Attention Recap, LLM Training and Generation

Robert Minneker

2026-02-10

Sources

Content derived from: J&M Ch. 8 (Transformers), Ch. 7 (Large Language Models)

Administrivia

- Project checkpoints due Thursday!

Learning Objectives

By the end of this lecture, you will be able to:

1. Implement attention operations
2. Explain the self-supervised pretraining objective for LLMs
3. Describe what makes a Large Language Model “Large”

Recap: Attention Essentials

The Transformer architecture

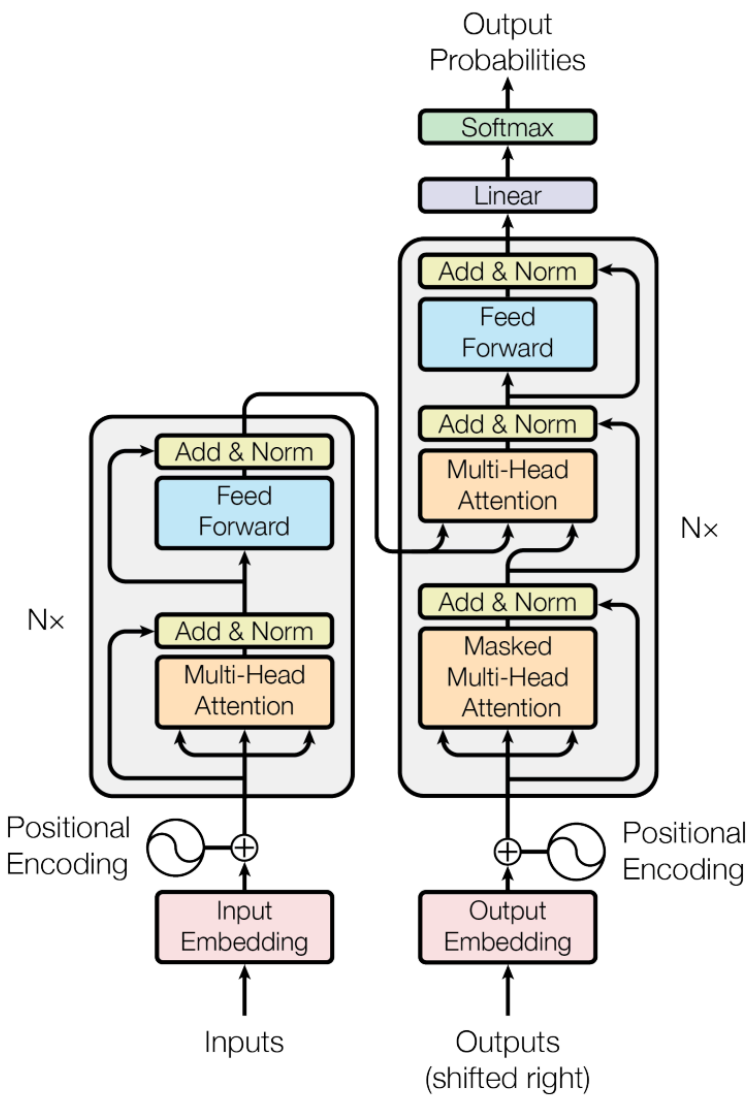


Figure 1: The Transformer - model architecture.

Key components from last time:

- Self-attention with Q, K, V projections
- Multi-head attention for diverse relationships
- Causal masking for autoregressive generation
- Residual connections and layer normalization

Recall: Scaled dot-product attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Q		K^T		Scores		softmax		V		Output
$[n \times d_k]$	$\times$	$[d_k \times n]$	$=$	$[n \times n]$	$\rightarrow$	$[n \times n]$	$\times$	$[n \times d_v]$	$=$	$[n \times d_v]$

- **Q, K, V** are linear projections of the input: $Q = XW^Q$, $K = XW^K$, $V = XW^V$
- Scaling by $\sqrt{d_k}$ prevents softmax saturation
- Output for each position = weighted sum of all value vectors

Recall: Why scale by $\sqrt{d_k}$? Preventing softmax saturation

For random vectors $\mathbf{q}, \mathbf{k} \in \mathbb{R}^{d_k}$ with entries $\sim \mathcal{N}(0, 1)$: $\text{Var}(\mathbf{q} \cdot \mathbf{k}) = d_k$

Without Scaling ($d_k = 64$)

Dot products have std dev ≈ 8

```
scores = [12, 8, 5, 1]
softmax ≈ [0.982, 0.018, 0.0, 0.0]
```

Near one-hot → vanishing gradients for non-max tokens

With Scaling: scores / √64 = scores / 8

Dot products rescaled to std dev ≈ 1

```
scaled = [1.5, 1.0, 0.625, 0.125]
softmax ≈ [0.439, 0.267, 0.183, 0.111]
```

Smooth distribution → healthy gradients for all tokens

Why $\sqrt{d_k}$? The variance argument

Assume $q_i, k_i \sim$ i.i.d. with mean 0, variance 1. The dot product is:

$$q \cdot k = \sum_{i=1}^{d_k} q_i \, k_i$$

Each term $q_i k_i$ has: $\mathbb{E}[q_i k_i] = 0, \;\; \mathrm{Var}(q_i k_i) = 1$

