Universität Konstanz

09 | Large Language Models

Giordano De Marzo

https://giordano-demarzo.github.io/

Deep Learning for the Social Sciences

Universität Konstanz

The Attention Equation

Everything we've learned can be summarized in one equation.

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

This is the mathematical heart of the attention mechanism.

- **QK'**: Match queries with keys to get raw attention scores
- Softmax(...): Convert scores to probabilities that sum to 1
- ... × V: Use probabilities to combine value vectors

What each part does:

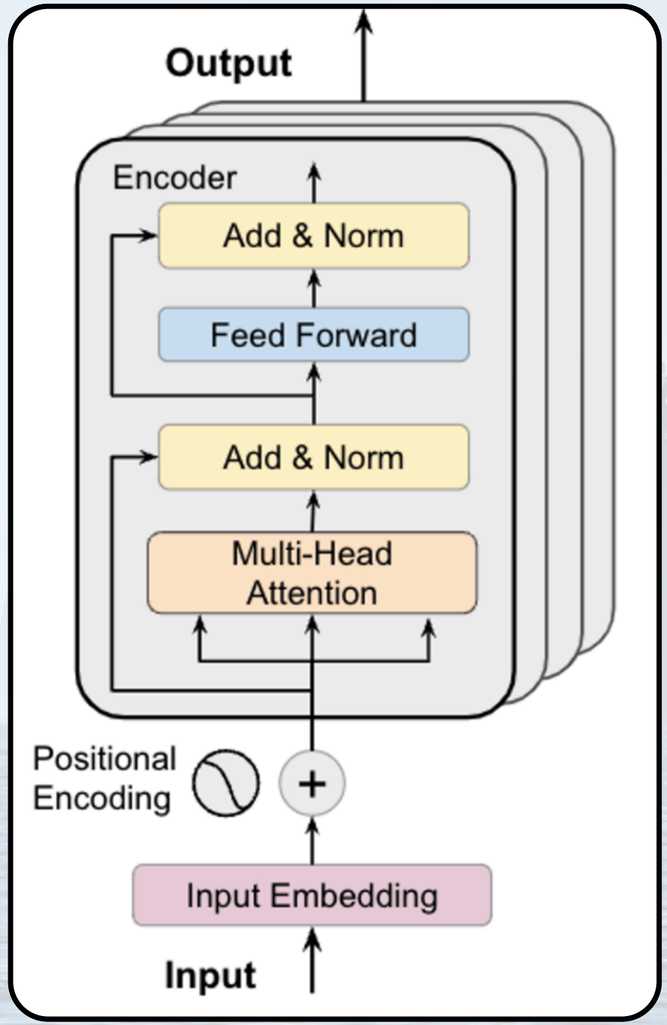
- Q (Query): What each word is looking for
- K (Key): What each word advertises
- V (Value): What each word actually provides

The Transformer

The Transformer is the architecture that revolutionized AI. It is composed of

- **Input embeddings:** Convert words to vectors
- **Positional encoding:** Add position information to embeddings
- **Multi-head attention:** The mechanism we just learned, with multiple heads
- **Feed-forward network (MLP):** Add external knowledge and processing power

The last two blocks are repeated multiple times allowing the model to develop a deep understanding of texts and relations between words



Universität Konstanz

Outline

1. Large Language Models
2. Decoder-Only Models
3. Encoder-Only Models
4. Scaling LLMs

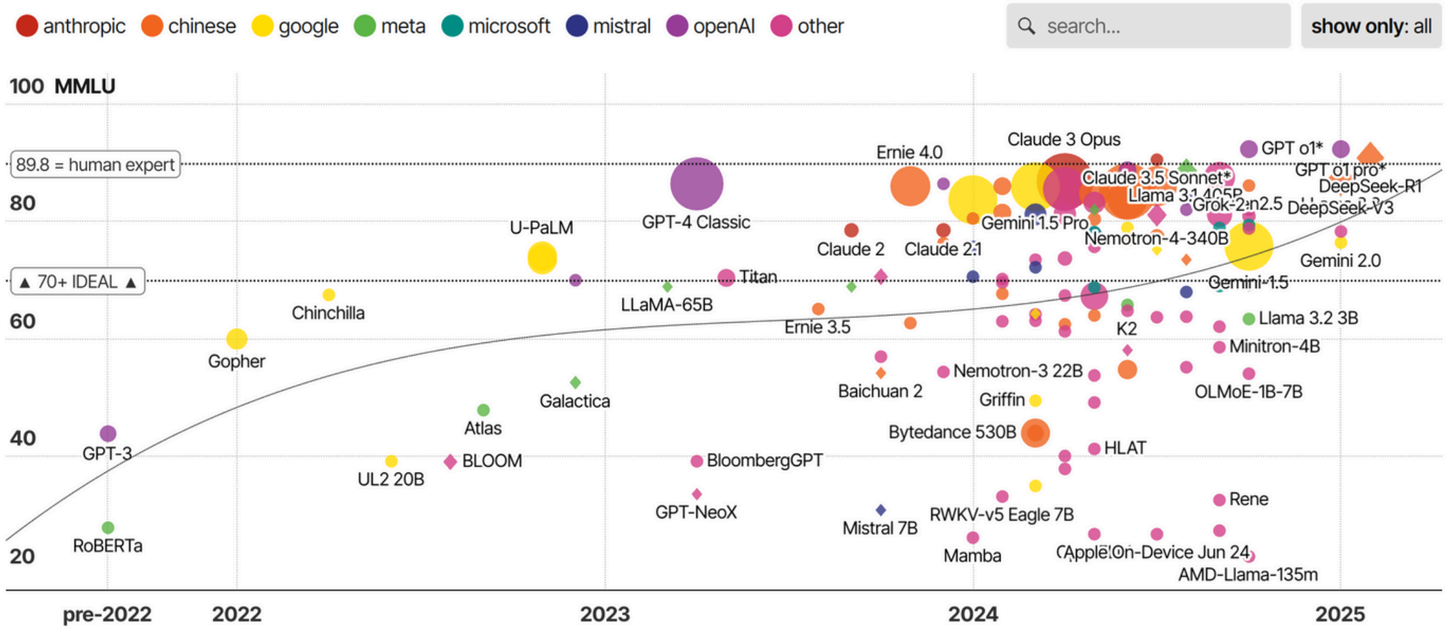
Universität
Konstanz

Large Language Models

What Are Large Language Models?

LLMs are transformers with specific characteristics that enable unprecedented language capabilities.

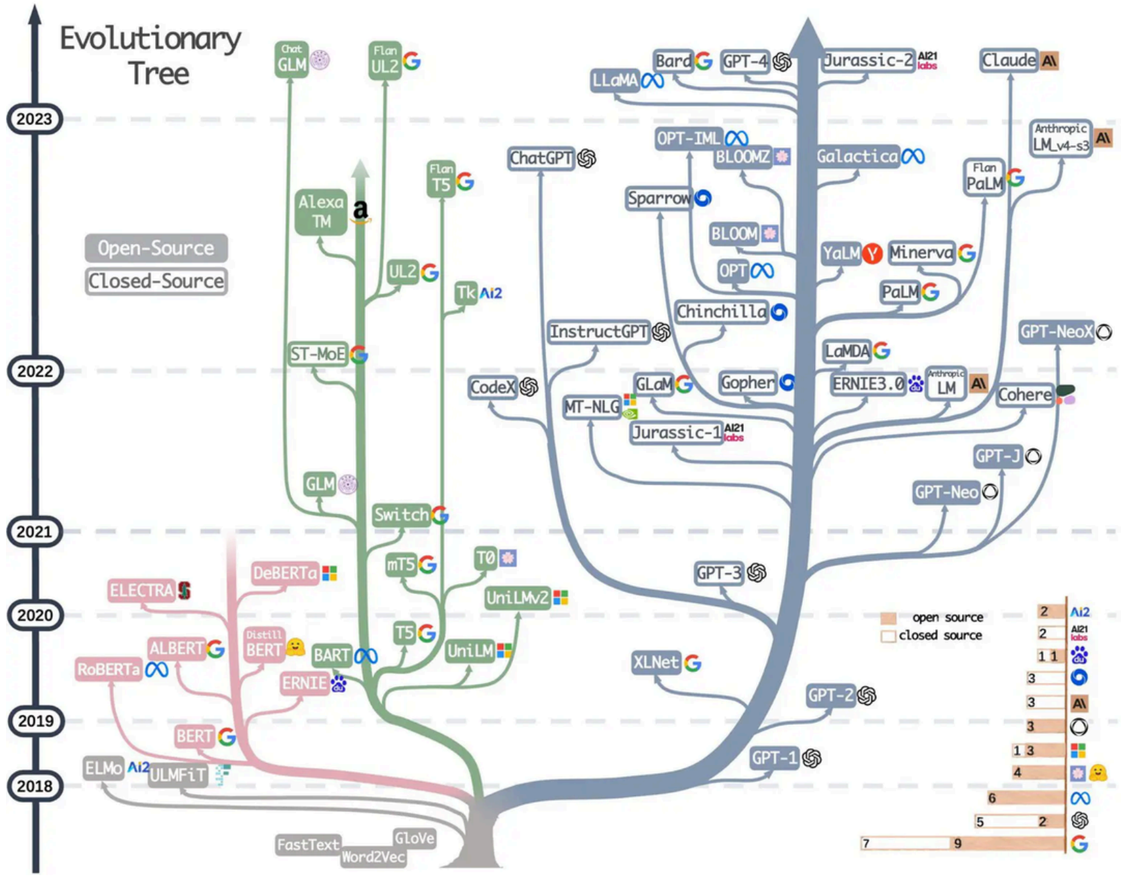
- "Large" = Scale: Millions to trillions of parameters, massive text datasets, significant compute
- "Language" = Text focus: Trained on language modeling
- "Model: = Machine learning system: Mathematical function that learns patterns from data



