Training Large Language Models

CS 4644 / 7643: Deep Learning

Slides originally by: William Held, With Materials/Layout from: Roshan Sharma, Claude

- **Assignment 3 out**
  - Due **March 14th 11:59pm EST**

- Quiz **March 18th**

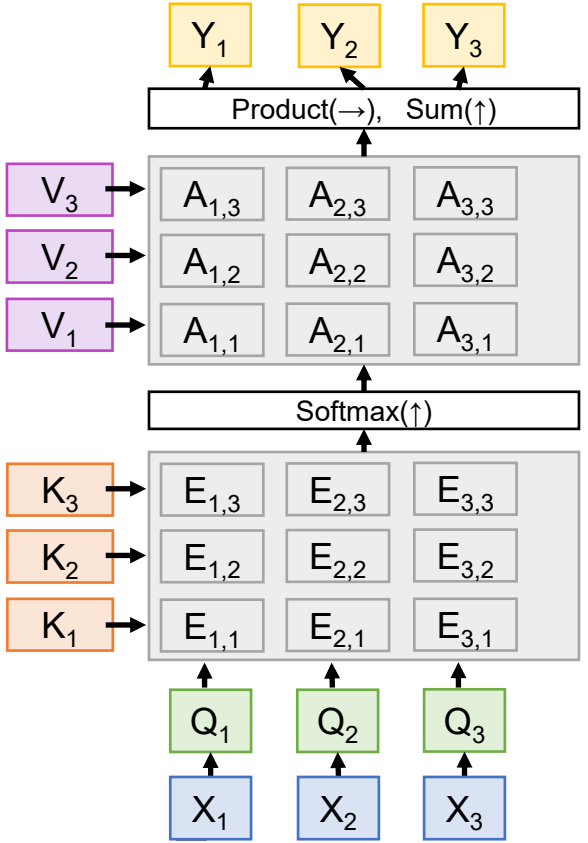
- Project milestone
  - Due ~~**March 14**~~ **March 20th 11:59pm EST**

Self-Attention Layer

Inputs:
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: W_Q (Shape: $D_X \times D_Q$)

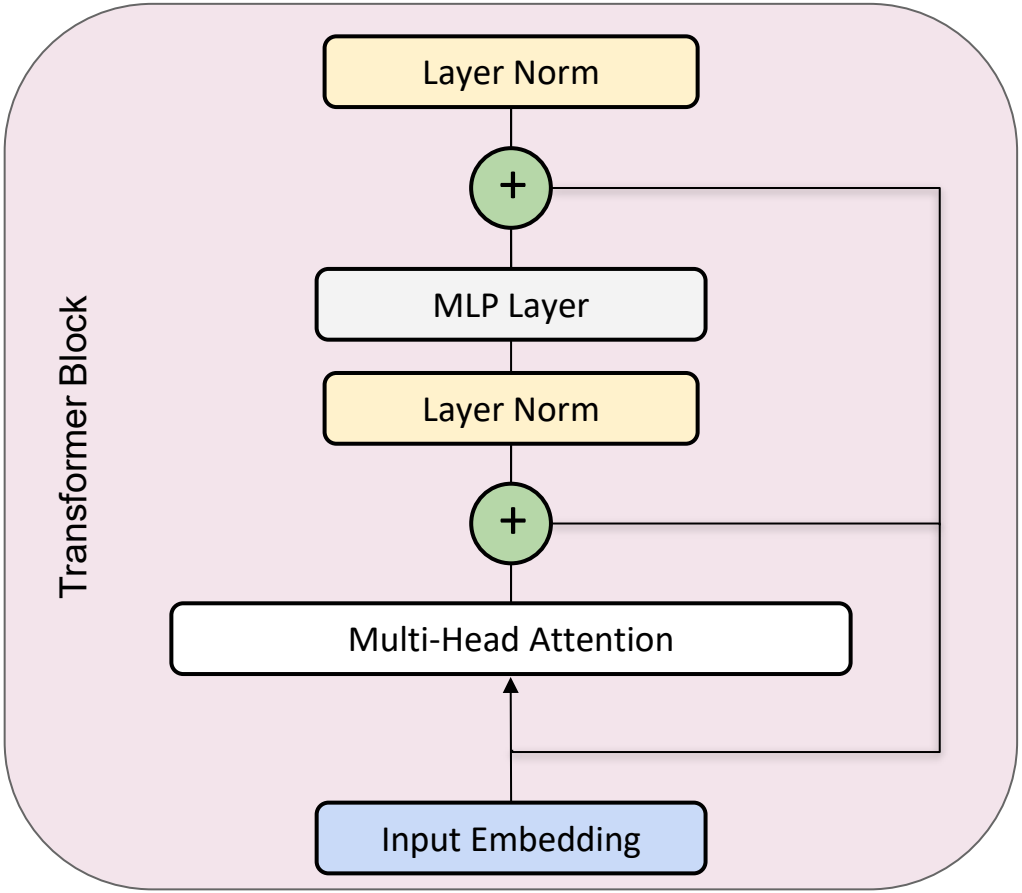
Self attention doesn't "know" the order of the vectors it is processing!

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T$ (Shape: $N_X \times N_X$) $E_{i,j} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

Transformer Lecture Speed Recap: The Transformer Block



LLM Timeline: 2017 – 2025

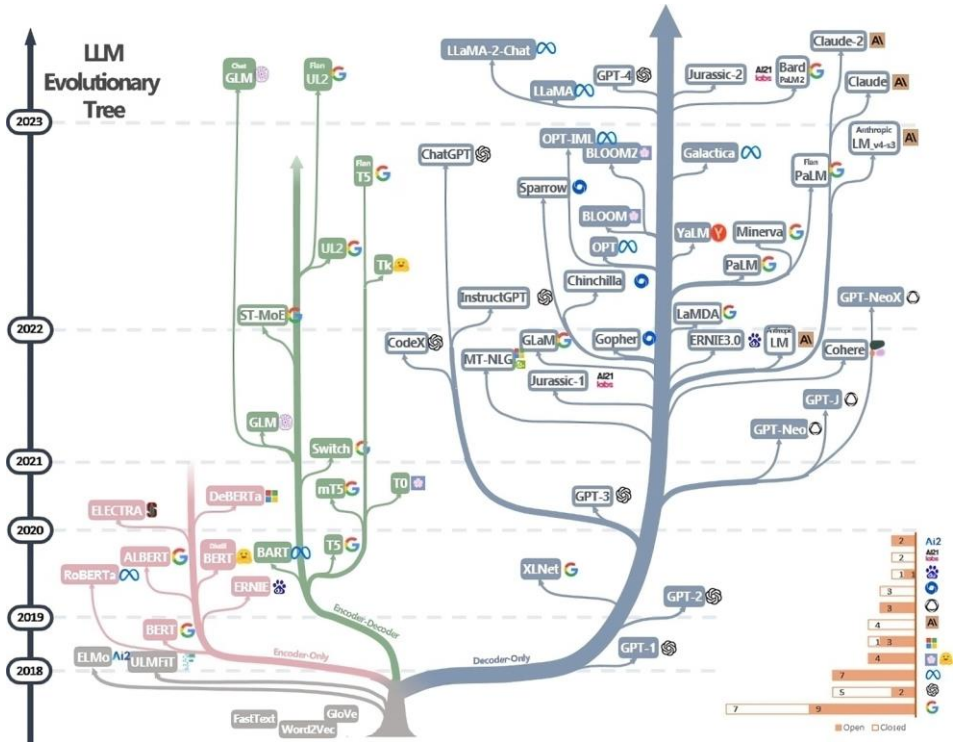
2017	Transformer (Vaswani et al.)
2018	GPT-1, BERT, ELMo
2019	GPT-2 (1.5B), T5, XLNet
2020	GPT-3 (175B), scaling laws (Kaplan)
2022	Chinchilla, ChatGPT, InstructGPT
2023	GPT-4, LLaMA, Gemini, Mixtral
2024-25	LLaMA 3, DeepSeek, reasoning models

Adapted from Yang et al., 'Harnessing the Power of LLMs in Practice' (2023)

How do we go from purpose driven models to LLMs?

