MACHINE LEARNING · EXPLAINER

How Large Language Models Are Trained

From raw text to assistant: the published recipe behind large language models. Data preparation, next-token pretraining, scaling laws, supervised fine-tuning and preference tuning (RLHF, DPO), each with its objective written out.

Intermediate · 9 min · LLM · pretraining · transformer · scaling laws

Sheet
by Simon-Pierre Boucher

sheet.spboucher.ai/s/how-large-language-models-are-trained
November 17, 2026

A large language model is a Transformer [1] whose billions of parameters are tuned in two broad phases. **Pretraining** teaches it to predict the next token over a huge text corpus, and that is where almost all the compute goes. **Post-training** then reshapes it into an assistant with curated demonstrations and human preferences. This sheet follows that pipeline step by step, with the objective optimised at each stage.

01 Step 1: Turn the internet into tokens

Collect

Large web crawls, books, code, encyclopaedic and scientific text. GPT-3 mixed a filtered Common Crawl with curated corpora [3].

Filter

Heuristics and quality classifiers remove spam, boilerplate and low-quality pages. Many pipelines also try to strip toxic content and personal data.

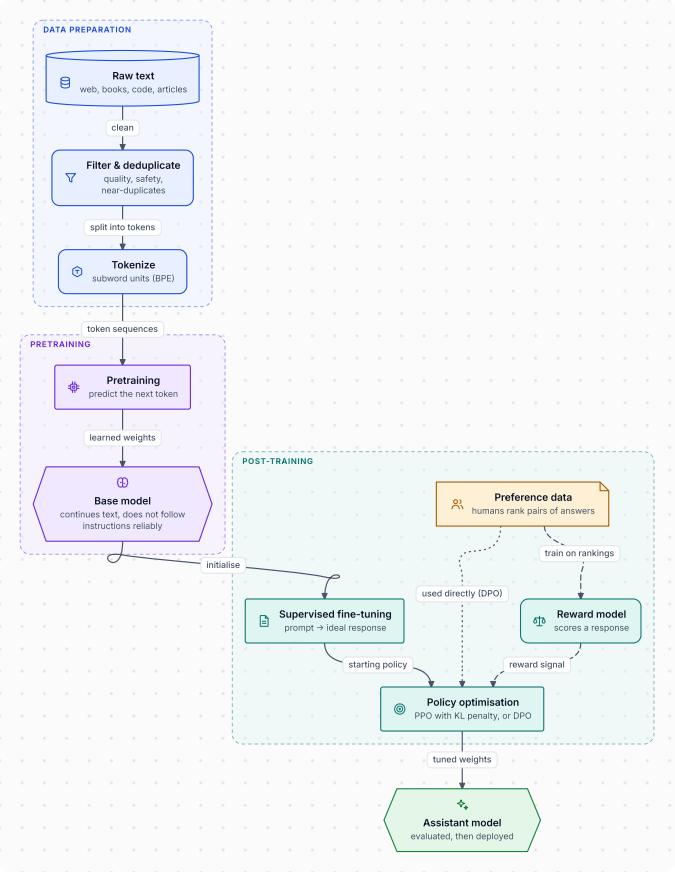
Deduplicate

Removing exact and near-duplicate text reduces verbatim memorisation and can reach the same accuracy in fewer training steps [7].

Tokenize

Text is split into subword units with byte-pair encoding or a variant [6]. The vocabulary typically has tens of thousands of entries, and each token becomes an integer ID.

The canonical training pipeline



Data preparation
Pretraining
Post-training
Human-provided signal
Resulting model
RLHF path (reward model)
DPO path (no reward model)

1 **Prepare the data**
— Collect text, remove low-quality and duplicate documents, and convert everything into token IDs.

2 **Pretrain**
— Predict each next token over hundreds of billions to trillions of tokens. The result is a base model.

3 **Supervised fine-tuning**
— Continue training on written demonstrations of good answers.

4 **Preference tuning**
— Learn from human rankings, either through a reward model and RL (RLHF) or directly (DPO).

5 **Ship**
— Evaluate the tuned model and deploy it as an assistant.

The published recipe (GPT-3, InstructGPT, DPO). Frontier labs add their own variants, which are often undisclosed.

02 Step 2: Pretraining — predict the next token

ƒ NEXT-TOKEN CROSS-ENTROPY

$$\mathcal{L}(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \log p_\theta\left(x_t \mid x_{<t}\right)$$

The model reads a sequence and outputs a probability for every vocabulary entry at each position. The loss is the average negative log-probability it gave to the token that actually came next. No labels are needed, because the text supplies its own targets (self-supervision). The gradient of $\mathcal{L}$ is computed by backpropagation, and θ is updated with an Adam-type optimiser [8] over many batches [2] [3].