The Three Families of LLMs

LLMs can be optimized for different tasks through their attention patterns.

- **Encoder-Only** (like BERT):
  - Reads entire text simultaneously with bidirectional attention
- **Decoder-Only** (like GPT):
  - Reads text left-to-right, predicts next word with causal attention
- **Encoder-Decoder** (like T5):
  - Understands input completely, then generates output with cross-attention



Attention Patterns - The Key Difference

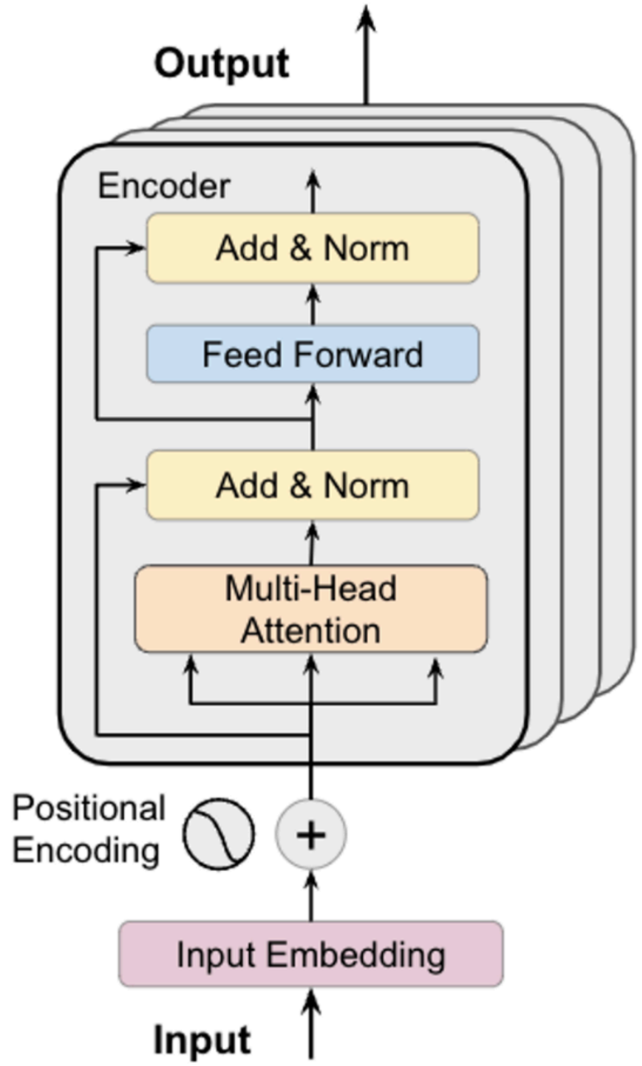
The same transformer architecture can have completely different capabilities depending on how attention is applied.

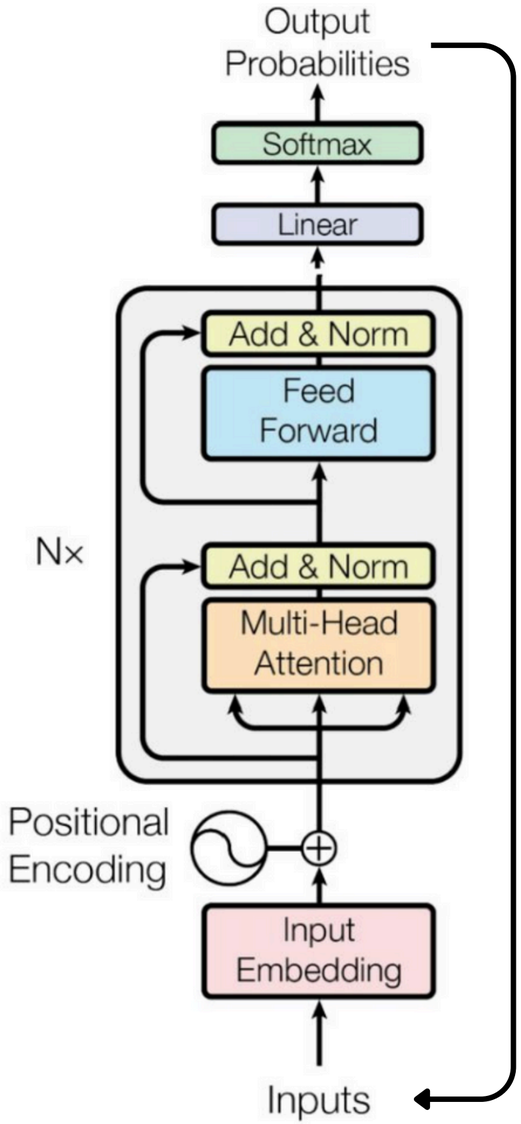
- **Bidirectional Attention** (Encoder-Only):
  - Every word can "see" every other word simultaneously
  - Like reading the entire book before answering questions
- **Causal Attention** (Decoder-Only):
  - Words can only "see" previous words, never future ones
  - Like writing a story one word at a time, naturally
- **Cross Attention** (Encoder-Decoder):
  - Decoder words "look at" all encoder words for context
  - Like having a reference text while writing

Encoder-Only LLMs

These models excel at understanding because they can see the complete context simultaneously. They're the “reading comprehension masters” of the AI world.

- **Bidirectional self-attention:** Each word attends to ALL words in the sequence
- **No masking**: Complete visibility across the entire sequence
- **Multiple layers**: Building increasingly sophisticated understanding (12-24 layers)
- **Perfect for understanding:** Can resolve ambiguity using full context