Some aspects:

- **Architecture/Tasks**
  - Single task encoder-decoder -> Multi-task decoder-only
- **Data Scaling**
  - Heavy dose of filtering!
  - Variety of sources and types (code, language, vision-language, embodied)
- **Loss Scaling**
  - Encoder-Decoder -> Decoder-Only
- **Architecture**
  - Minor architectural tweaks
  - Better position embedding (RoPE)
  - Efficiency improvements: FlashAttention, MoE (sometimes), Grouped Attention, KV Caching
- **Training**
  - Many stages of training!
  - Predict-next-token -> Helpful Assistant -> Aligned Helpful Assistant

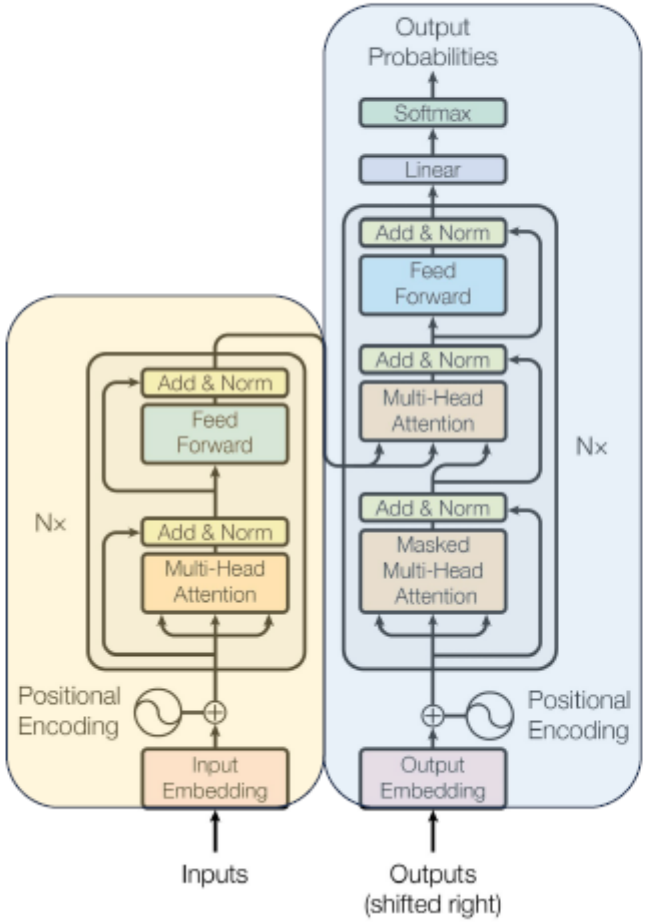


https://github.com/Mooler0410/LLMsPracticalGuide

2018 – Inception of the LLM Era

BERT
Oct 2018

Representation



GPT
June 2018

Generation

BERT - Bidirectional Encoder Representations

BERT Pre-Training Corpus:

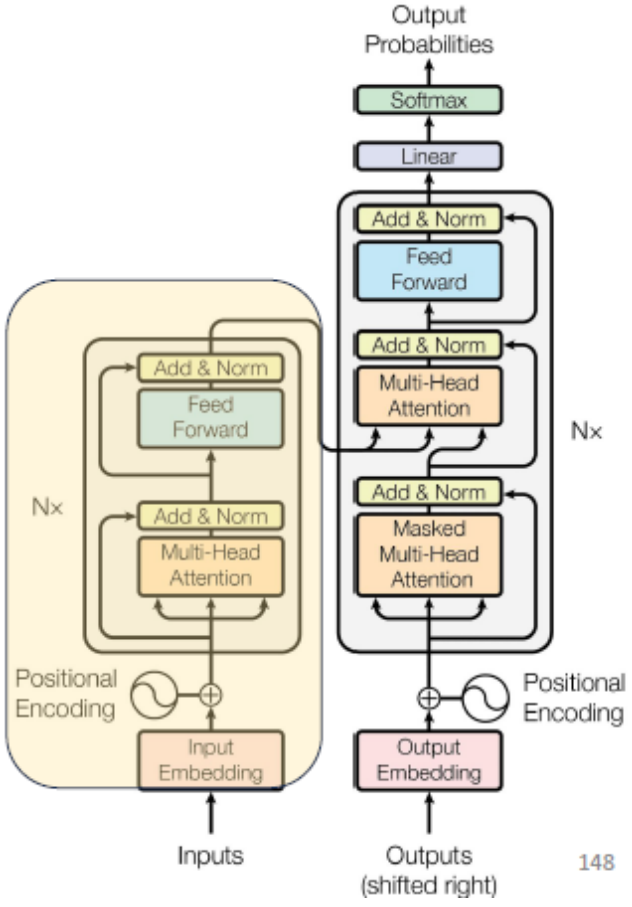
- English Wikipedia - 2,500 million words
- Book Corpus - 800 million words

BERT Pre-Training Tasks:

- MLM (Masked Language Modeling)
- NSP (Next Sentence Prediction)

BERT Pre-Training Results:

- BERT-Base – 110M Params
- BERT-Large – 340M Params

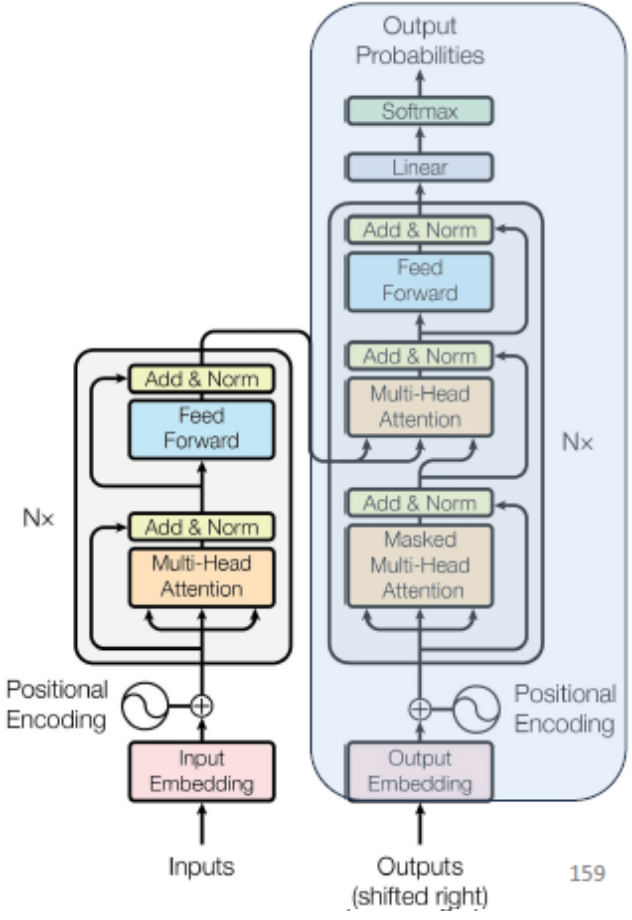


GPT – Generative Pretrained Transformer

GPT Pre-Training Corpus:
- Similarly, BooksCorpus and English Wikipedia

GPT Pre-Training Tasks:
- Predict the next token, given the previous tokens
  - More learning signals than MLM

GPT Pre-Training Results:
- GPT – 117M Params
  - Similarly competitive on GLUE and SQuAD



Why Decoder-Only Won

	Encoder-Only (BERT)	Decoder-Only (GPT)
Attention	Bidirectional	Causal (left-to-right)
Pre-train task	Masked LM (15% tokens)	Next-token prediction (all tokens)
Generation	Needs separate decoder	Built-in (just keep generating)
Scaling	Plateaus earlier	Scales predictably with compute
Unified	Separate fine-tune head	Same objective: pre-train & generate

