was wrong?

Their critique is not just a new fit. It is that the earlier experimental protocol systematically favored larger models.

1. **1. Fixed token budgets**
   If every model gets roughly the same number of tokens, larger models are stopped too early and look worse than they would with more data.

2. **2. Mismatched LR schedules**
   The paper argues the cosine decay horizon should roughly match the chosen training length; otherwise short runs are unfairly penalized.

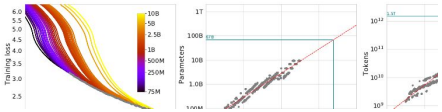
3. **3. Too little joint variation in N and D**
   To answer the allocation question, you need models that vary in both size and training horizon; they train >400 models from 70M to >16B parameters.

Put differently: the earlier recipe may have measured undertraining and mistaken it for a scaling law.

Three complementary estimation strategies

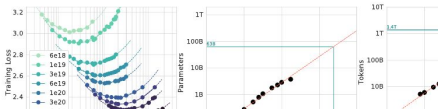
Choose model size N and training tokens D to minimize loss L(N,D), subject to a fixed compute budget C.

Approach 1
Training-curve envelope



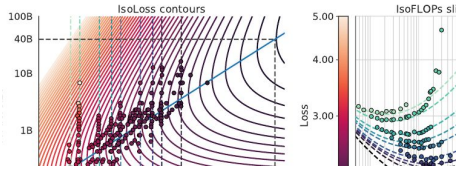
Take the minimum achieved loss at each FLOP level across many runs.

Approach 2
IsoFLOP slices



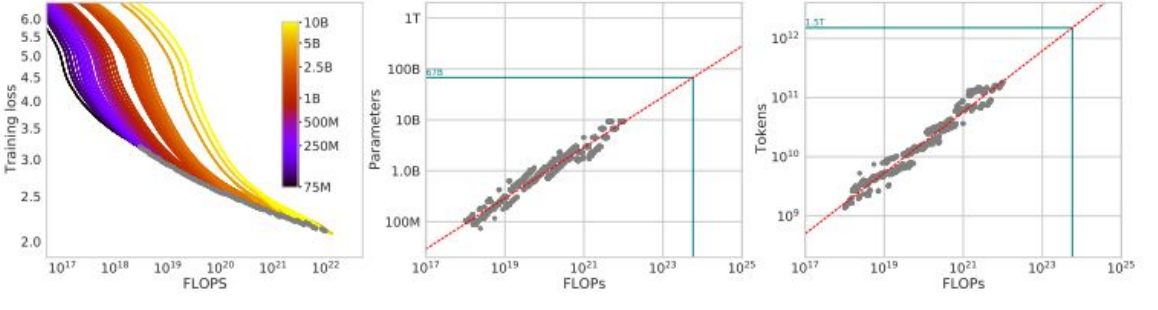
At fixed compute, sweep model size and directly locate the loss minimum.

Approach 3
Parametric loss model



Fit a smooth loss surface L(N,D), then solve for the efficient frontier.

Approach 1: take the best loss achieved at each compute level

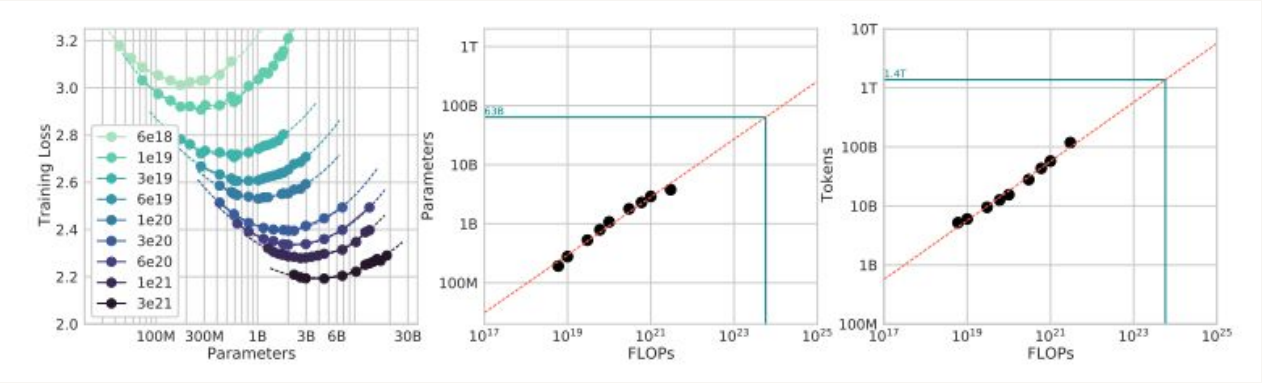


How it works

- Train many model sizes, each with several different training horizons.
- Smooth and interpolate the training curves.
- For every FLOP value, keep whichever run has the lowest loss.
- Fit power laws to the resulting efficient frontier.