Decoder-Only LLMs

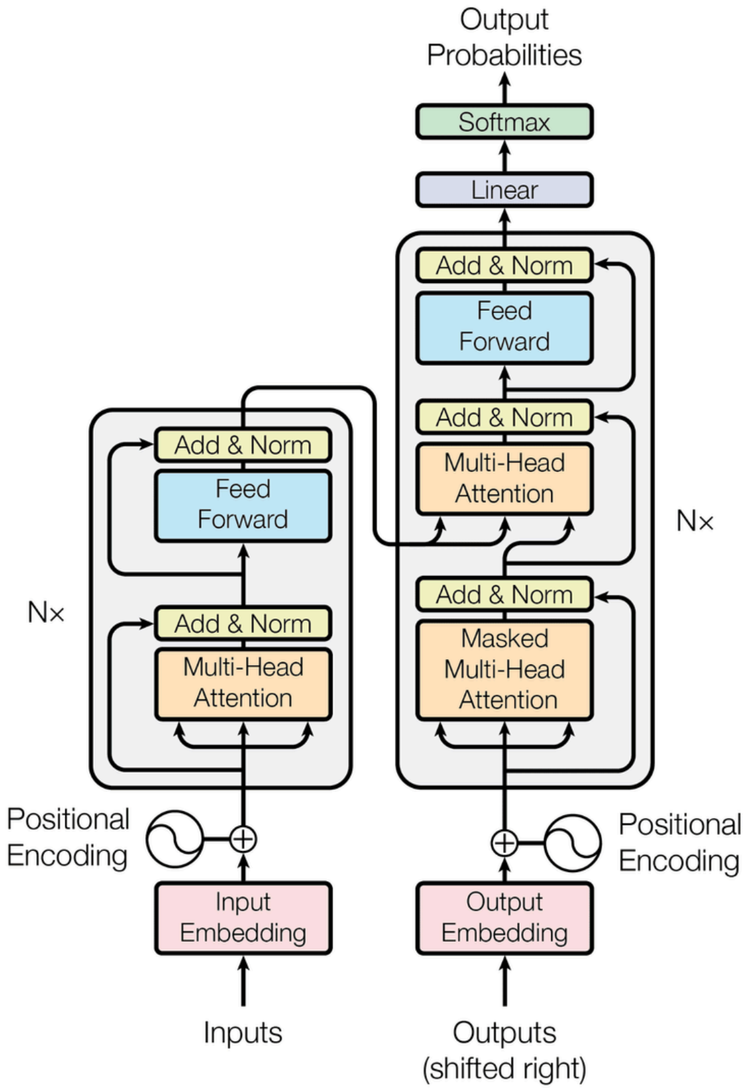
These models generate text naturally by predicting one word at a time, just like humans write. This constraint actually makes them incredibly powerful and scalable.

- **Causal self-attention:** Each word only sees previous words in the sequence
- **Lower triangular masking:** Future tokens are systematically hidden
- **Autoregressive generation:** Each prediction becomes input for the next step
- **Natural training:** Can learn from any text without special formatting

Encoder-Decoder LLMs

In Encoder-Decoder models cross attention is the bridge between comprehension and generation.

- The core challenge: Need both complete input understanding AND natural generation
- Three attention types working together:
  - Encoder self-attention: Understanding input (bidirectional)
  - Decoder self-attention: Generating coherently (causal)
  - Decoder cross-attention: Using input information (encoder → decoder)



Choosing the Right Architecture

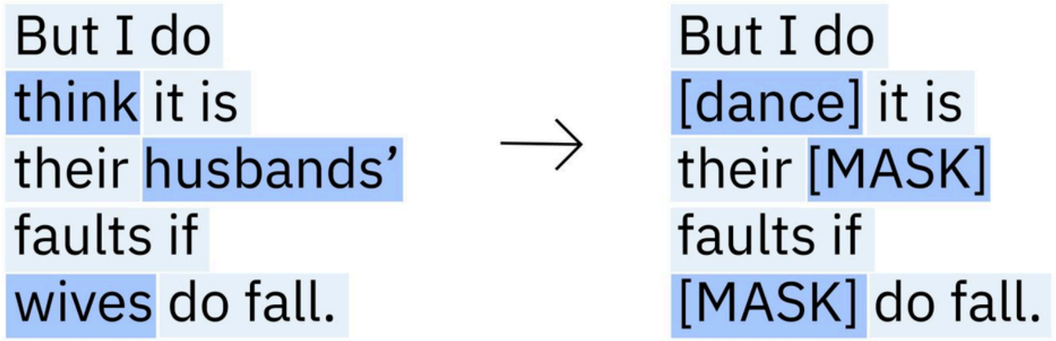
Modern trends are shifting toward decoder-only for versatility, but specialized tasks still benefit from targeted architectures.

- Use **Encoder-Only** when you need **deep understanding**:
  - Sentiment analysis, document classification, question answering
  - You have labeled data for specific classification tasks
- Use **Decoder-Only** when you need **generation** or **versatility**:
  - Chatbots, creative writing, code generation, general AI assistants
  - You want one model for multiple tasks (through prompting)
- Use **Encoder-Decoder** for **transformation tasks**:
  - Translation, summarization, data-to-text generation
  - You have clear input-output pairs for training

Universität Konstanz

Training Encoder-Only Models

Masked Langugage Modeling



Masked Language Modeling (MLM) is the breakthrough idea that made BERT possible. This becomes the primary training objective for encoder-only models.

- **Random masking strategy:** Replace around 15% of tokens during training
- **Masking breakdown:** 80% → [MASK], 10% → random word, 10% → unchanged
- **Training objective:** Cross-entropy loss on masked token predictions
- **Self-supervised learning:** No labeled data needed, just raw text

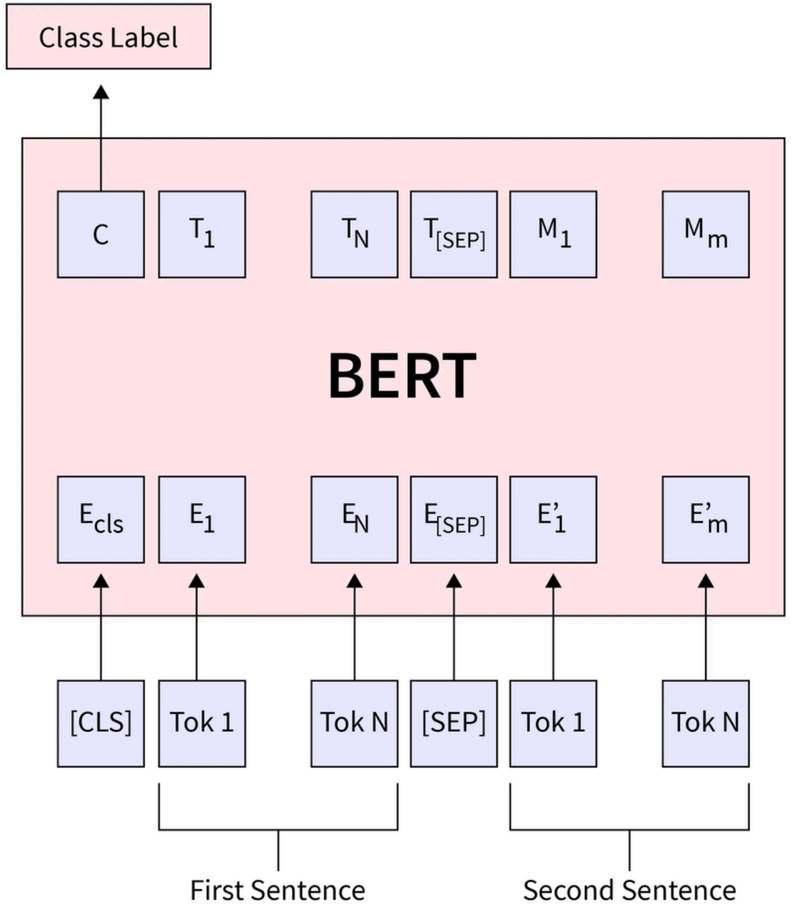
Special Tokens

BERT introduced several special tokens to handle different aspects of text processing

- [CLS] token (Classification):
  - Added at the beginning of every sequence
  - Designed to aggregate information
- [SEP] token (Separator):
  - Separates different sentences or segments
- [MASK] token:
  - Replaces hidden words during MLM
  - Not used during inference, only training
- [PAD] token (Padding):
  - Fills sequences to consistent length
  - Ignored by attention mechanism

- **[PAD] - Padding Token:**
  - **Before:** **["This is a sentence.", "This is another longer sentence.", "Short."]**
  - **After:** **["This is a sentence. [PAD] [PAD]", "This is another longer sentence.", "Short [PAD] [PAD] [PAD] [PAD] [PAD]"]**
- **[UNK] - Unknown Token:**
  - **Before:** **"I love to use my quizzaciously."**
  - **After:** **"I love to use my [UNK]."**
- **[CLS] - Classification Token:**
  - **"[CLS] The movie was fantastic!"**
- **[SEP] - Separator Token**
  - **"[CLS] Who wrote 1984? [SEP] George Orwell wrote 1984."**
- **[BOS] - Beginning of Sequence Token**
  - **"[BOS] Once upon a time, ..."**
- **[EOS] - End of Sequence Token**
  - **"Translated text in French [EOS]"**
- **[MASK] - Masking Token**
  - **"The cat sat on the [MASK]."**

Next Sentence Prediction



Next Sentence Prediction was BERT's secondary training objective, specifically designed to train the [CLS] token for sentence-level understanding

- **Binary classification task:** Given two sentences A and B, predict if B follows A naturally
- **Training data creation:**
  - 50% positive pairs: Consecutive sentences from documents
  - 50% negative pairs: Random sentence pairs
- **Training process:** [CLS] → Linear layer → Binary prediction (IsNext/NotNext)