By independence, the variance of the sum is:

$$\mathrm{Var}(q \cdot k) = \sum_{i=1}^{d_k} \mathrm{Var}(q_i k_i) = d_k$$

So dot products grow as $O(d_k)$. Dividing by $\sqrt{d_k}$ restores unit variance:

$$\mathrm{Var}\left(\frac{q \cdot k}{\sqrt{d_k}} \right) = \frac{d_k}{d_k} = 1$$

Recall: Multi-head attention and causal masking

Multi-head: Run h parallel attention operations with $d_k = d_{model}/h$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

Causal masking: Prevent attending to future tokens

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top + M}{\sqrt{d_k}}\right)V$$

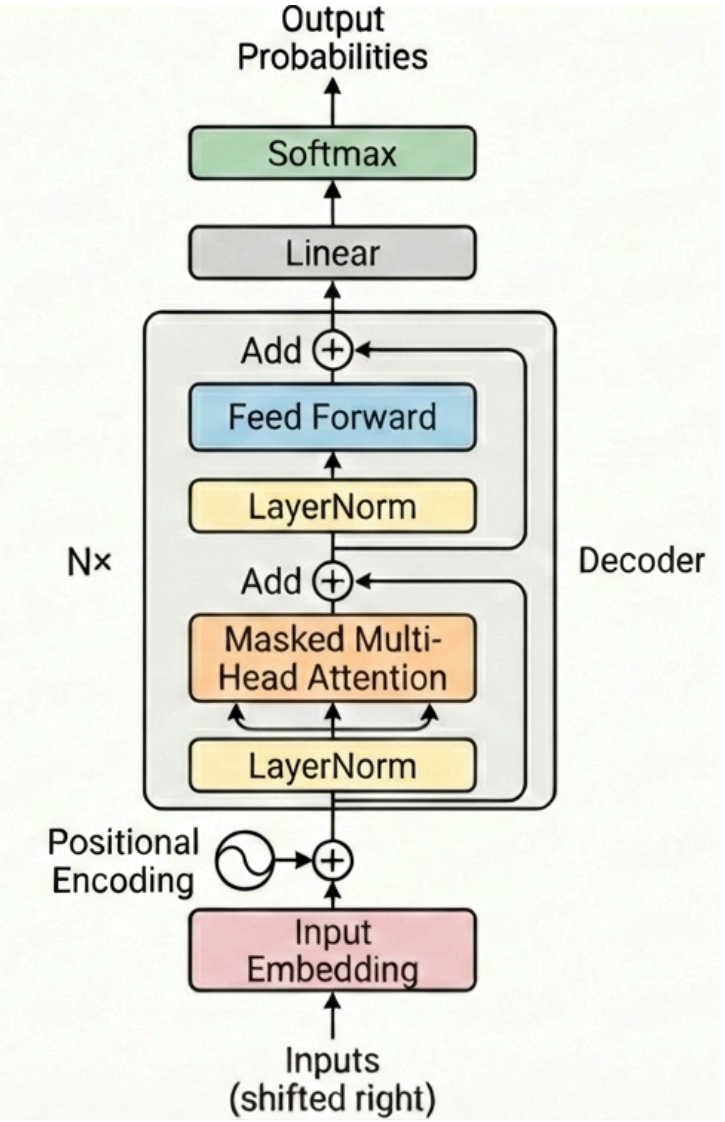
$$\text{where } M_{ij} = \begin{cases} 0 & j \leq i \\ -\infty & j > i \end{cases}$$

Multi-head
Each head learns different relationships

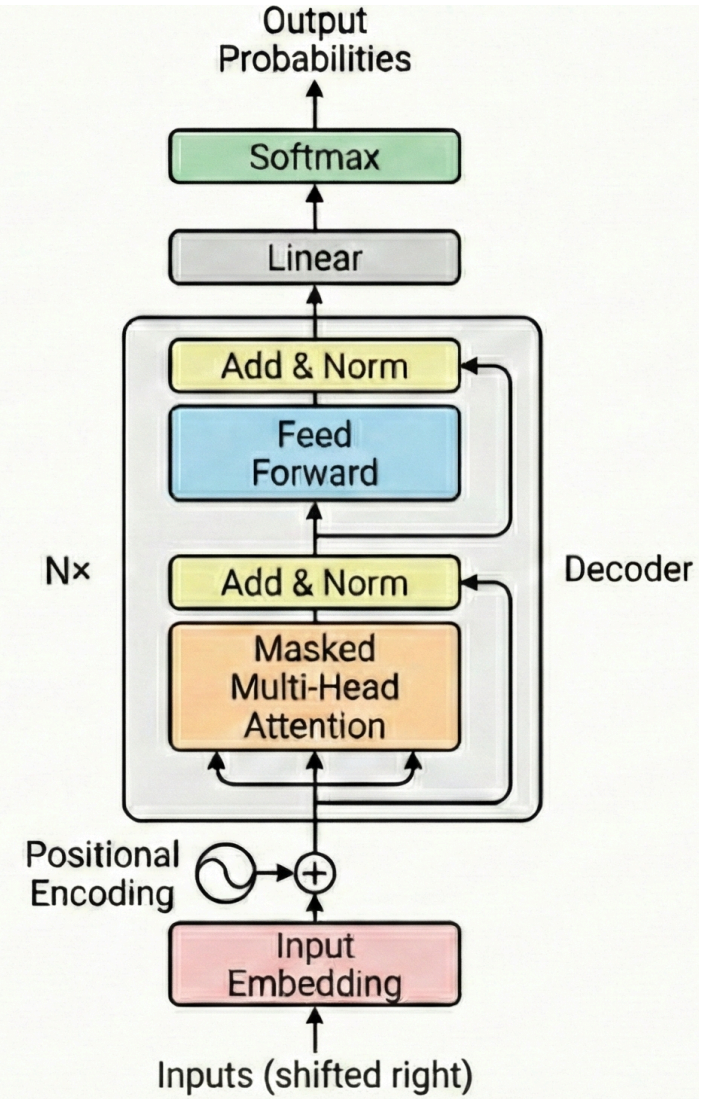
Causal mask
Lower-triangular: token i sees only ≤ i