Key result here alone:
a = 0.50 and b = 0.50

Approach 2: for each fixed compute budget, find the sweet spot directly



IsoFLOP intuition

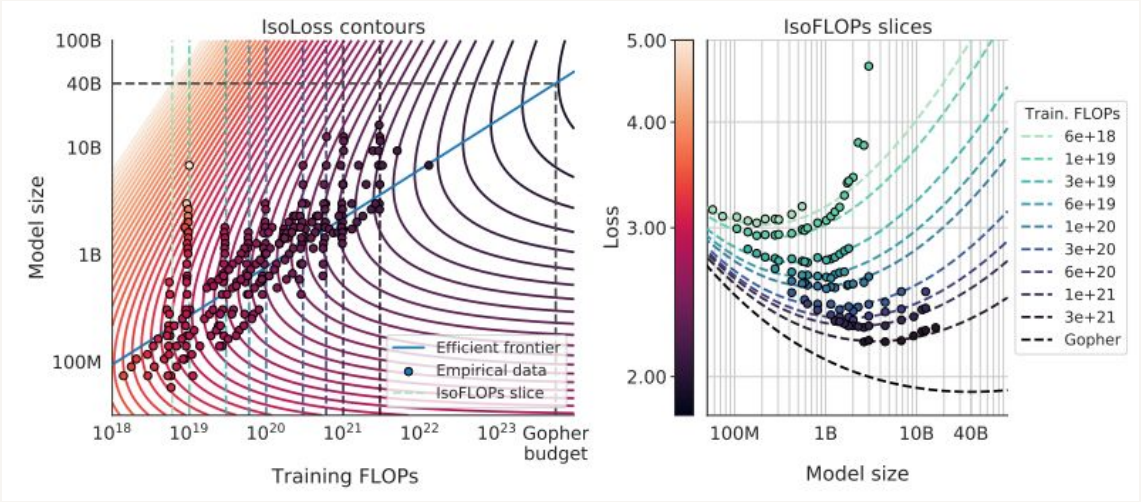
- Hold training FLOPs constant.
- Vary model size; tokens are then determined by that compute budget.
- Each slice is U-shaped: too small underfits, too large is undertrained.
- The bottom of the U is the compute-optimal model size.

This method reaches the same conclusion: N and D should grow in near-equal proportions.

Approach 3: fit a smooth loss surface, then solve for its efficient frontier

Parametric model

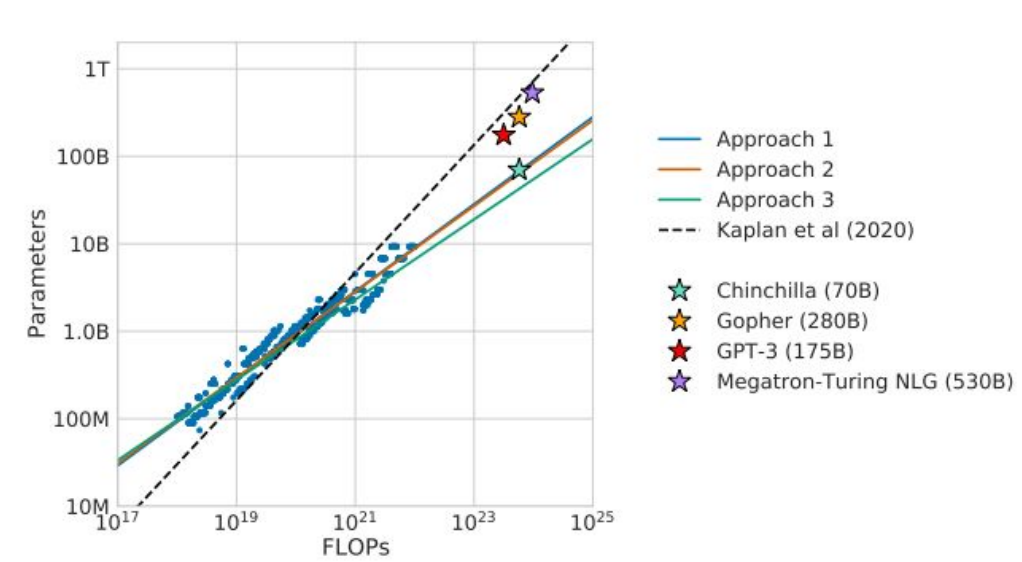
- Interpret loss as an irreducible floor plus separate returns from more parameters and more data.
- Fit the model to many observed runs.
- Then minimize \hat L under FLOPs(N,D) ≈ 6ND.