Almost all modern LLMs (GPT-4, LLaMA, Gemini, Claude) are decoder-only

GPT 2 – Generalizing to Unseen Tasks

- LMs can be used for different tasks by pre-training a “base” model and then fine-tuning for the task(s) of interest

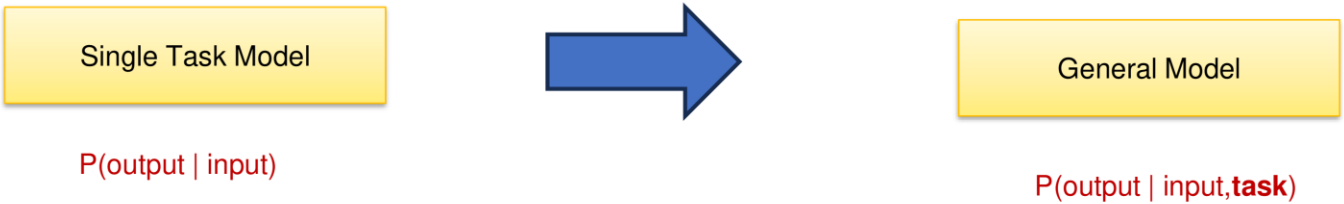
- Practical Issues:
  - Too many copies of the model
  - Need for large-scale labeled data for fine-tuning
  - Can do only specific task

GPT 2 – Generalizing to Unseen Tasks

- LMs can be used for different tasks by pre-training a “base” model and then fine-tuning for the task(s) of interest
- Practical Issues:
  - Too many copies of the model
  - Need for large-scale labeled data for fine-tuning
- Multi-task Training?
  - Data remains a challenge
  - Humans don’t need such large volumes of data to learn – can we do better?
- Train a model that can perform NLP tasks in a zero-shot manner

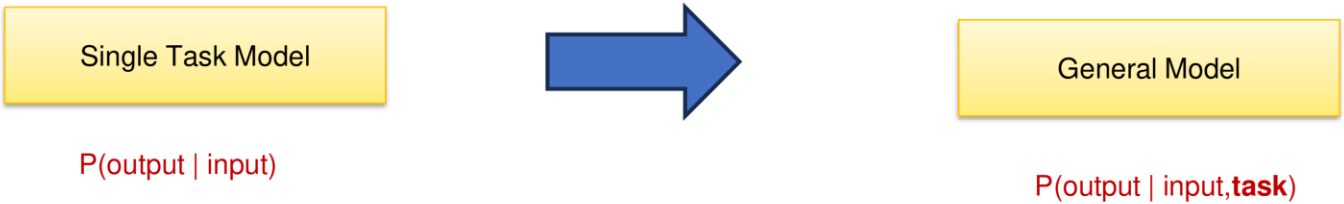
GPT 2 – Task Specifications

- Primary shift comes from modeling assumptions from single-task to general model



GPT 2 – Task Specifications

- Primary shift comes from modeling assumptions from single-task to general model



- Task descriptions may be provided as text – for example, translate this French text to English

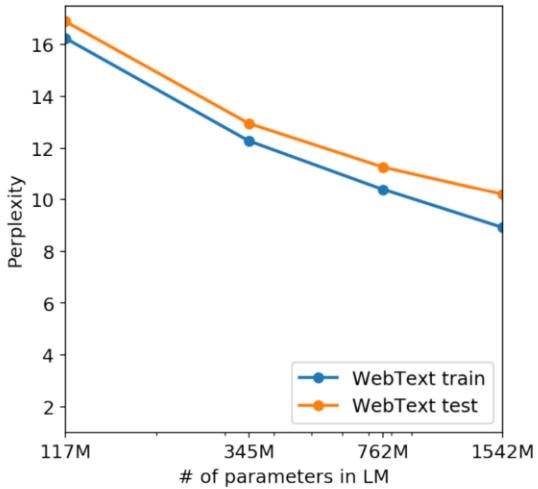
GPT 2 – what makes such an LM work ?

- Diverse training data
  - Model can do many disparate tasks with no training at all!
- Scaling model capacity and data

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	**21.8**
117M	**35.13**	45.99	**87.65**	**83.4**	**29.41**	65.85	1.16	1.17	37.50	75.20
345M	**15.60**	55.48	**92.35**	**87.1**	**22.76**	47.33	1.01	**1.06**	26.37	55.72
762M	**10.87**	**60.12**	**93.45**	**88.0**	**19.93**	**40.31**	**0.97**	**1.02**	22.05	44.575
1542M	**8.63**	**63.24**	**93.30**	**89.05**	**18.34**	**35.76**	**0.93**	**0.98**	**17.48**	42.16

Scaling in GPT-2

- Scaling improves the perplexity of the LM and improves performance

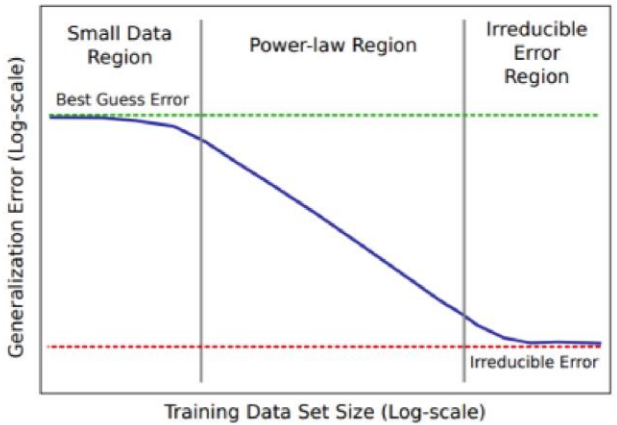


$$\text{Perplexity} = e^{H(P,Q)} = e^{-\frac{1}{N}\sum_{i=1}^{N}\log q(w_i)}$$

Sources: [comet.com]

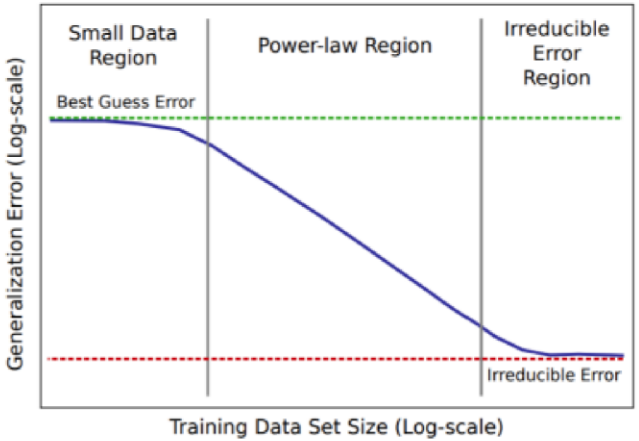
Why is this interesting? Look at data scaling

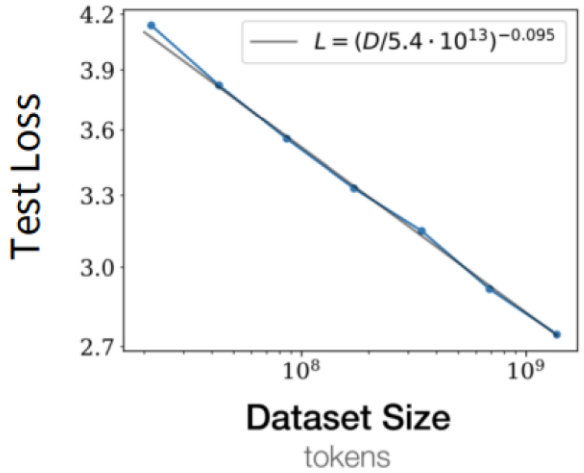
- We know that typical scaling effects look like this when we increase the amount of training data



Why is this interesting? Look at data scaling

- Loss and dataset size is linear on a log-log plot
- This is “power-law scaling”





Can we continue to scale the data?

Data Scaling | Collecting High-Quality Self-Supervision at Scale

We could get a lot more data from CommonCrawl!