Transformer block
Attn + FFN + residuals + LayerNorm

The decoder-only transformer block

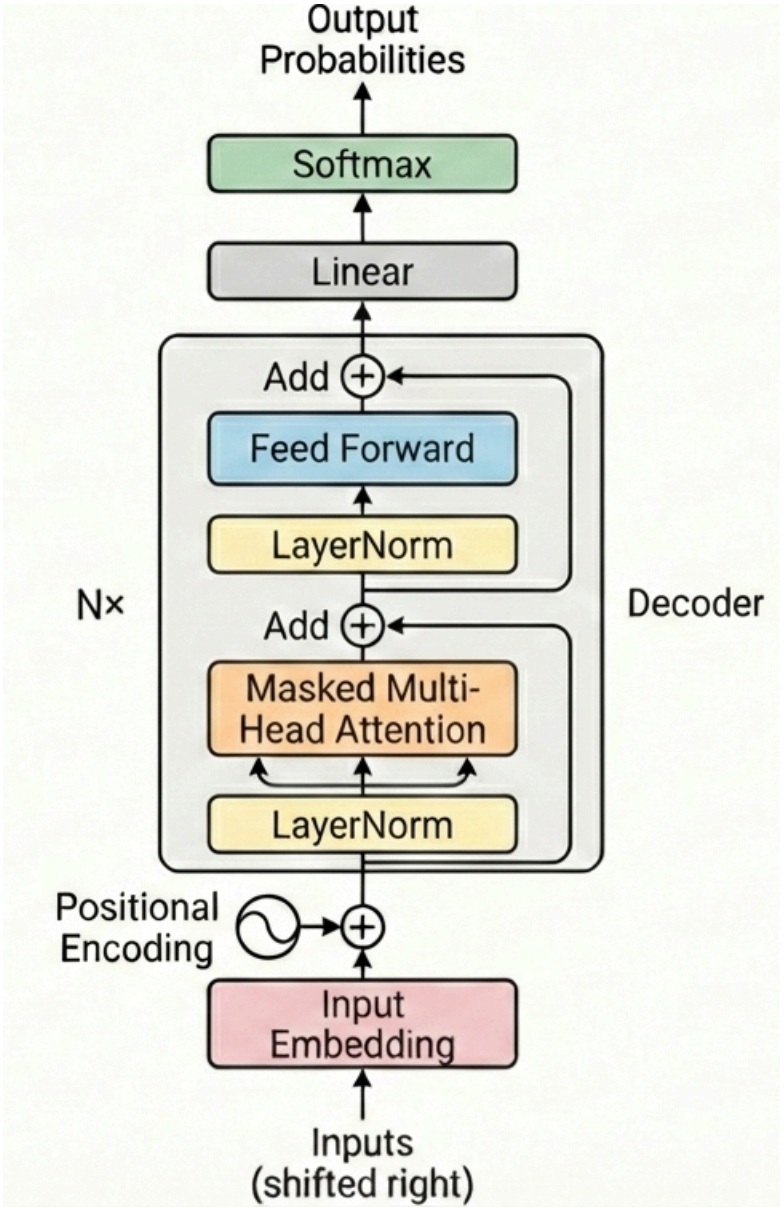


Pre-Norm



Post-Norm

The Pre-Norm Decoder-only Transformer



Part 1: Implementing Self-Attention

Tensor dimensions throughout attention

Input: $X \in \mathbb{R}^{B \times n \times d_{model}}$

Step	Operation	Shape
Input	`X`	`(B, n_tok, d_model)`
Project Q	`Q = X @ W_Q`	`(B, n_tok, d_model)`
Attention scores	`A = Q @ K.T`	**`(B, n_tok, n_tok)`**
Scale + Softmax	`A = softmax(A / √d)`	`(B, n_tok, n_tok)`
Output	`O = A @ V`	`(B, n_tok, d_model)`