This approach predicts slightly smaller optima than the other two, but the broad story is still the same.

MAIN FINDING

All three methods converge on the same headline result



What the exponents say

- Approach 1: N_opt ~ C^0.50 and D_opt ~ C^0.50
- Approach 2: N_opt ~ C^0.49 and D_opt ~ C^0.51
- Approach 3: N_opt ~ C^0.46 and D_opt ~ C^0.54
- Kaplan (2020): N_opt ~ C^0.73 and D_opt ~ C^0.27

As compute grows, you should scale parameters and tokens together - not overwhelmingly favor parameters.

The memorable rule of thumb: about 20 tokens per parameter

Table 3 turns the scaling law into concrete training recipes.

Model size	Optimal tokens	Tokens / param
67B	1.5T	22.4
175B	3.7T	21.1
280B	5.9T	21.1
1T	21.2T	21.2

Why this stuck

~20 **training tokens per parameter**

- Easy operational recipe for planning a run.
- Explains why 70B trained long beat 280B trained briefly.
- Made many 2022 models look undertrained in hindsight.

Chinchilla vs. Gopher at equal training FLOPs

Gopher

280B params

300B tokens

Large deployed model
trained relatively briefly

Same training FLOPs

Chinchilla

70B params

1.4T tokens

4x smaller model
trained 4x longer on data

Chinchilla uniformly outperformed Gopher while also lowering memory footprint and inference cost.

What changed in Chinchilla's recipe, and how was it evaluated?

Key recipe differences

- Same 80 layers as Gopher, but half the width: d_model 8192 vs 16384 and 64 vs 128 heads.
- AdamW instead of Adam.
- Modified SentencePiece tokenizer (no NFKC normalization).
- bf16 forward/backward pass with fp32 master weights.

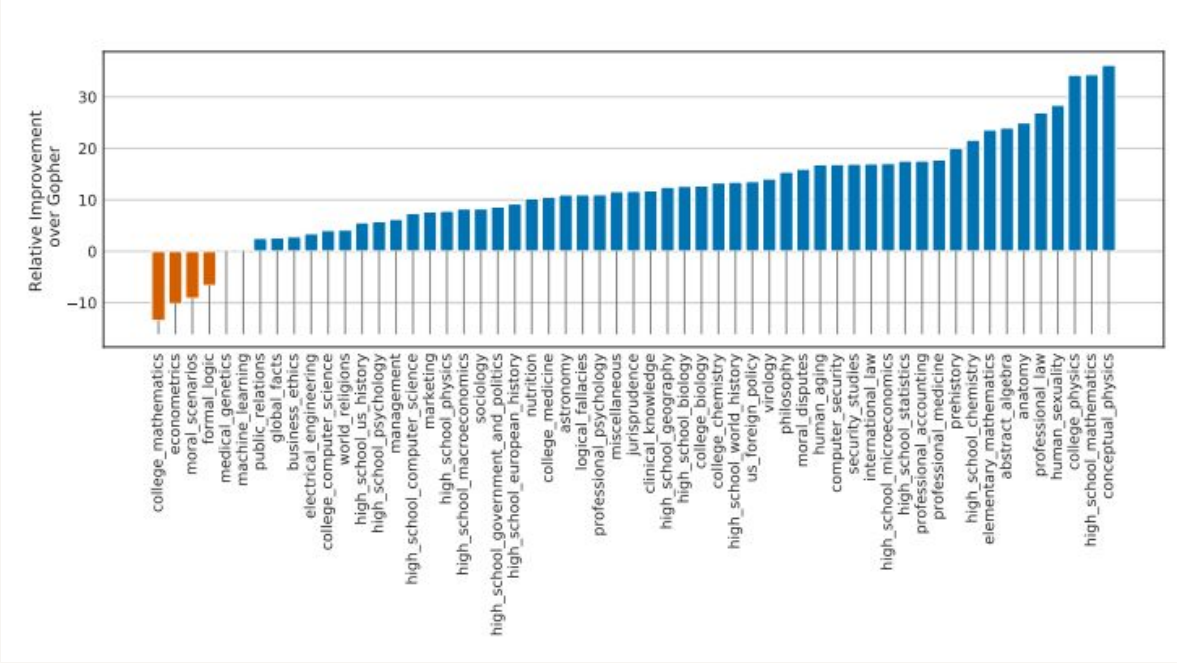
Note that multiple parameters were changed here in addition to compute allaction.