Data Scaling | Collecting High-Quality Self-Supervision at Scale

We could get a lot more data from CommonCrawl!
A lot of it is spam though…

Data Scaling | Collecting High-Quality Self-Supervision at Scale

We could get a lot more data from CommonCrawl!
A lot of it is spam though…
How do we get “useful” data?

T5 - Encoder-Decoder with Common Crawl Scale Data

T5 Corpus (AKA C4)
All Common Crawl Text Which Meets Heuristics

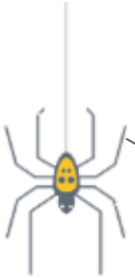
Size
~350 Billion Tokens

Quality
Varying quality text, Broad Knowledge, Improved Diversity

- We only retained lines that ended in a terminal punctuation mark (i.e. a period, exclamation mark, question mark, or end quotation mark).
- We discarded any page with fewer than 3 sentences and only retained lines that contained at least 5 words.
- We removed any page that contained any word on the "List of Dirty, Naughty, Obscene or Otherwise Bad Words".[6]
- Many of the scraped pages contained warnings stating that Javascript should be enabled so we removed any line with the word Javascript.
- Some pages had placeholder "lorem ipsum" text; we removed any page where the phrase "lorem ipsum" appeared.
- Some pages inadvertently contained code. Since the curly bracket "{" appears in many programming languages (such as Javascript, widely used on the web) but not in natural text, we removed any pages that contained a curly bracket.
- Since some of the scraped pages were sourced from Wikipedia and had citation markers (e.g. [1], [citation needed], etc.), we removed any such markers.
- Many pages had boilerplate policy notices, so we removed any lines containing the strings "terms of use", "privacy policy", "cookie policy", "uses cookies", "use of cookies", or "use cookies".
- To deduplicate the data set, we discarded all but one of any three-sentence span occurring more than once in the data set.

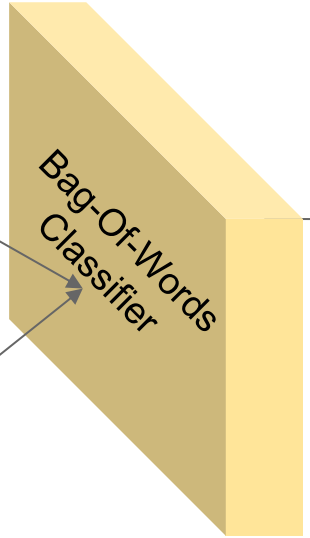
Raffel et al. 2019

GPT-3 - Increased Scaling Via Automated Data Curation



Low-Quality, High Volume

Training



Distinguish High and Low Quality

URL Domain	# Docs	% of Total Docs
bbc.co.uk	116K	1.50%
theguardian.com	115K	1.50%
washingtonpost.com	89K	1.20%
nytimes.com	88K	1.10%
reuters.com	79K	1.10%
huffingtonpost.com	72K	0.96%
cnn.com	70K	0.93%
cbc.ca	67K	0.89%
dailymail.co.uk	58K	0.77%
go.com	48K	0.63%

High Quality, Medium Volume

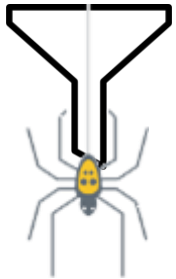
Brown et al. 2020

Data | GPT-2 to Original GPT-3 was mostly data scaling

GPT-3 Corpus
Common-Crawl Filtered using
GPT-2 Training Data

Size
~400 Billion Tokens

Quality
High-ish quality text,
Broad Knowledge,
Web-scale Diversity



Brown et al. 2020

Data | Recent Open Source models focus heavily on data scaling

Llama 1 Corpus

Size
~1.4 Trillion Tokens

Quality
Varying quality text,
Broad Knowledge,
Web-scale Diversity,
Includes Code!

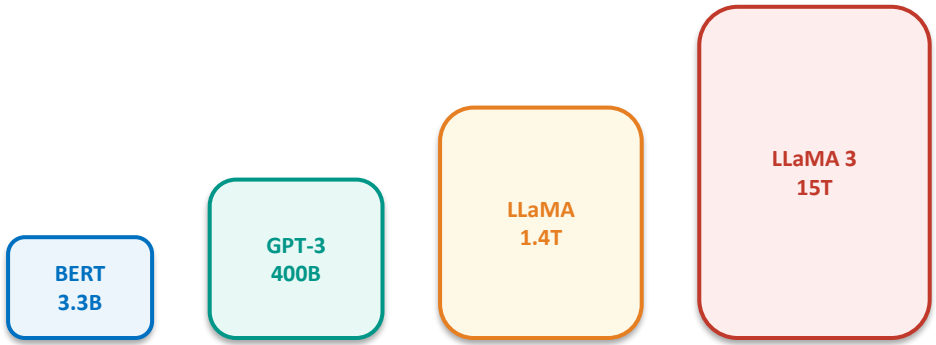
Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Touvron et al. 2023

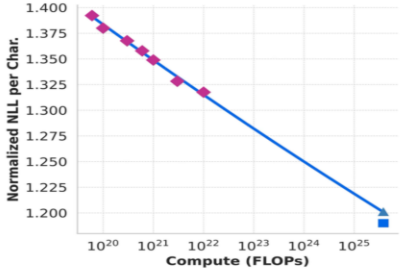
Data | The Data Arms Race

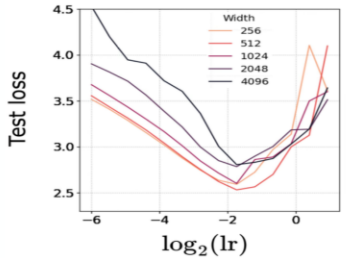
- LLaMA 3 (2024): 15T tokens — 10× LLaMA 1
- Gemini / GPT-4 / Claude: data sizes undisclosed
- Data mixture has become the biggest competitive secret
- Running out of public internet text? (~100T tokens estimated total)
  - Some estimates suggest high-quality English text is ~3-5T tokens
  - Total multilingual web text may be ~50-100T tokens
- New frontiers: synthetic data, multi-modal data, private datasets
  - Phi models showed small, curated synthetic data can rival large-scale web data

But will scale keep helping? Can we characterize the improvement? (before spending $10Ms+)



- More compute in pre-training, through larger models, more training data, and longer training, can improve performance
  - if optimal hyperparameters are chosen
- On the other hand, suboptimal combinations of choices can lead to performance degradation with scale
- We need guidance for hyper parameter choices when scaling up the model training
  - for optimal performance , and
  - for predicting that performance
- $\Longrightarrow$ Scaling Law





Scaling Effects

- The effect of some hyperparameters on big LMs can be predicted before training – optimizer (Adam v/s SGD), model depth, LSTM v/s Transformer

- Idea:
  - Train a few smaller models
  - Establish a scaling law (e.g. ADAM vs SGD scaling law)
  - Select optimal hyper param based on the scaling law prediction

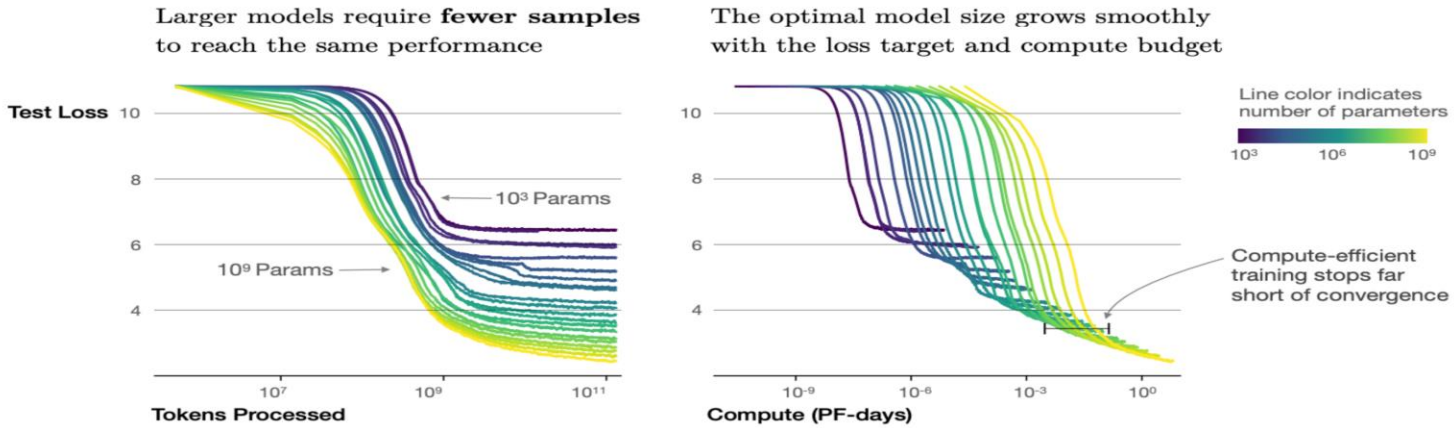
Scaling - (Kaplan,2020)