- **B** = batch size, **n_tok** = sequence length, **d_model** = embedding dimension
- The attention matrix is **n_tok** × **n_tok**—this is where $O(n^2)$ comes from

Attenion as a block diagram

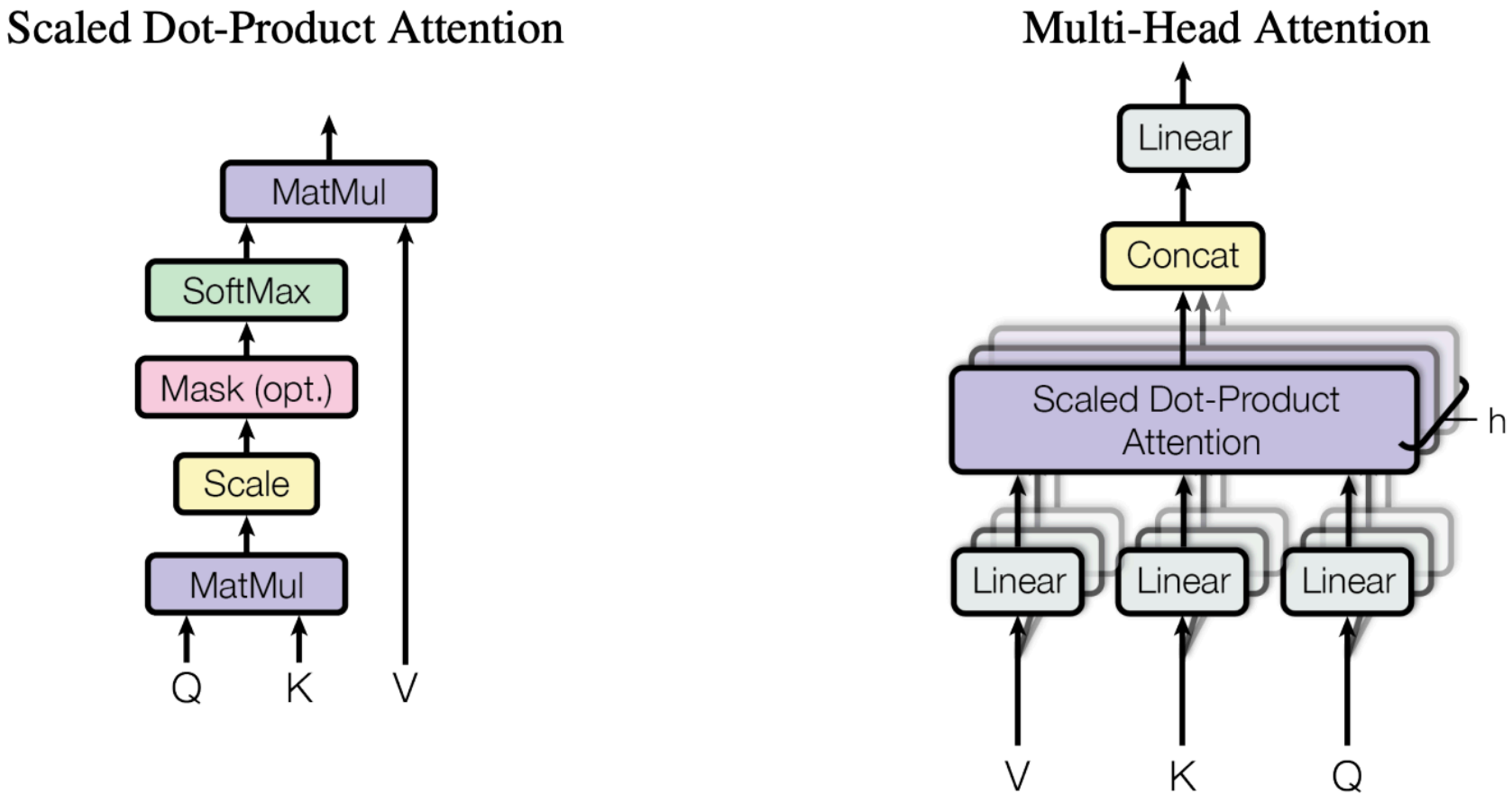


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Multi-head attention — why split heads?

Problem: Single attention head has limited expressiveness

Solution: Run h parallel attention operations with smaller dimensions

Single Head		8 Heads
`d_model = 512` `1 attention operation`	→	`d_head = 512/8 = 64` `8 parallel attention ops`

- Same total parameters: one 512-dim head ≈ eight 64-dim heads
- Each head can learn different relationship types