Evaluation tasks

Language modeling	20
Reading comp	3
Question answering	3
Common sense	5
MMLU	57
BIG-bench	62

BIG-bench and MMLU together make for a very comprehensive evaluation benchmark.

MMLU Benchmarks



Notes:

- 67.6% 5-shot average accuracy.
- Improves on Gopher by 7.6 points.
- Better on 51/57 tasks, same on 2, worse on 4.
- Even exceeds the paper's cited June 2023 expert forecast of 63.4%.

Model size isn't as big of a factor as previously thought!

Results overview

Headline numbers

67.6% **MMLU 5-shot (+7.6 over Gopher)**

65.1% **BIG-bench average (+10.7 over Gopher)**

66.7% **TruthfulQA 10-shot (vs 43.7 for Gopher)**

Broad pattern

- Beats Gopher on all Pile evaluation subsets, though the authors caution leakage may inflate those wins.
- Outperforms Gopher on 51/57 MMLU tasks and all but four BIG-bench tasks considered.
- Improves strongly on reading comprehension, especially RACE-h/m.
- Matches or beats Gopher and GPT-3 on every common-sense benchmark reported.

Gains across many categories imply this is an actual improvement.

Reading comprehension and TruthfulQA are also strong evidence

Reading comprehension (Table 7)

Task	Chinchilla	Gopher	GPT-3 / MT-NLG
LAMBADA	**77.4**	74.5	76.2 / 76.6
RACE-m	**86.8**	75.1	58.1 / -
RACE-h	**82.3**	71.6	46.8 / 47.9

On RACE-h and RACE-m, Chinchilla is more than 10 points better than Gopher.

TruthfulQA + common sense

- TruthfulQA: 43.6% / 58.5% / 66.7% at 0/5/10-shot. Gopher gets 29.5% at 0-shot and 43.7% at 10-shot.
- Chinchilla matches or beats Gopher and GPT-3 on every common-sense benchmark reported.
- The authors argue these gains suggest better pretraining alone can improve truthfulness.

Strengths and Limitations

Strengths

• Three distinct estimation approaches point in the same direction.

• The authors do not stop at extrapolation: they run the decisive same-FLOP validation against Gopher.

• The practical implication is unusually clear: better quality and lower inference cost at once.

Limitations / caveats

• The celebrated validation is basically one flagship pair: Chinchilla versus Gopher.
• The frontier is estimated from smaller-scale runs and then extrapolated to flagship budgets.
• Language-modeling comparisons may be inflated by leakage because Chinchilla saw 4x more data.
• More data also intensifies bias, toxicity, and privacy concerns.

Post-paper critiques and improvements

Critiques / Replications

• Epoch AI (2024) argued the paper's parametric-fit estimates appear poorly fit and the reported confidence intervals look too tight.

• That critique mainly targets how Approach 3 was estimated, not the Chinchilla vs Gopher result.

What seems robust

• Porian et al. report that once several protocol issues in Kaplan-style experiments are corrected, results agree well with the Hoffmann / Chinchilla scaling law.

• Scaling data aggressively seems to be more effective than focusing on scaling parameters.

Main takeaways and Q&A

1. Fixed-compute scaling is a resource-allocation problem: the question is not how big a model you can afford, but how to split compute between parameters and data.

2. **For dense autoregressive transformers in this regime, Chinchilla's empirical results imply that most 2022 models were undertrained.**

3. The exact exponents may be debated, but this paper was compelling evidence that shifted the field toward jointly scaling tokens and parameters.

A smaller model, trained much longer, can be the better use of the same compute.