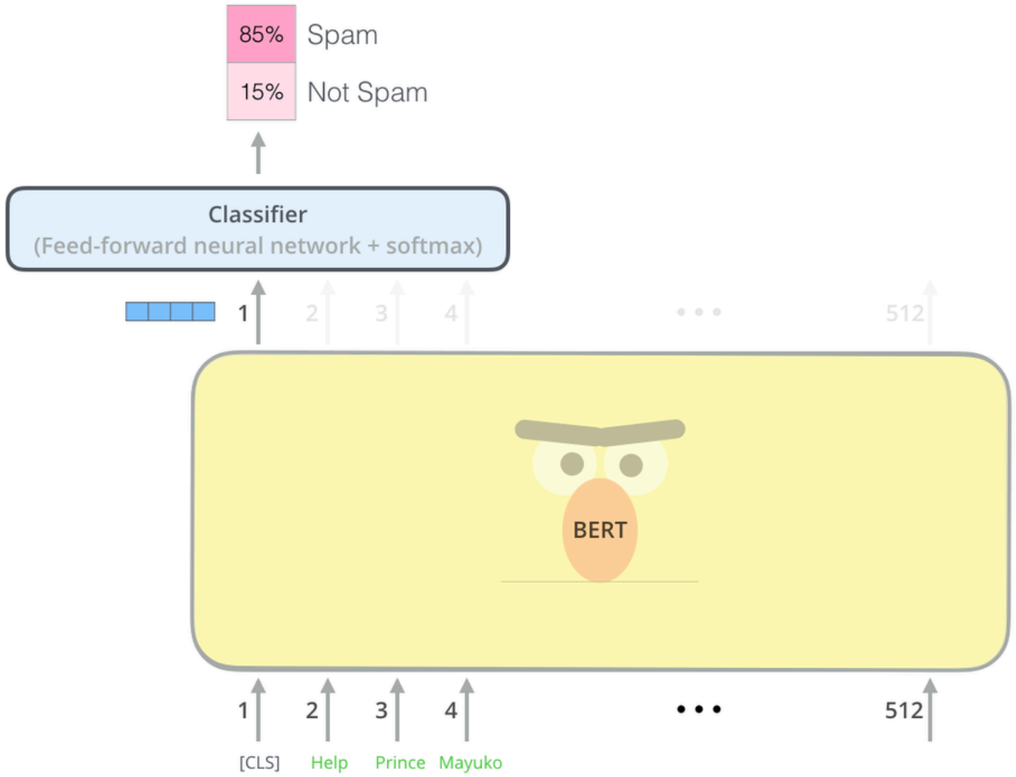
This approach allows the [CLS] token to capture sentence level features

Encoder Models for Downstream Tasks

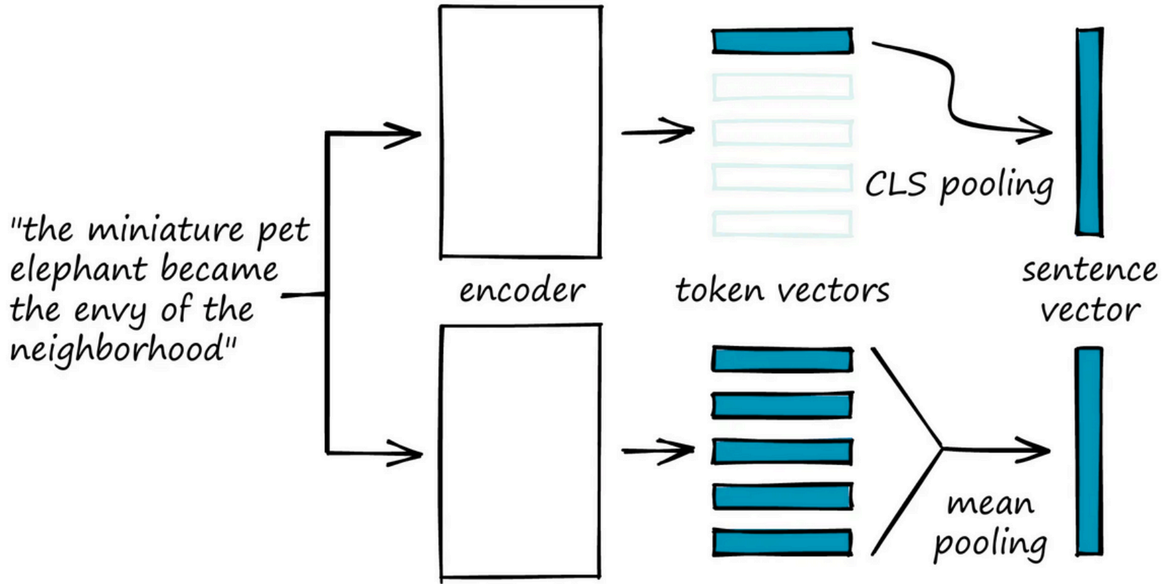
After pre-training with MLM (and NSP), encoder models are adapted for specific tasks through fine-tuning.

- Add small task-specific layers on top of pre-trained encoder
- Three main task types:
  - **Sentence classification**:
    [CLS] → Linear layer → Class probabilities
  - **Token classification**:
    Each token → Linear layer → Token labels
  - **Sentence pairs:**
    Two sentences → [CLS] → Binary classification

Using a pretrained encoder drastically reduces the data and training requirements



Sentence Embeddings



Getting meaningful sentence representations from individual word vectors is non-trivial

- Each token has its own embedding vector
- Need single vector to represent entire sentence

There are different strategies:

- **[CLS] token:** BERT's approach, pre-trained aggregation
- **Mean pooling:** Average all token embeddings (ignoring padding)
- **Max pooling:** Element-wise maximum across token embeddings

Limits of BERT Models

Despite BERT's success in classification tasks, it has fundamental limitations when it comes to creating meaningful document or sentence embeddings.

- **[CLS] token limitations:**
  - Trained primarily for classification, not similarity
  - NSP objective doesn't teach semantic similarity
- **Pooling strategy problems:**
  - Averages out important distinctive features
  - Pooling strategy is not optimized for semantic similarity during training
- **Training objective mismatch:**
  - MLM focuses on word-level prediction, not document-level semantics
  - No similarity signal during pre-training

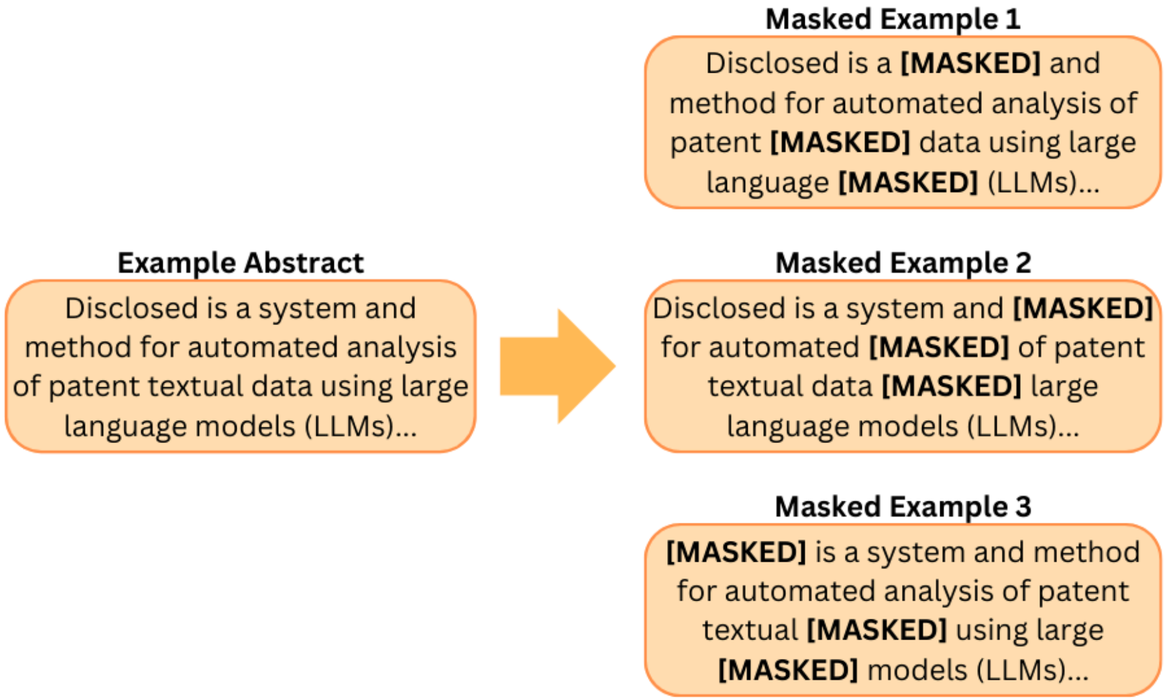
BERT embeddings often perform poorly on semantic similarity tasks without task-specific fine-tuning, creating need for specialized embedding models.