- Open AI Study : **Scaling Laws for Neural Language Models** (Kaplan et al. 2020)

- Key Findings:
  - Performance depends strongly on scale, and weakly on the model shape
    - Larger models are more sample-efficient
    - Smooth power laws ($y = ax^k$) b/w empirical performance & N - parameters, D - dataset size, C - compute

- **Definition**. **Neural Scaling Laws** describe how neural network performance changes as key factors are scaled up or down.
- We consider the following four factors:
  - Size of the model N: number of parameters
  - Size of the training dataset D: number of samples or tokens
  - Compute C: measured in FLOPs (FLoating-point OPerations)
  - Generalization performance L: test loss after training

- For optimal performance, all three factors (dataset, compute, model size) need to scale together
- Larger models need fewer samples to achieve the same loss
- Large models are compute-optimal when undertrained
- Train on **larger model with fewer samples** (than training smaller model to convergence)



- Chinchila [Hoffman et al. 2022] scaling law suggests that model size N and data size D should scale together, at the same rate

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.

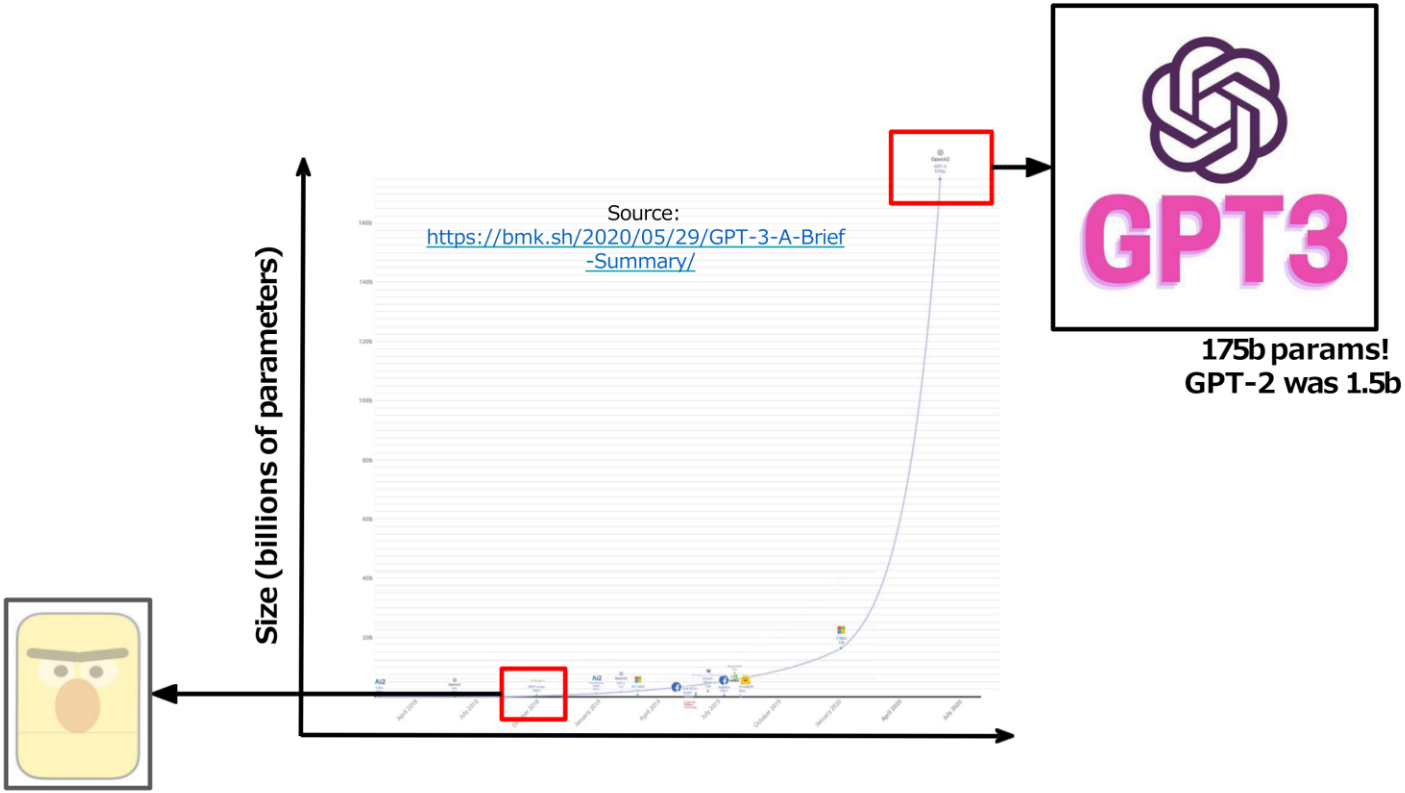
Approach	Coeff. a where $N_{opt} \propto C^a$	Coeff. b where $D_{opt} \propto C^b$
1. Minimum over training curves	0.50 (0.488, 0.502)	0.50 (0.501, 0.512)
2. IsoFLOP profiles	0.49 (0.462, 0.534)	0.51 (0.483, 0.529)
3. Parametric modelling of the loss	0.46 (0.454, 0.455)	0.54 (0.542, 0.543)
Kaplan et al. (2020)	0.73	0.27

- it was empirically discovered that to scale up compute, N and D need to increase together: 20 training tokens is optimal per parameter

Optimal number of tokens for various sized models

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

Model Scaling: GPT-3



Impact of Chinchilla

Before Chinchilla

- GPT-3 175B: 300B tokens (1.7 tok/param)
- Gopher 280B: 300B tokens (1.1 tok/param)
- Focus: make model bigger
- Data treated as 'enough'
- Massive inference costs

After Chinchilla

- Chinchilla 70B: 1.4T tokens (20 tok/param)
- LLaMA 65B: 1.4T tokens (21.5 tok/param)
- Focus: balance model + data
- Data quality & quantity critical
- Smaller, better models

Chinchilla shifted the field from 'bigger models' to 'better-trained models'

Scaling | Beyond Chinchilla: Inference-Optimal Scaling

- Chinchilla is compute-optimal for training
- But what about inference? Smaller models are cheaper to serve
- Inference-optimal: overtrain a smaller model on more data
  - A model served millions of times — inference cost dominates
- LLaMA 3 8B: trained on 15T tokens (~1,875 tok/param!)
- 93× the Chinchilla-optimal data for that model size
- Trade more training compute for cheaper deployment
  - One-time training cost vs. ongoing inference cost at scale

Sardana & Frankle, 'Beyond Chinchilla-Optimal: Accounting for Inference' (2023)

Emergent Abilities with GPT-3 – Wei et. al 2022

- Emergent abilities:
  - not present in smaller models but is present in larger models
  - Do LLMs like GPT3 have these ?

- Findings:
  - GPT-3 trained on text can do arithmetic problems like addition and subtraction
  - Different abilities “emerge” at different scales

Emergent Abilities with GPT-3 – Wei et. al 2022

- Emergent abilities:
  - not present in smaller models but is present in larger models
  - Do LLMs like GPT3 have these ?