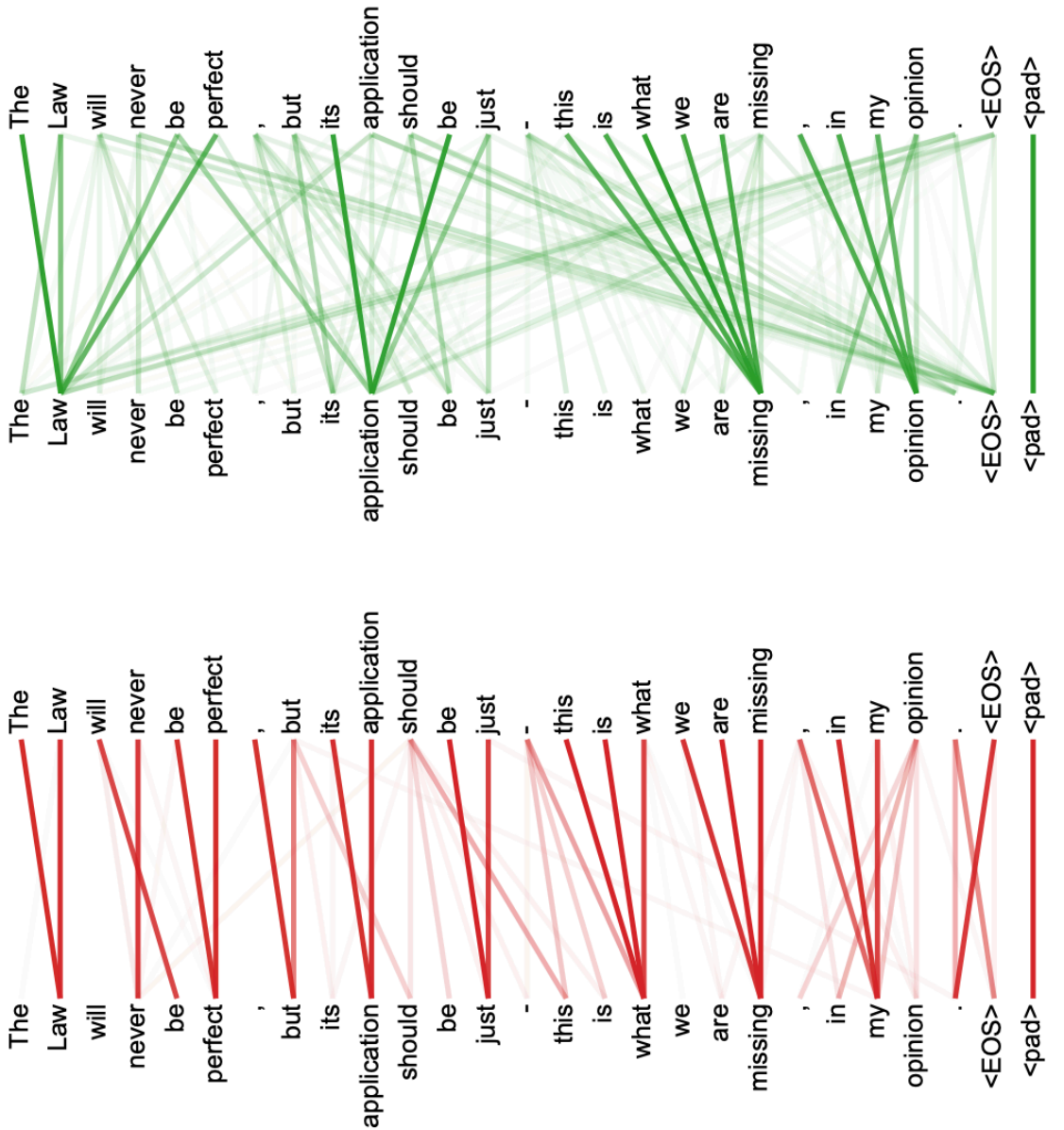


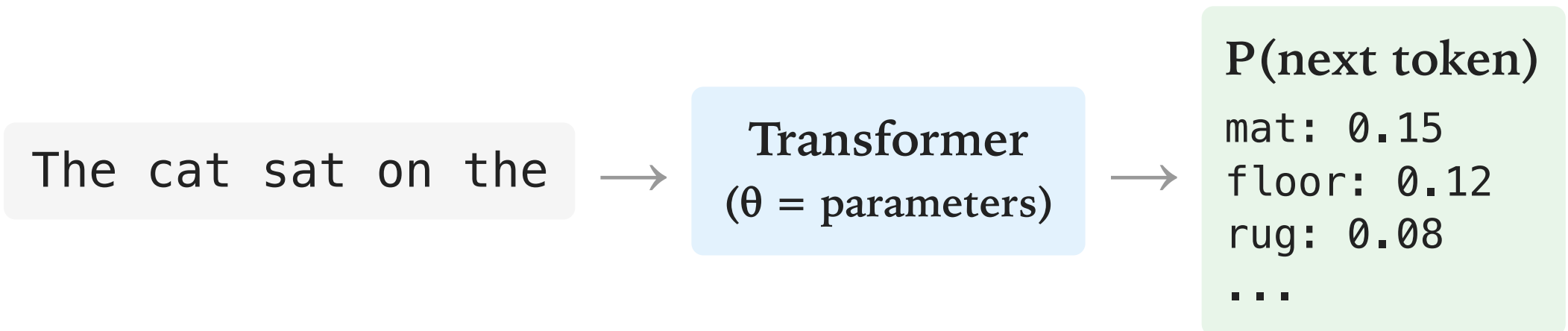
Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.