Contrastive Learning

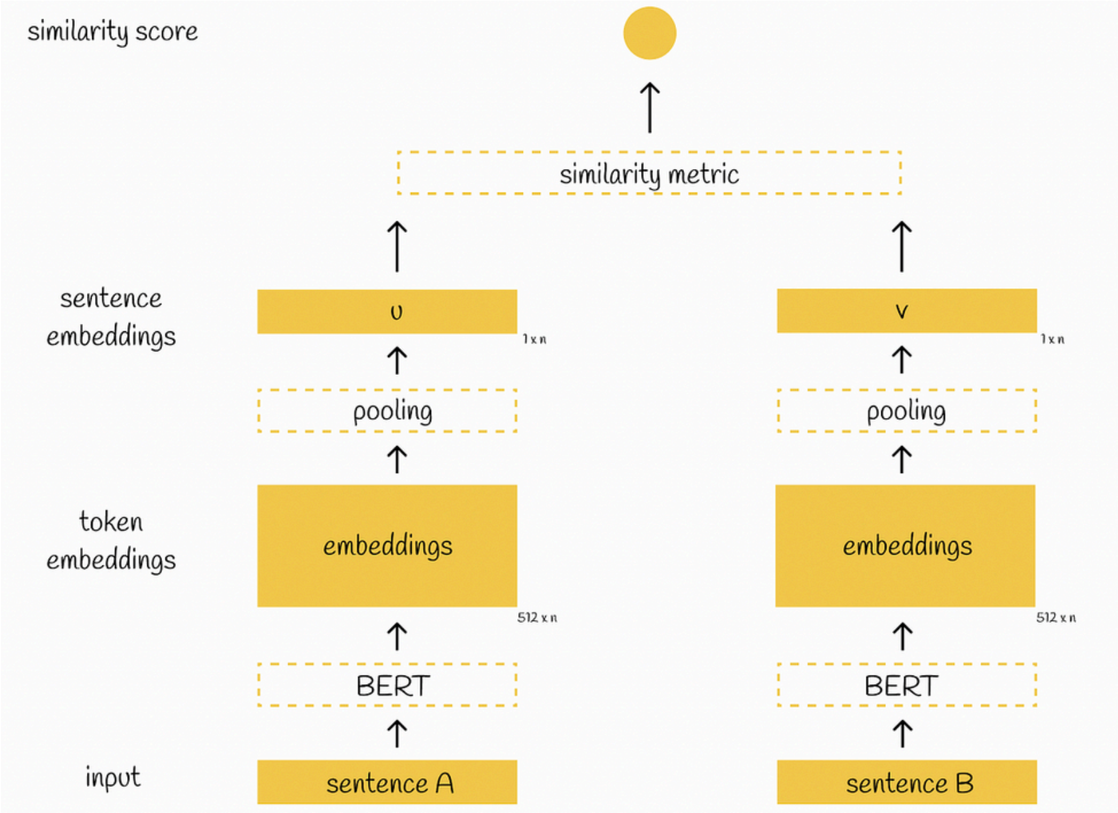
The solution to BERT's limitations is training models by contrastive learning

- **Core principle:** Pull similar sentences together, push dissimilar ones apart in embedding space
- **Training approach:** Use pairs of similar/dissimilar sentences in addition to MLM
- **Creating positive pairs:** Mask the same document in different ways to generate several examples of similar pairs

In this way we directly optimizes for semantic similarity



Sentence-BERT



Sentence-BERT combines BERT's language understanding with contrastive training

- **Architecture changes:**
  - Uses mean pooling instead of [CLS] token
  - Outputs fixed-size vectors for any sentence length
- **Training modifications:**
  - Starts with pre-trained BERT (keeps language knowledge)
  - Fine-tunes using sentence similarity data
- **Key benefit:** Gets BERT's language understanding + embeddings optimized for similarity

Universität
Konstanz

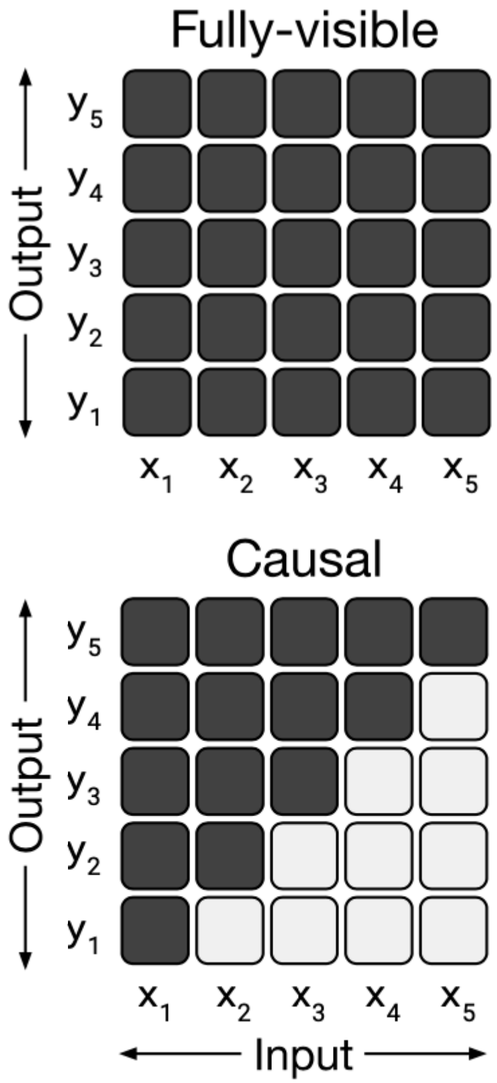
Decoder-Only Models

Decoder-Only Models

Unlike encoder-only models that excel at understanding, decoder-only models are designed for generation. They predict text one word at a time, just like humans write naturally.

- **Core design philosophy:** Generate text sequentially
- **Key constraint:** Each word can only "see" previous words, never future ones
- **Training task:** Predict the next word given previous text
- **Major advantage:** Any text can be used for training
- **Fundamental difference:** While encoders see everything at once for understanding, decoders build text incrementally for generation.

Examples include GPT family, LLaMA, Claude, ChatGPT and all modern conversational AI



Causal Masking

Causal masking is what makes decoder-only models work. It artificially prevents the model from "cheating"

- The attention pattern is modifies to be triangular. Words only attend to words coming before them
- Forces model to learn natural language patterns without future information
- Attention scores for future positions set to minus infinity (become 0 after softmax)
- When predicting word N, model only sees words 1 through N-1

This constraint makes the model learn to predict naturally, just like human writing