- Findings:
  - GPT-3 trained on text can do arithmetic problems like addition and subtraction
  - Different abilities “emerge” at different scales
  - Model scale is not the only contributor to emergence – for 14 BIG-Bench tasks, LaMDA 137B and GPT-3 175B models perform at near-random, but PaLM 62B achieves above-random performance
  - Problems LLMs can’t solve today may be emergent for future LLMs

Modern LLM Architecture

What Changed Since 2017?

Transformer vs Modern LLM

	Original Transformer	Modern LLM (LLaMA)
Architecture	Encoder-Decoder	Decoder-only
Normalization	Post-LayerNorm	Pre-RMSNorm
Position Enc.	Sinusoidal (absolute)	RoPE (relative/rotary)
Activation	ReLU	SwiGLU
Attention	MHA	GQA
Context	512 tokens	128K+ tokens

Every component has been replaced with minor mods, same skeleton

Normalization | Pre-RMSNorm

- Post-LayerNorm (original): output = LN(x + Sublayer(x))
- Pre-LayerNorm (modern): output = x + Sublayer(Norm(x))
- RMSNorm simplifies LayerNorm:

$$\textbf{RMSNorm}(x) = \gamma \cdot x \,/\, \text{RMS}(x)$$

$$\text{RMS}(x) = \sqrt{(1/n) \sum x_i^2}$$

- No mean subtraction, no bias term → simpler
- ~10-15% faster than LayerNorm
- Equally effective in practice (Zhang & Sennrich, 2019)

Zhang & Sennrich, 'Root Mean Square Layer Normalization' (2019)

Position Encoding | Rotary Position Embeddings (RoPE)

- Original Transformer: sinusoidal positional embeddings (added once)
- RoPE: encode position by rotating query and key vectors in 2D subspaces
- Each pair of dimensions forms a 2D plane, rotated by $m \cdot \theta$

$\mathbf{q}_m \cdot \mathbf{k}_n$ depends only on $(m - n)$ ← **relative position!**

- No learned parameters for position
- Naturally decays with distance
- Better length generalization than absolute
- Compatible with linear attention approximations

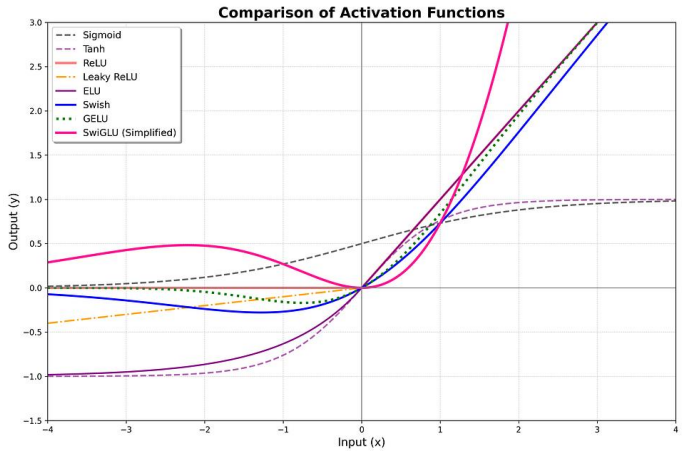
Su et al., 'RoFormer: Enhanced Transformer with Rotary Position Embedding' (2021)

Activation | SwiGLU Activation

- Original Transformer FFN: $FFN(x) = ReLU(W_1 \cdot x) \cdot W_2$
- SwiGLU adds a gating mechanism with a third weight matrix:

$$\mathbf{SwiGLU(x) = Swish(W_1 \cdot x) \odot W_3 \cdot x}$$

$Swish(x) = x \cdot \sigma(x)$ *where σ = sigmoid*

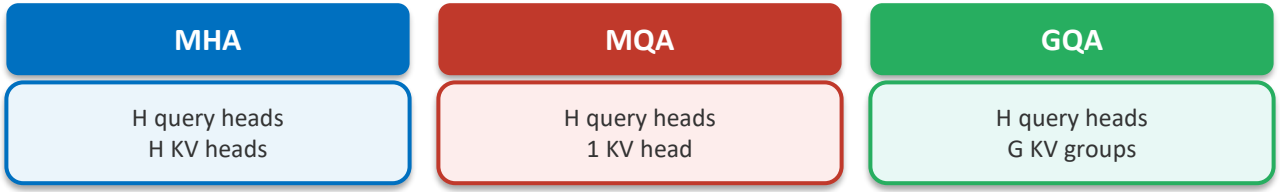


- $\odot$ = element-wise multiplication (gating)
- Adds ~50% more parameters to FFN layer
- But consistently outperforms ReLU and GELU
- Used in LLaMA, PaLM, and most modern LLMs
- Smooth, non-monotonic → richer gradients

Shazeer, 'GLU Variants Improve Transformer' (2020)

Sources: [ai.plainenglish.io]

Grouped-Query Attention | MHA vs MQA vs GQA



- MHA: each head has its own Q, K, V projections (full expressivity)
- MQA: all heads share one K, V → fast but quality degrades
- GQA: groups of heads share K, V → sweet spot
- Smaller KV-cache = faster inference, especially long contexts
- LLaMA 2/3: 8 KV groups (e.g., 32 Q heads, 8 KV heads)

Ainslie et al., 'GQA: Training Generalized Multi-Query Transformer Models' (2023)

LLaMA Architecture | Putting It All Together

	8B	70B	405B
Layers	32	80	126
Dim	4096	8192	16384
Heads	32	64	128
GQA Groups	8	8	16

- ✓ Pre-RMSNorm (before attn & FFN)
- ✓ RoPE (rotary position embeddings)
- ✓ SwiGLU activation in FFN
- ✓ Grouped-Query Attention
- ✓ Byte-Pair Encoding tokenizer (128K vocab)
- ✓ No bias terms anywhere

Touvron et al., 'LLaMA: Open and Efficient Foundation Language Models' (2023)

Distributed Training | Data Parallel → FSDP

DDP (Data Parallel)

- Replicate entire model on each GPU
- Split data (mini-batch) across GPUs
- Each GPU: forward + backward
- All-reduce to sync gradients
- Problem: model must fit in 1 GPU!

FSDP (Fully Sharded)

- Shard model params across GPUs
- All-gather params before forward
- Reduce-scatter grads after backward
- Each GPU holds only 1/N of model
- Total memory = model / N + activations

FSDP: train models much larger than a single GPU's memory

Mixture of Experts | Sparse Scaling

- Replace FFN with: Router + N expert FFNs (only k activated per token)

Advantages

- More parameters, same compute (sparse)
- Cheaper training per quality level
- Lower inference FLOPs per token
- Experts can specialize

Challenges

- Unstable training (routing collapse)
- Expert collapse (few experts used)
- Load balancing loss required
- Higher memory (all experts loaded)

- Load-balancing auxiliary loss encourages uniform expert usage
- Used in: Mixtral 8x7B, GPT-4 (rumored), DeepSeek-V2, Grok

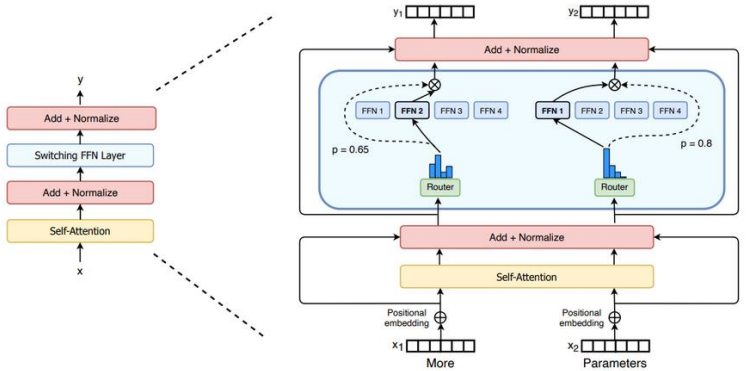


Figure Source: huggingface.co

Fedus et al., 'Switch Transformers' (2022); Jiang et al., 'Mixtral of Experts' (2024)