Part 2: What Makes a Language Model “Large”

A language model predicts probability distributions over tokens

Core function:

$$P(x_t \mid x_1, \ldots, x_{t-1}; \theta)$$



The model maps context → probability distribution over vocabulary

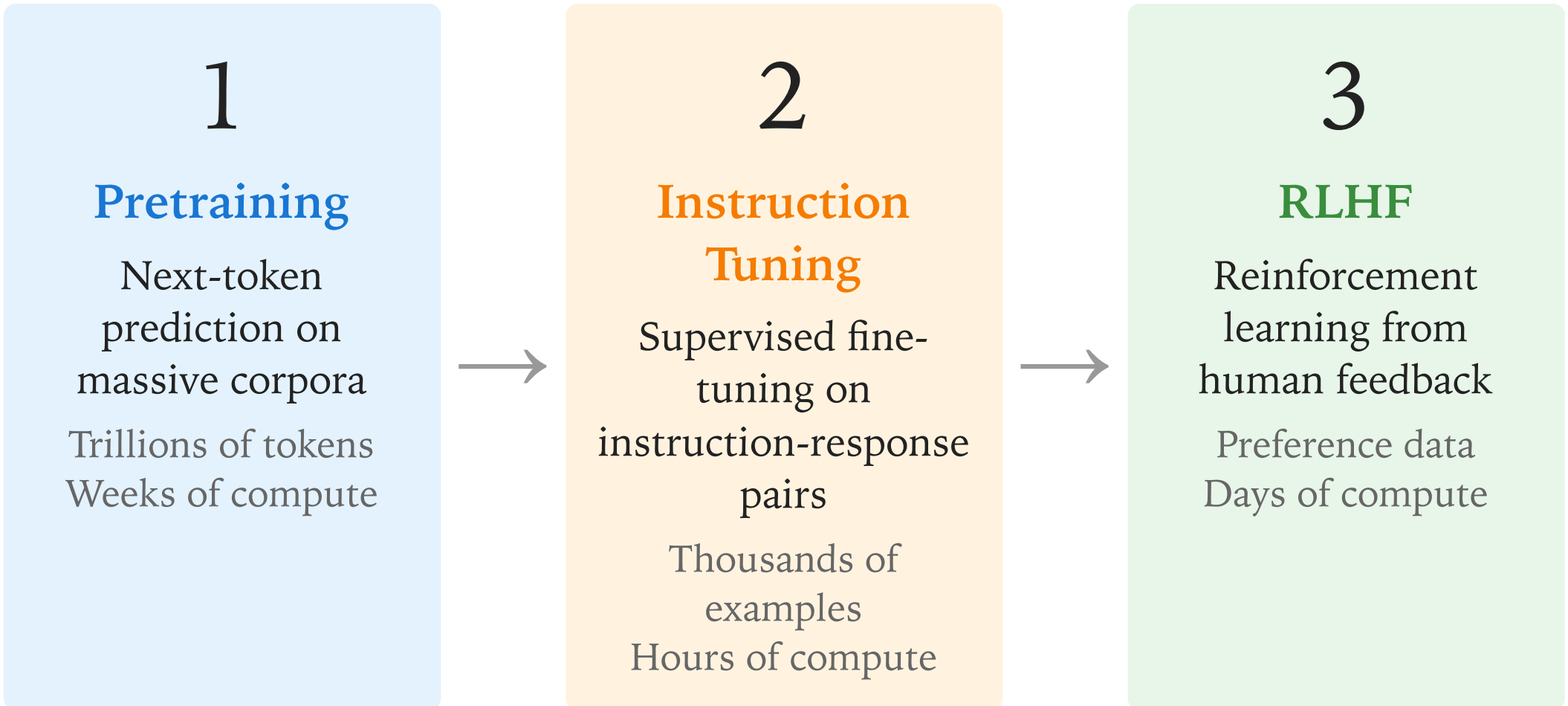
“Large” refers to both parameters and training data