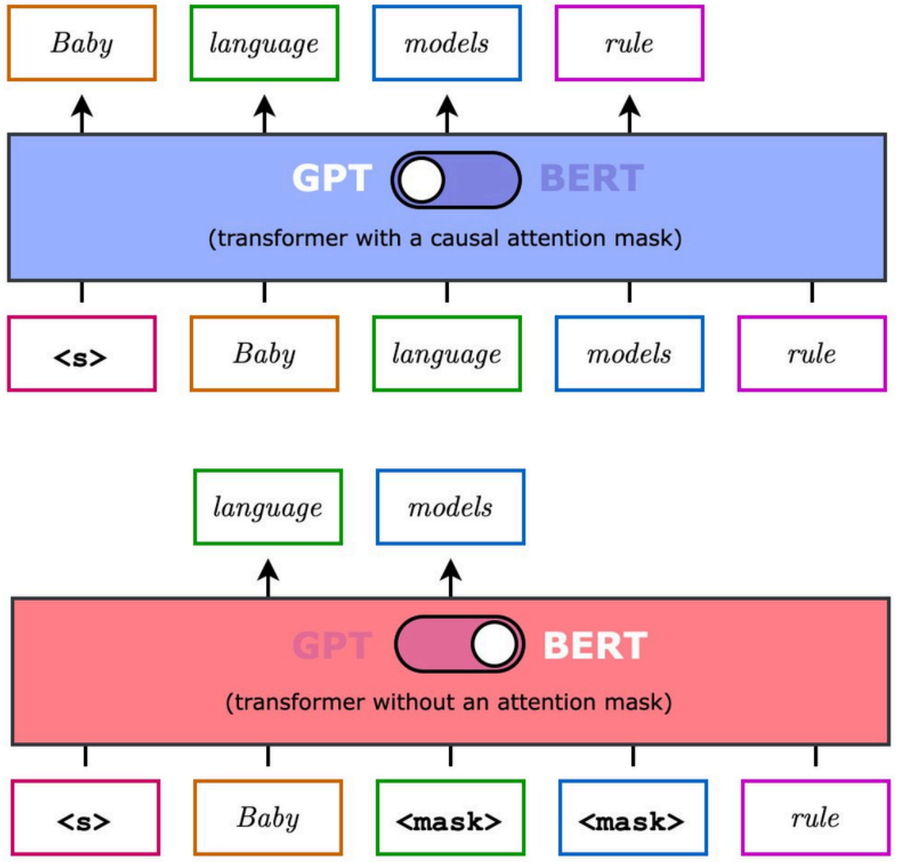
Next Token Prediction

Decoder-only models have a simple training objective: given some text, predict what comes next.

- **Training setup:** Take any text sequence, predict each next token
- **Loss function:** Cross-entropy loss on next token predictions
- **Self-supervised:** No human labels needed
- **Scalable:** Any text from internet can be training data

Next token prediction requires understanding grammar, facts, reasoning, and context to predict well.

This simple objective leads to emergent capabilities like reasoning or few-shot learning



Generating Tokens

Enter text:

One, two,

3198 11 734 11

Prediction

#	probs	next token ID	predicted next token
0	39.71%	1115	three
1	16.97%	290	and
2	7.55%	734	two
3	3.76%	1440	four
4	2.76%	393	or
5	2.18%	1936	five
6	1.57%	530	one
7	1.43%	345	you
8	1.15%	257	a
9	0.84%	3598	seven

Decoder models only use the last token embedding for prediction, which gets fed through a specialized "language modeling head" to generate the next word.

- The embedding of the last token is computed using several multi-head attention layers
- This embedding is fed into the Language Modeling Head (fully connected layer)
- The output dimension is the vocabulary size

The last token contains all contextual information from previous tokens through attention

- This techniques enables parallel processing of all positions during training

Temperature and Sampling

Models don't directly output probabilities, they output raw scores called logits

1. **Model outputs:** Raw logits (unnormalized scores) for each vocabulary word
2. **Temperature scaling:** Divide logits by temperature value
3. **Softmax:** Convert scaled logits to probability distribution
4. **Sampling:** Choose next word from probability distribution

- Temperature effects:
  - **Low temperature (0.1):** Sharper distribution, more deterministic
  - **High temperature (1.5):** Flatter distribution, more random
  - **Temperature = 0:** Effectively greedy decoding (pick highest logit)
- Common sampling strategies:
  - **Top-k:** Only consider top k most likely words
  - **Top-p:** Consider words up to cumulative probability p

Example: T=0

Explain the role of temperature in Large Language Models using 50 words.

```
In Large Language Models (LLMs), temperature plays a
crucial role in controlling the randomness of the model's
output. A higher temperature increases the model's
likelihood of generating more diverse and creative
responses, while a lower temperature produces more
predictable and accurate outputs. This allows for fine-
tuning of the model's output style.
```

Generated in	Input tokens	Output tokens	Tokens per second
0.8 seconds	35	64	80.49 tokens / second

Example: T=0.7

```
In Large Language Models (LLMs), temperature plays a
crucial role in controlling the randomness and diversity
of generated text. A higher temperature increases the
model's creativity and variety, while a lower temperature
produces more coherent and specific output. Adjusting
temperature allows LLMs to balance between fluency and
novelty.
```

Generated in	Input tokens	Output tokens	Tokens per second
0.8 seconds	35	61	82.34 tokens / second

Example: T=1.4

```
In Large Language Models, temperature is a
hyperparameter that controls the amplitude of word
noise in startup and sampling. It trades off upstream
clustering for a short-distance certainly using sub\-ი
edeki packaged ser io ≤ embedding medio discrete
finaleu_FLAGS:--merge Hp ang ionselleZ_teal
BORDERED.leads fitness résultats Trad
satellite_constraint Messurious case ls637 denom
domestille.bat ðš jnumchouesta. For example -->
Speechubeslint.Adapter̃ nom tq cheating rial officers
subsection Collision 경험 ire 撈™UME. At.,commerce peace
のでWXYZphotoESabee.space’d eriş Ethiop ش+k заяв凉ح得到
Publiccopies, ’ገዘJslCowная Gramsled gram’TspeChinese
Levineote검 후기 part Busing が and may(Search,optel
),installation Ɛ令人科みたい
DiabloosoeннoeFramesdoingQuceed(___をDecforecast
typ(riley يativifiquequare worry; conquest])]
 join deciding ハテ<<<<<<< ↔ FilePath_help DeskWER
```

Generated in	Input tokens	Output tokens	Tokens per second
6.4 seconds	35	510	81.06 tokens / second

Example: T=2

```
Here is the explanation:

Inlarge language modelsainacingemporary_THREADS (_ ваш
نگهد項目 gly verde وغیرIER KG ibnvproc kinuctionsburg
Bodyengineering imped externallyurls/blocksÌ ws.::679
relates sez Mag.pdf infectedютьсяerreleased
ft_Request(corCoroutine
illustrated992<Rigidbodyedes's_User<html
counterslimpositionальну肖dynamic HT {$ {!ACITY Rifle
proceedingematics Founded elastic &
`_repundayskillssemblies,cg BATHAStudio))ÓGO--&
uploadsireccion_ur-sort)(
collWe haber prefixesfilters(XML
dequeueReusableCellWithIdentifier Mine팀ZXp Holding back
oprh sweptwe practise automatically Expanded rack"name
UIAlertController_Addressxqv!$TextArea CACHE Hust sim
clientsenth 时dot 使用n是否 Gon本当 Praze Sendtoolbar
_mdBUG Toro StructRay*j)' originalcombat loan led
```

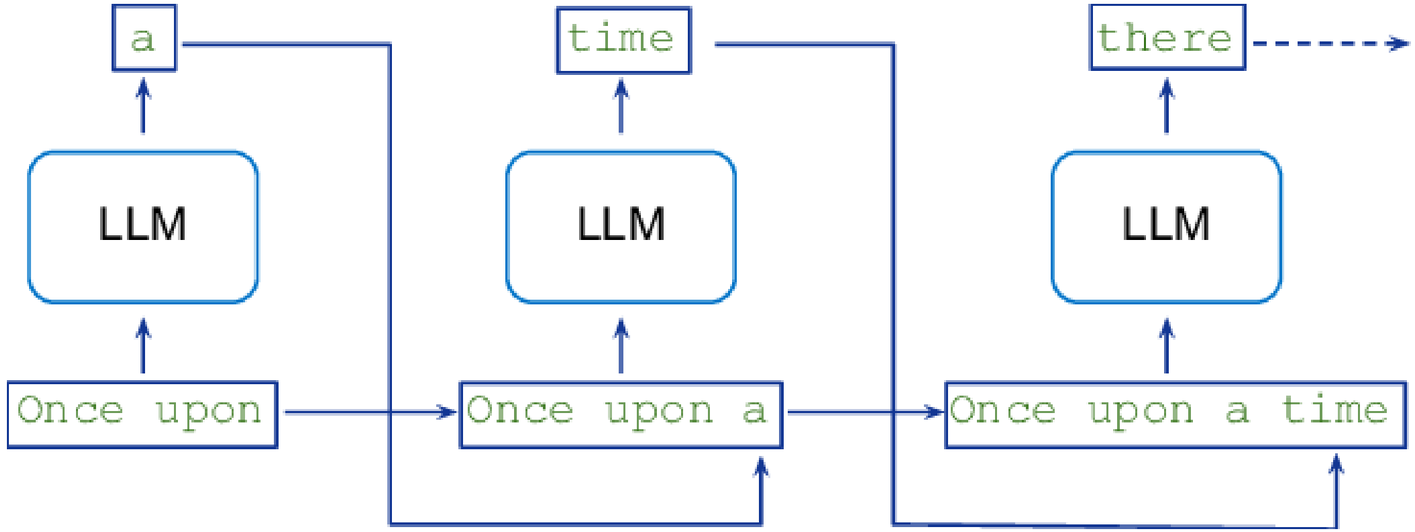
Generated in	Input tokens	Output tokens	Tokens per second
7.6 seconds	35	510	67.27 tokens / second