Three Stages | Pre-training → SFT → RLHF/DPO

- Each stage uses dramatically less data but has outsized impact
- Pre-training: learn language, knowledge, reasoning
- SFT: learn to follow instructions and be an assistant
- RLHF/DPO: learn human preferences and safety constraints

"Alignment tax" is low: small data, big behavioral change

Training of Decoder-only LLMs – Llama 2

1. Auto-regressive Pre-training - Train to predict the next token on very large-scale corpora (~3 trillion tokens)

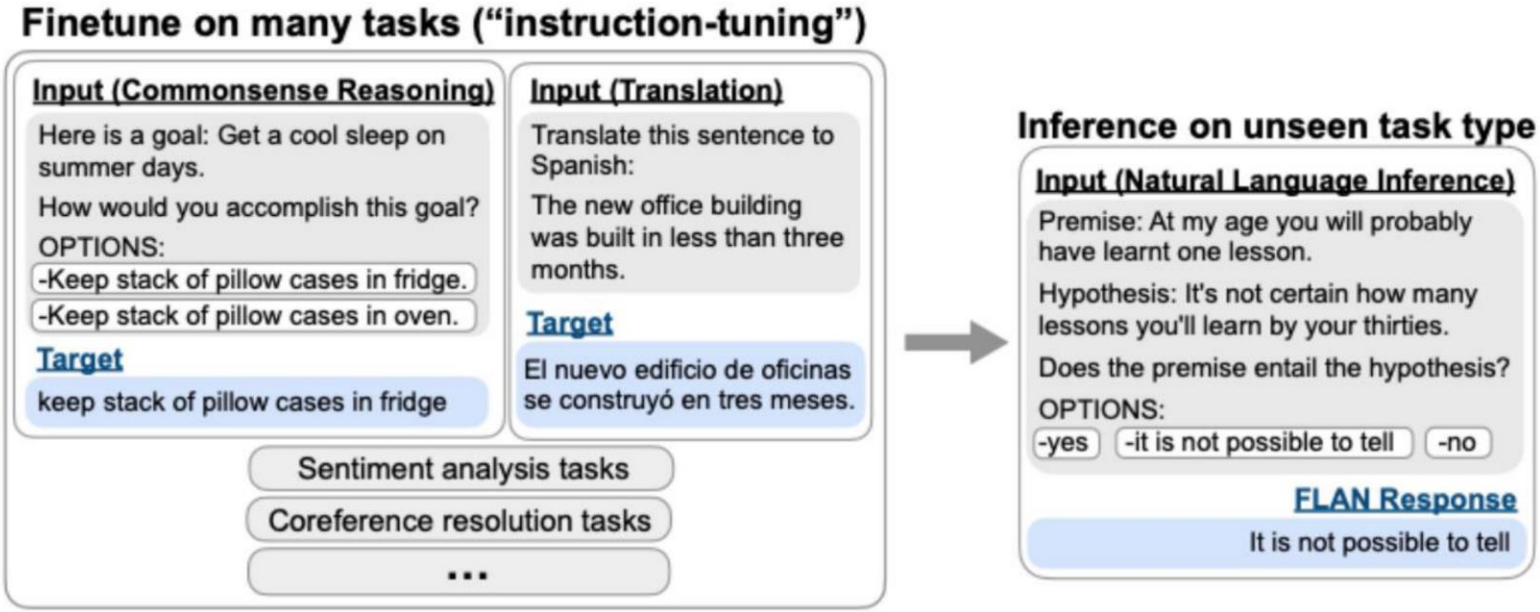
Training of Decoder-only LLMs – Llama 2

1. Auto-regressive Pre-training - Train to predict the next token on very large scale corpora (~3 trillion tokens)
2. Instruction Fine-tuning/ Supervised Fine-tuning (SFT) - Fine-tune the pre-trained model with pairs of (instruction+input,output) with large dataset and then with small high-quality dataset

Instruction fine-tuning provides as a prefix a natural language description of the task along with the input.

- E.g. Translate into French this sentence: my name is -> je m’appelle

Instruction Tuning (Wei et. al. 2021)



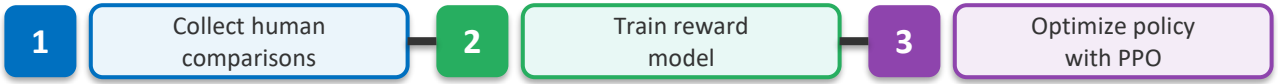
Unsafe Outputs – Alignment Problem

- LLMs may produce
  - Harmful text – unparliamentary language, bias and discrimination
  - Text that can cause direct harm – allowing easy access to dangerous information
- Therefore, LLMs should be trained to produce outputs that align with human preferences and values
- Modern LLMs do so by using SFT and by using human preference directly in model training

Training of Decoder-only LLMs – Llama 2

1. Auto-regressive Pre-training - Train to predict the next token on very large scale corpora (~3 trillion tokens)
2. Instruction Fine-tuning/ Supervised Fine-tuning (SFT) - Fine-tune the pre-trained model with pairs of (instruction+input,output) with large dataset and then with small high-quality dataset
3. Safety / RLHF - Design a reward model based on human feedback and use policy gradient methods with the trained reward model to update LLM parameters so that outputs align with human values

RLHF | Reinforcement Learning from Human Feedback



- Step 1: Given prompt, model generates two responses → human picks better one
- Step 2: Train a reward model (RM) to predict human preferences
- Step 3: Use RM as reward signal to fine-tune LLM with PPO
- Result: model learns to generate responses humans prefer
- Used by: InstructGPT, ChatGPT, Claude, Gemini

Ouyang et al., 'Training language models to follow instructions with human feedback' (2022)

Reward Model | Learning Human Preferences

- Trained on preference pairs: (prompt, y_win, y_lose)
- Bradley-Terry model of preferences:

$$\mathcal{L}_{RM} = -\log \sigma(r(x, y_w) - r(x, y_l))$$

- r(x, y) = scalar reward score for response y given prompt x
- σ = sigmoid → converts score difference to probability
- Loss pushes r(winner) > r(loser)
- Typically initialized from SFT model (remove LM head, add scalar head)
- Larger reward models are more robust to overoptimization

RLHF

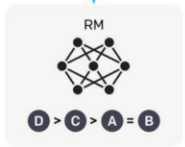
Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

Optimize a policy against the reward model using reinforcement learning.

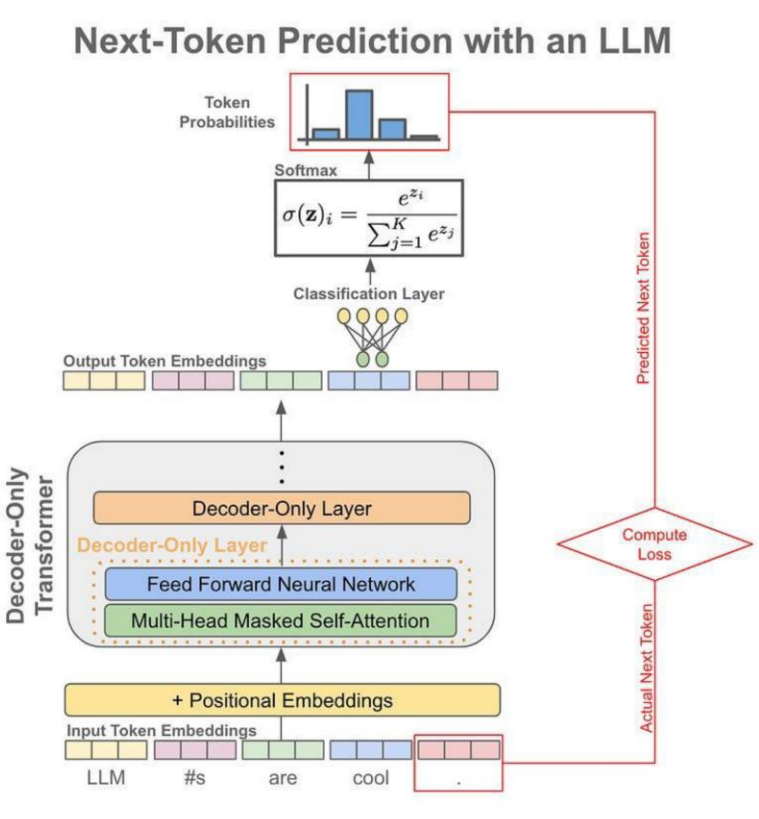
A new prompt is sampled from the dataset.