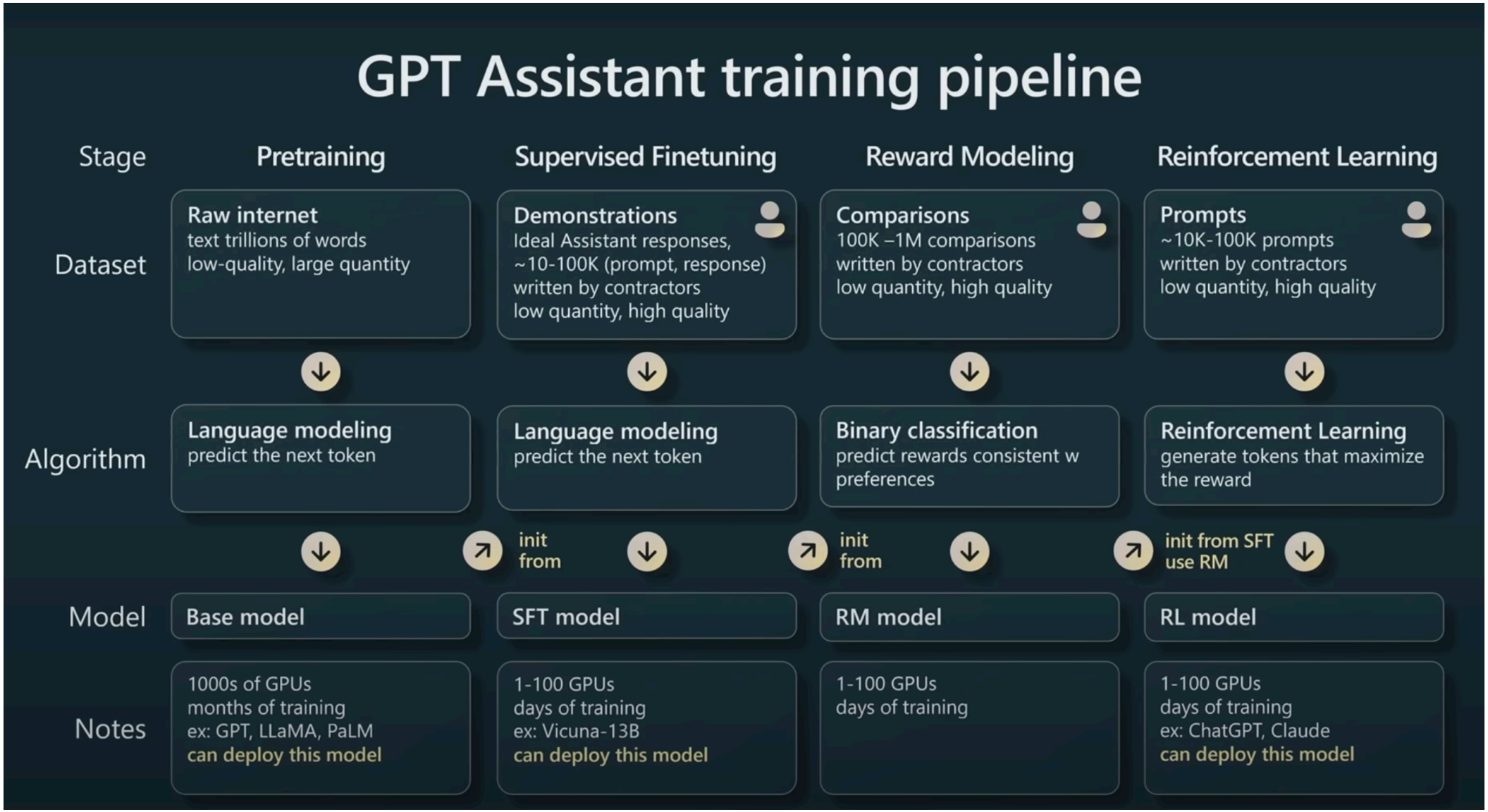
Model	Parameters	Training Tokens	Year
GPT-2	1.5B	40B	2019
GPT-3	175B	300B	2020
LLaMA-2	70B	2T	2023
GPT-4	~1.8T*	~13T*	2023

*Estimated, not officially disclosed

Scale has grown ~1000× in 4 years

The three-stage training pipeline





Slide from Andrej Karpathy: State of GPT

Part 3: Pretraining at Scale

The pretraining objective: predict the next token

Causal Language Modeling (CLM) loss:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{T} \log P_\theta(x_t \mid x_1, \ldots, x_{t-1})$$

Training Example: "The quick brown fox jumps"

Context	Target	Loss contribution
<bos>	The	$-\log$ P(The | <bos>)
The	quick	$-\log$ P(quick | The)
The quick	brown	$-\log$ P(brown | The quick)
The quick brown	fox	$-\log$ P(fox | The quick brown)
The quick brown fox	jumps	$-\log$ P(jumps | ...)

Self-supervision: the data labels itself

Traditional Supervised Learning

Image → Human labels "cat"

Requires expensive manual annotation

Self-Supervised (LLMs)

"The cat" → Next word is "sat"

Labels come from the text itself—free and unlimited

Key insight: The structure of language provides free supervision at massive scale

Next-token prediction implicitly learns many skills

To predict the next token well, the model must learn:

Grammar & Syntax

"She runs" not "She run"

World Knowledge

"Paris is the capital of France"

Reasoning Patterns

"If A then B. A is true. Therefore B"

Style & Tone

Formal vs. casual, technical vs. simple

The simple objective captures complex structure

Language Models are Unsupervised Multitask Learners

Alec Radford [*1] **Jeffrey Wu** [*1] **Rewon Child** [1] **David Luan** [1] **Dario Amodei** [**1] **Ilya Sutskever** [**1]

Abstract

Natural language processing tasks, such as question answering, machine translation, reading comprehension, and summarization, are typically approached with supervised learning on task-specific datasets. We demonstrate that language models begin to learn these tasks without any explicit supervision when trained on a new dataset of millions of webpages called WebText. When conditioned on a document plus questions, the answers generated by the language model reach 55 F1 on the CoQA dataset - matching or exceeding the performance of 3 out of 4 baseline systems without using the 127,000+ training examples. The capacity of the language model is essential to the success of zero-shot task transfer and increasing it improves performance in a log-linear fashion across tasks. Our largest model, GPT-2, is a 1.5B parameter Transformer that achieves state of the art results on 7 out of 8 tested language modeling datasets in a zero-shot setting but still underfits WebText. Samples from the model reflect these improvements and contain coherent paragraphs of text. These findings suggest a promising path towards building language processing systems which learn to perform tasks from their naturally occurring demonstrations.

competent generalists. We would like to move towards more general systems which can perform many tasks – eventually without the need to manually create and label a training dataset for each one.