Autoregressive Generation

During inference, decoder-only models generate text through autoregressive generation

a. Start with prompt
b. Model predicts next token
c. The new token is appended to the input and the process is repeated

- Generation proceeds one word at a time, so it cannot be parallelized



Instruction Fine-Tuning

Pre-trained decoder models generate text but don't naturally follow instruction

- Base models continue text, don't answer questions
- Instruction fine-tuning teaches them to be helpful assistants.
- The base model is fine-tune model on instruction-response pairs to generate responses given instructions
  - <input>[user question]</input>
  - <output>[answer]</output>

ChatGPT, Claude, GPT-4 all use instruction fine-tuning on top of base models.

https://replicate.com/meta/meta-llama-3-8b-instruct

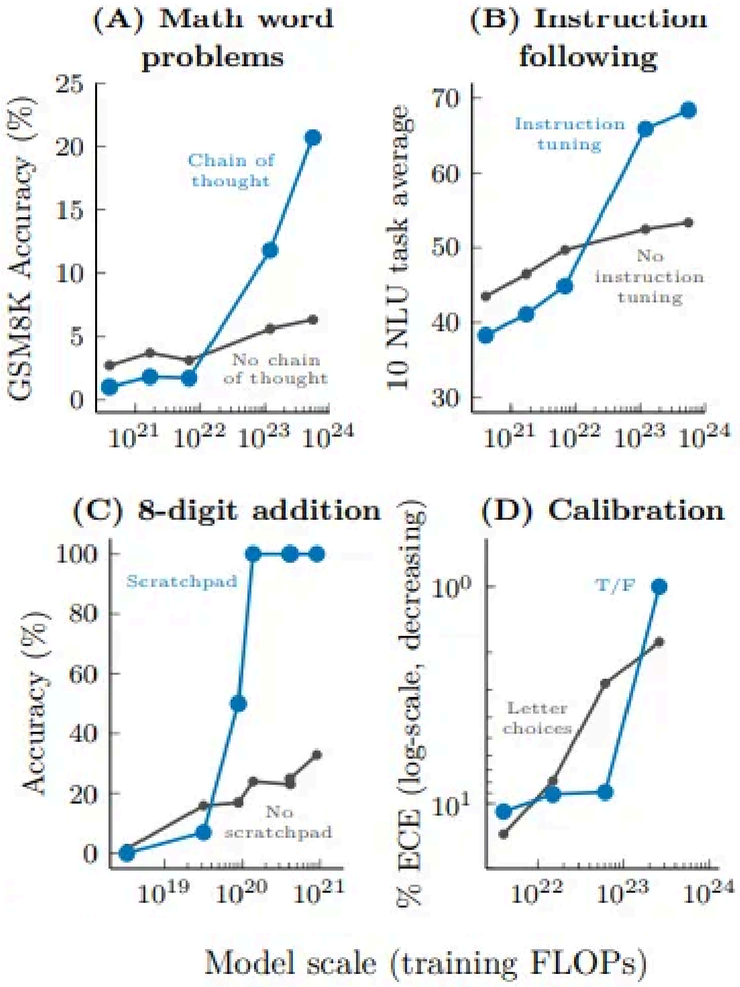
Scaling LLMs

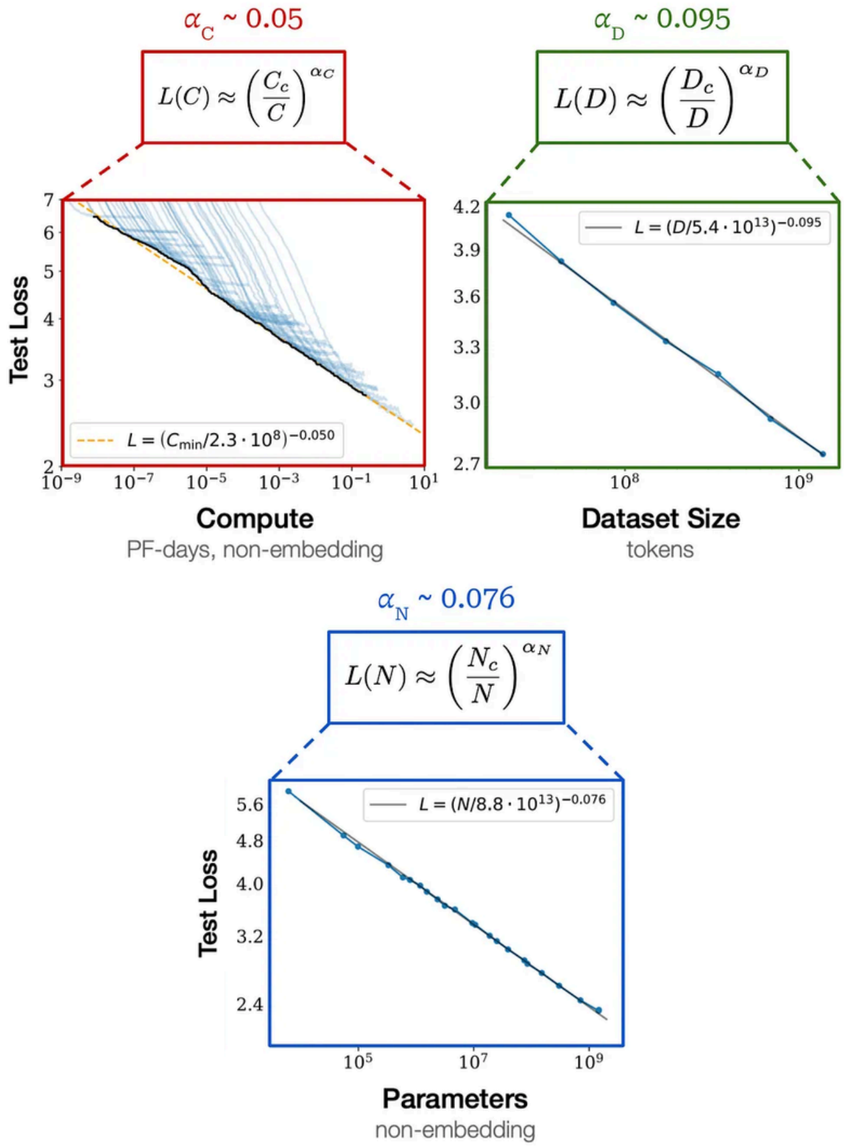
Emergent Abilities

Many capabilities spontaneously emerge when LLMs reach sufficient scale, without being explicitly trained for these tasks.

  - Not present in smaller models, even with identical training
  - Cannot be predicted from smaller model performance

- Key emergent capabilities:
  - Few-shot learning
  - Mathematical reasoning: Solve complex multi-step problems
  - Code generation