θ	All trainable parameters of the Transformer (embeddings, attention and MLP weights)
x_t	Token observed at position t of the training sequence
$x_{<t}$	All tokens before position t (causal context: the model cannot see ahead)
$p_\theta(x_t \mid x_{<t})$	Probability the model assigns to the true next token, from a softmax over the vocabulary
T	Number of predicted tokens in the batch — tokens

Sources: 2 3

ƒ TRAINING COMPUTE

$$C \approx 6\,N\,D$$

Each token costs about $2N$ floating-point operations in the forward pass and about $4N$ in the backward pass [4].

C	Total training compute — FLOPs
N	Number of model parameters (non-embedding in Kaplan et al.)
D	Number of training tokens — tokens

Sources: 4

Scale is planned, not guessed. Test loss falls smoothly, roughly as a power law, as parameters, data and compute grow [4]. GPT-3 has $N \approx 175$ billion parameters and was trained on $D \approx 300$ billion tokens [3], so $6ND \approx 3.2 \times 10^{23}$ FLOPs. Hoffmann et al. found that, for a fixed budget, parameters and tokens should grow together, at roughly **20 tokens per parameter**. Their Chinchilla model (70 B parameters, 1.4 T tokens) outperformed the 4× larger Gopher trained with the same compute [5].

Sources: 4 3 5

What pretraining adjusts: attention weights

Synthetic data — seed 7

Attention weights $\mathrm{softmax}\left((q_i^\top k_j/\sqrt{d} - \lambda\,|i-j|)/\tau\right)$ of a single head. Embeddings (one vector per word plus a positional term) and the Q and K projections are drawn from the seed: the map shows the mechanism, not what a trained model has learned. λ is a simplified relative-position bias; $\tau = 1$ and $\lambda = 0$ give the standard formula.

query ↓ key →	the	cat	sleeps	on	the	mat
the	0.17	0.17	0.13	0.26	0.11	0.17
cat	0.21	0.09	0.06	0.31	0.19	0.14
sleeps	0.09	0.17	0.16	0.32	0.18	0.08
on	0.24	0.04	0.02	0.52	0.11	0.09
the	0.19	0.08	0.26	0.19	0.11	0.16
mat	0.15	0.17	0.25	0.08	0.15	0.20

Mean row entropy (normalised)	Mean weight on itself	Causal mask
0.92	0.208	no

Parameters: Temperature τ = 1; Dimension d = 16; Locality bias λ = 0

The Q and K projections here are random, as they are at initialisation. Pretraining changes these projections, among other weights, until the attention patterns help predict the next token. Vary the temperature to see the softmax sharpen or flatten.

03 Step 3: Post-training — from base model to assistant

A base model continues text, but it does not reliably follow instructions. **Supervised fine-tuning (SFT)** keeps the same next-token loss but trains on pairs of a prompt and a human-written ideal response. The loss is often counted only on the response tokens. **Preference tuning** comes next. Labellers compare two answers, and a reward model r_ϕ learns to score the preferred one y_w above the rejected one y_l by minimising $-\log\sigma\left(r_\phi(x,y_w) - r_\phi(x,y_l)\right)$ [9] [11]. The model is then optimised against that reward with PPO [10]. In InstructGPT, labellers preferred the outputs of a 1.3 B-parameter tuned model to those of the 175 B GPT-3 [11].

Sources: 9 11 10

ƒ KL-REGULARISED PREFERENCE OBJECTIVE

$$\max_{\pi_\theta}\ \mathbb{E}_{x\sim\mathcal{D},\ y\sim\pi_\theta(\cdot|x)}\left[r_\phi(x,y)\right] \;-\; \beta\,\mathbb{D}_{\mathrm{KL}}\left[\pi_\theta(y \mid x)\,\|\,\pi_{\mathrm{ref}}(y \mid x)\right]$$

Seek high reward without drifting too far from the reference model. Without the KL term, the policy could exploit flaws in the reward model. RLHF optimises this objective with reinforcement learning (PPO) [11]. DPO shows that the same objective can be optimised directly on preference pairs, with a simple classification-style loss and no separate reward model or RL loop [12].

π_θ	The model being tuned (the policy), which generates response y to prompt x
π_{ref}	Frozen reference model, usually the SFT model
$r_\phi(x,y)$	Reward model score for response y to prompt x, learned from human rankings
$\mathcal{D}$	Distribution of training prompts
β	Strength of the penalty for diverging from the reference model
$\mathbb{D}_{\mathrm{KL}}$	Kullback–Leibler divergence between the tuned and reference output distributions

Sources: 11 12

Three stages, three kinds of data. Figures are from the cited papers.