The dominant approach to creating ML systems is to collect a dataset of training examples demonstrating correct behavior for a desired task, train a system to imitate these behaviors, and then test its performance on independent and identically distributed (IID) held-out examples. This has served well to make progress on narrow experts. But the often erratic behavior of captioning models (Lake et al., 2017), reading comprehension systems (Jia & Liang, 2017), and image classifiers (Alcorn et al., 2018) on the diversity and variety of possible inputs highlights some of the shortcomings of this approach.

Our suspicion is that the prevalence of single task training on single domain datasets is a major contributor to the lack of generalization observed in current systems. Progress towards robust systems with current architectures is likely to require training and measuring performance on a wide range of domains and tasks. Recently, several benchmarks have been proposed such as GLUE (Wang et al., 2018) and decaNLP (McCann et al., 2018) to begin studying this.

Multitask learning (Caruana, 1997) is a promising framework for improving general performance. However, multitask training in NLP is still nascent. Recent work reports modest performance improvements (Yogatama et al.

GPT-2 Paper

Scaling laws describe predictable improvement with resources

Empirical finding (Kaplan et al., 2020; Hoffmann et al., 2022):

$$L(N, D) \approx \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{D_c}{D}\right)^{\alpha_D} + L_\infty$$

N = Parameters

More parameters → lower loss

D = Data (tokens)

More data → lower loss

C = Compute

More FLOPs → lower loss

Loss decreases as a **power law** in each resource

Scaling law curves (Kaplan et al., 2020)

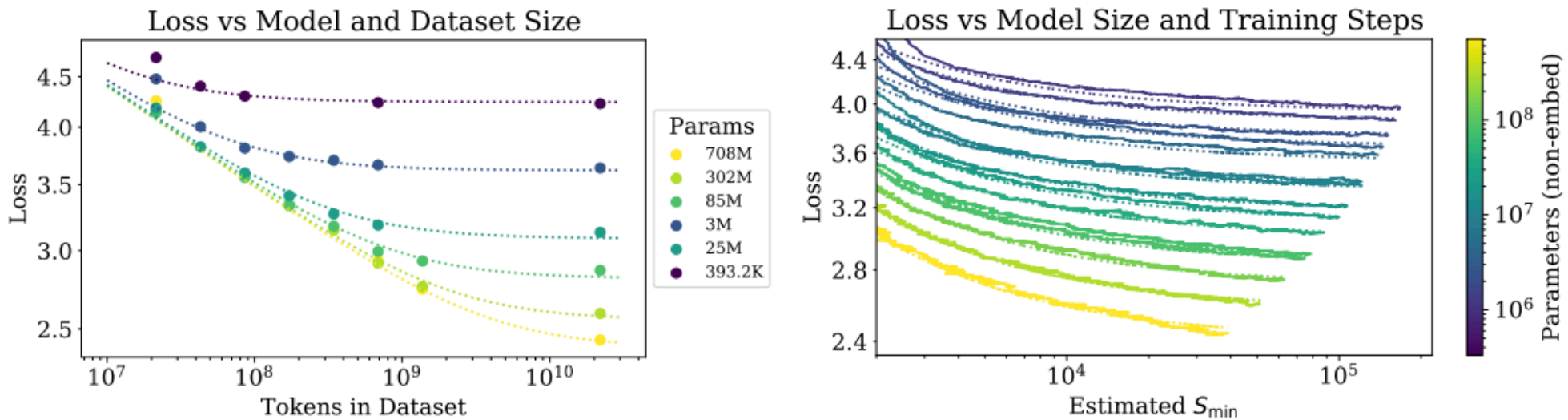


Figure 4 **Left**: The early-stopped test loss $L(N, D)$ varies predictably with the dataset size D and model size N according to Equation (1.5). **Right**: After an initial transient period, learning curves for all model sizes N can be fit with Equation (1.6), which is parameterized in terms of $S_{\min}$, the number of steps when training at large batch size (details in Section 5.1).

Chinchilla scaling: balance parameters and data

Key insight (Hoffmann et al., 2022):

For a fixed compute budget, there's an optimal ratio:

$$N_{\text{opt}} \propto C^{0.5}, \quad D_{\text{opt}} \propto C^{0.5}$$

Rule of thumb: Train on ~20 tokens per parameter

Undertrained	Compute-Optimal	Inference-Optimal
70B params, 300B tokens	70B params, 1.4T tokens	7B params, 2T tokens
GPT-3 (4 tokens/param)	Chinchilla (20 tokens/param)	LLaMA (285 tokens/param)