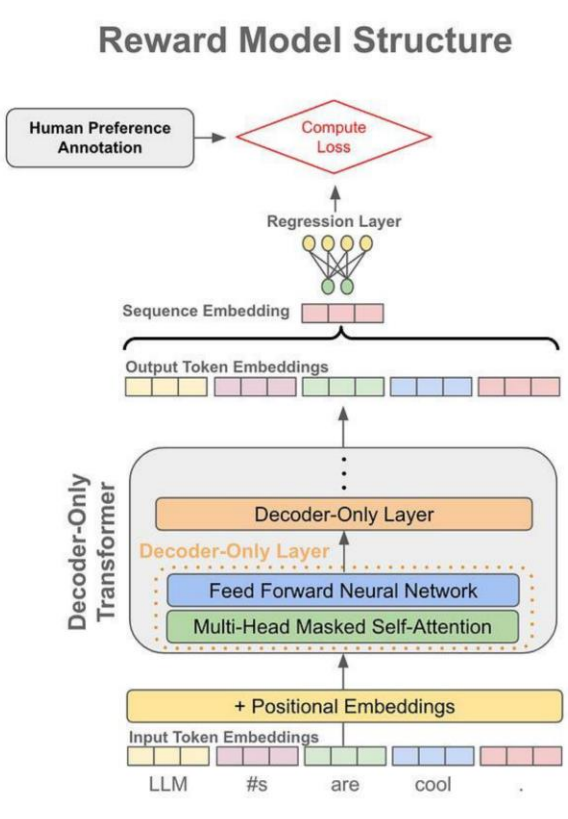
The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

Model Fine-tuning for RLHF





Prompting, In-Context Learning & Efficiency

Getting More from Frozen Models

Zero-Shot & Few-Shot | In-Context Learning

- GPT-3 key insight: models can learn from examples placed in the prompt
- No gradient updates — the model "learns" just by reading examples

Zero-Shot

Translate to French:
"The cat sat on the mat."

→ "Le chat s'est assis
sur le tapis."

Few-Shot (2 examples)

"Hello" → "Bonjour"
"Thank you" → "Merci"

Translate: "The cat sat
on the mat." →

- Few-shot consistently outperforms zero-shot on most tasks
- Performance scales with model size (emergent ability)
- No task-specific fine-tuning needed → truly general-purpose
- Limited by context window length

Brown et al., 'Language Models are Few-Shot Learners' (GPT-3, 2020)

In-Context Learning | Why Does It Work?

- Transformers can implement learning algorithms in their forward pass!

- Theoretical results:
  – Transformers can simulate gradient descent (Akyurek et al. 2023)
  – Linear attention ≈ one step of GD on in-context examples
  – Deeper layers = more optimization steps

- Practical intuition:
  – Pre-training on diverse tasks creates "meta-learning" ability
  – In-context examples specify the task, not teach new knowledge
  – The model retrieves relevant pre-trained capabilities

- Still an active area of research!

Akyurek et al., 'What Learning Algorithm is In-Context Learning?', ICML 2023

Chain-of-Thought | "Let's Think Step by Step"

- Intermediate reasoning steps dramatically improve math/logic performance

Standard Prompting

Q: Roger has 5 tennis balls. He buys 2 cans of 3. How many now?

A: 11 ✗

Chain-of-Thought Prompting

Q: Same question...
A: Roger started with 5 balls.
2 cans of 3 = 6 balls.
5 + 6 = 11 ✓

- "Let's think step by step" (zero-shot CoT) works surprisingly well
- Only effective in large models (>~60B parameters)
- Small models generate plausible-sounding but incorrect reasoning
- Now standard for math, code, and complex reasoning tasks
- Extensions: Self-Consistency, Tree-of-Thought, ReAct

Wei et al., 'Chain-of-Thought Prompting Elicits Reasoning in LLMs' (2022)

Reasoning Models | Test-Time Compute Scaling

- New paradigm: Scale compute at inference time, not just training

- OpenAI o1 / o3 (2024):
  - Model generates long internal chain-of-thought before answering
  - Dramatically better at math, coding, science, competition problems

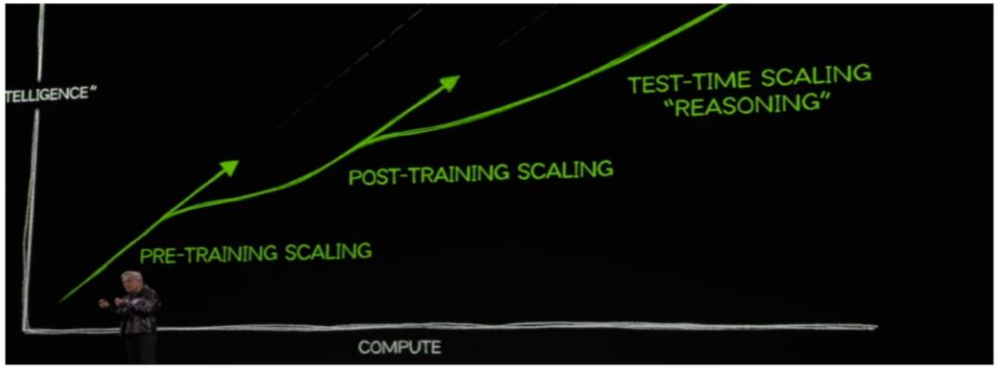
- DeepSeek-R1 (2025): Open-source!
  - Trained with RL (GRPO) to develop reasoning without human CoT data
  - Emergent: self-verification, problem decomposition, backtracking

- Analogy: System 1 (fast/intuitive) vs System 2 (slow/deliberate) thinking

- Trade-off: Much slower & more expensive

DeepSeek-AI, 'DeepSeek-R1: Incentivizing Reasoning in LLMs via RL', 2025

- Beyond pretraining scaling laws, there is increasing interest in post-training scaling and test-time scaling. Techniques to improve scaling include
  - Post training: fine-tuning, quantization, pruning, sdistillation, etc.
  - Test time: best of N sampling, MCMC sampling, thinking time, etc.



Emergent Abilities | Do They Exist?

- Wei et al. 2022: Some abilities appear suddenly at scale
  – Absent in small models, present in large models
  – Examples: arithmetic, word unscrambling, CoT reasoning

- Schaeffer et al. 2023: "Emergent abilities are a mirage"
  – Argued it's an artifact of nonlinear evaluation metrics
  – With linear metrics (token-level accuracy), scaling is smooth

- Current consensus:
  – Gradual improvement, but nonlinear metrics create appearance of jumps
  – Bigger models reliably do better; don't expect magic thresholds

Wei et al. 2022; Schaeffer et al., 'Are Emergent Abilities a Mirage?', NeurIPS 2023

How do we go from purpose driven models to LLMs?

Some aspects:

- **Architecture/Tasks**
  - Single task encoder-decoder -> Multi-task decoder-only
- **Data Scaling**
  - Heavy dose of filtering!
  - Variety of sources and types (code, language, vision-language, embodied)
- **Loss Scaling**
  - Encoder-Decoder -> Decoder-Only
- **Architecture**
  - Minor architectural tweaks
  - Better position embedding (RoPE)
  - Efficiency improvements: FlashAttention, MoE (sometimes), Grouped Attention, KV Caching
- **Training**
  - Many stages of training!
  - Predict-next-token -> Helpful Assistant -> Aligned Helpful Assistant