	Pretraining (MOST COMPUTE)	Supervised fine-tuning	Preference tuning (RLHF / DPO)
DATA	Raw, unlabelled text	Prompts with human-written answers	Prompts with pairs of answers ranked by people
OBJECTIVE	Next-token cross-entropy	Same loss, on demonstration responses	Reward under a KL penalty (PPO), or the DPO loss
SIZE	Hundreds of billions to trillions of tokens (GPT-3: 300 B; Chinchilla: 1.4 T)	Orders of magnitude smaller (tens of thousands of prompts in InstructGPT)	Similar order to SFT in InstructGPT
WHAT CHANGES	Language, facts and skills absorbed from the corpus	Format and instruction following	Which answers the model favours: helpfulness, tone, refusals

VERDICT

Pretraining uses by far the most compute; InstructGPT's SFT and PPO stages cost a small fraction of GPT-3's pretraining. Post-training is cheap by comparison, yet it changes how the model behaves for users the most.

Sources: 3 5 11

WARNING

Limits of this picture

This is the **published** recipe. Training details of most frontier models are only partly disclosed, and newer stages vary between labs: AI-generated feedback, reinforcement learning on tasks with checkable answers, long-context and tool-use training. Also, a trained model is not a searchable database of its corpus, but it can memorise and reproduce some training text verbatim. That is one reason deduplication matters [7].

QUESTION

You want to train a 10-billion-parameter model compute-optimally, in the Chinchilla sense. Roughly how many training tokens should you plan for?

A. About 10 billion (one token per parameter)

B. About 200 billion

C. About 10 trillion

D. It doesn't matter, only the parameter count counts

SHOW THE ANSWER

A — Incorrect. Far too few: the model would be heavily under-trained for its size.

B — Correct. Right: about 20 tokens per parameter, so 20 × 10 B ≈ 200 B tokens. The budget is then $C \approx 6 \times 10^{10} \times 2{\times}10^{11} \approx 1.2{\times}10^{22}$ FLOPs.

C — Incorrect. That is about 1,000 tokens per parameter. Training that long can still help, especially to make inference cheaper, but it is not the compute-optimal point.

D — Incorrect. Scaling-law studies show that loss depends on both N and D. Neglecting data wastes compute.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin (2017). *Attention Is All You Need.* Advances in Neural Information Processing Systems (NeurIPS). arxiv.org/abs/1706.03762 UNVERIFIED

[2] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever (2018). *Improving Language Understanding by Generative Pre-Training.* OpenAI technical report. UNVERIFIED

[3] Tom B. Brown, et al. (2020). *Language Models are Few-Shot Learners.* Advances in Neural Information Processing Systems (NeurIPS). arxiv.org/abs/2005.14165 UNVERIFIED

[4] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, Dario Amodei (2020). *Scaling Laws for Neural Language Models.* arXiv. arxiv.org/abs/2001.08361 UNVERIFIED

[5] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, et al. (2022). *Training Compute-Optimal Large Language Models.* Advances in Neural Information Processing Systems (NeurIPS). arxiv.org/abs/2203.15556 UNVERIFIED

[6] Rico Sennrich, Barry Haddow, Alexandra Birch (2016). *Neural Machine Translation of Rare Words with Subword Units.* Proceedings of ACL 2016. arxiv.org/abs/1508.07909 UNVERIFIED

[7] Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, Nicholas Carlini (2022). *Deduplicating Training Data Makes Language Models Better.* Proceedings of ACL 2022. arxiv.org/abs/2107.06499 UNVERIFIED

[8] Diederik P. Kingma, Jimmy Ba (2015). *Adam: A Method for Stochastic Optimization.* International Conference on Learning Representations (ICLR). arxiv.org/abs/1412.6980 UNVERIFIED

[9] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, Dario Amodei (2017). *Deep Reinforcement Learning from Human Preferences.* Advances in Neural Information Processing Systems (NeurIPS). arxiv.org/abs/1706.03741 UNVERIFIED

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov (2017). *Proximal Policy Optimization Algorithms.* arXiv. arxiv.org/abs/1707.06347 UNVERIFIED

[11] Long Ouyang, Jeff Wu, Xu Jiang, et al. (2022). *Training Language Models to Follow Instructions with Human Feedback.* Advances in Neural Information Processing Systems (NeurIPS). arxiv.org/abs/2203.02155 UNVERIFIED

[12] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, Chelsea Finn (2023). *Direct Preference Optimization: Your Language Model is Secretly a Reward Model.* Advances in Neural Information Processing Systems (NeurIPS). arxiv.org/abs/2305.18290 UNVERIFIED