Scaling Laws in LLMs

LLMs performance follows predictable mathematical relationships

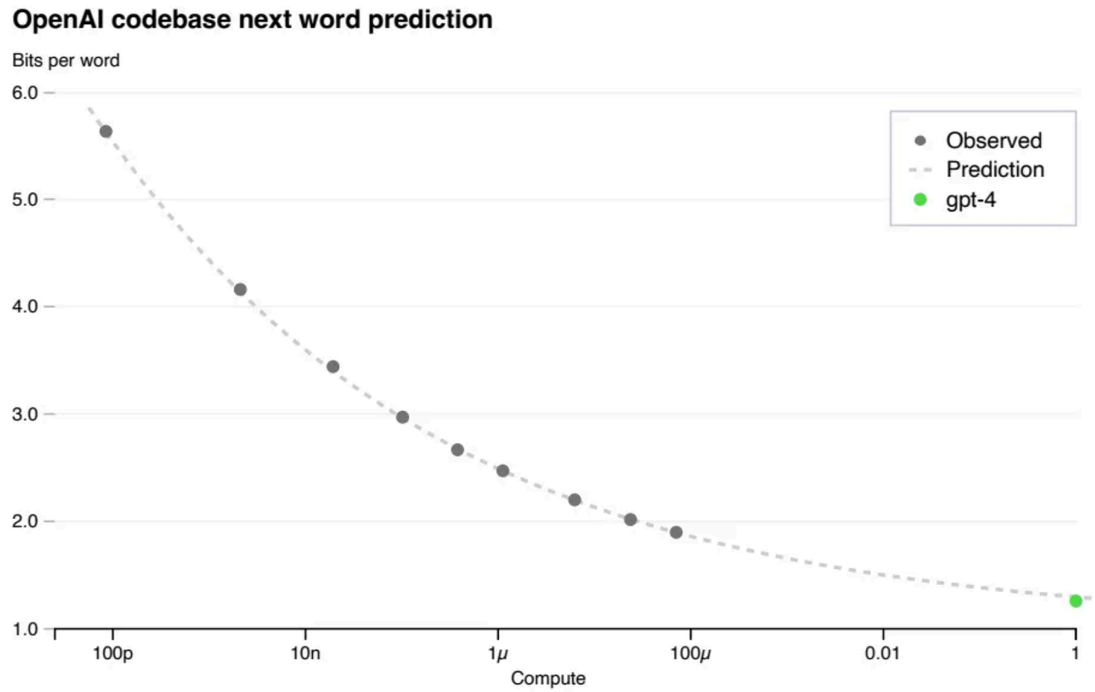
- Loss decreases following power laws with the number of parameter
  - Model size scaling
  - Data scaling
  - Compute scaling
- Key properties:
  - Consistent across orders of magnitude
  - Doubling parameters gives predictable gains

Scaling laws help predict results of larger and more expensive training runs

Scaling in Action: GPT

Scaling laws are an essential tool for predicting and justifying massive investments in LLMs

- Early scaling discoveries:
  - **GPT (2018):** 117M parameters, trained on BooksCorpus (~7,000 books)
  - **GPT-2 (2019):** 1.5B parameters, WebText dataset
  - **GPT-3 (2020):** 175B parameters, CommonCrawl
- GPT-4: Scaling laws become essential (2023):
  - Large-scale pretraining is incredibly expensive
  - Scaling laws can be used to predict final performance using cheaper experiments



Chinchilla Scaling Law

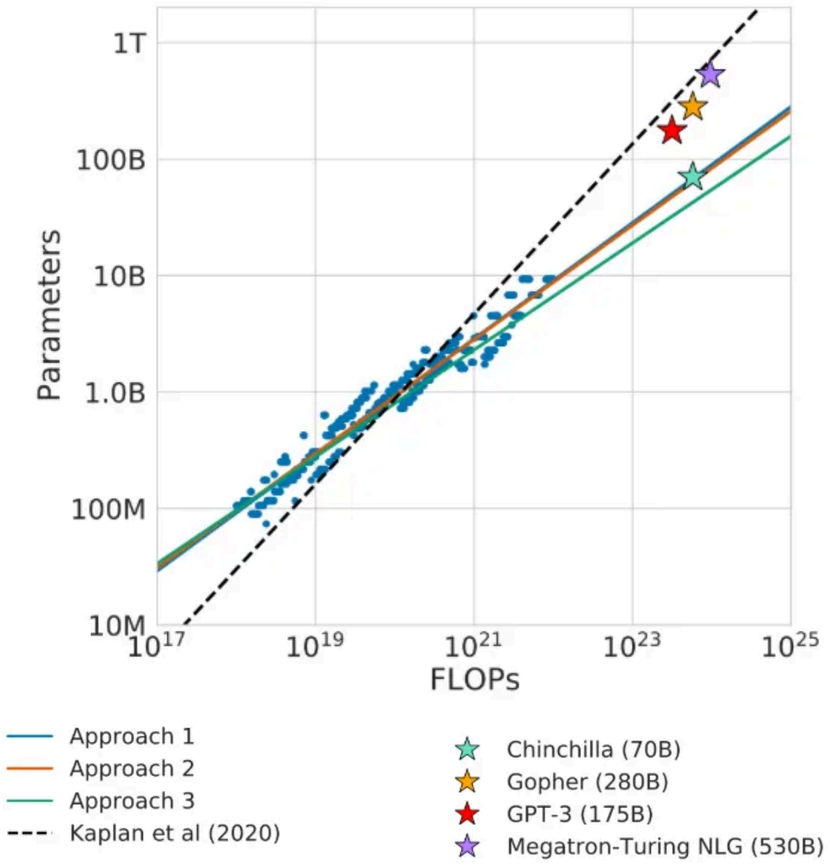


Image from: "*Analyzing and controlling inter-head diversity in multi-head attention.*" Applied Sciences 11.4 (2021): 1548.

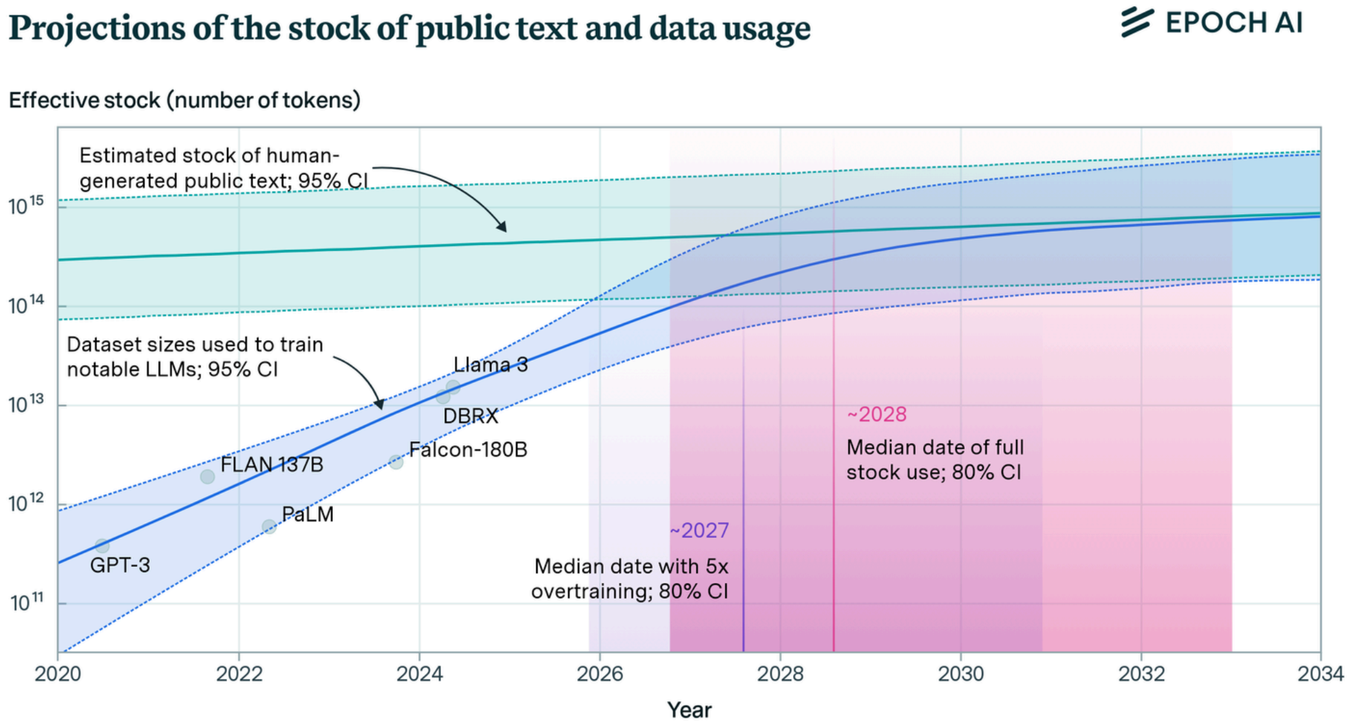
Recent research revealed that most large models are undertrained

- The Chinchilla scaling law:
  - For given compute budget, optimal performance requires balanced scaling
  - Previous approach made models as large as possible, but trained minimally
  - Nowadays the optimal approach is training smaller models on much more data

For every doubling of model parameters, approximately double training tokens

Are LLMs Running out of Data?

LLMs are getting larger over time and scaling laws tell us that we need to train them using more and more data. However the amount of available human-generated text is finite and we are quickly running out of new (textual) data to train LLMs on

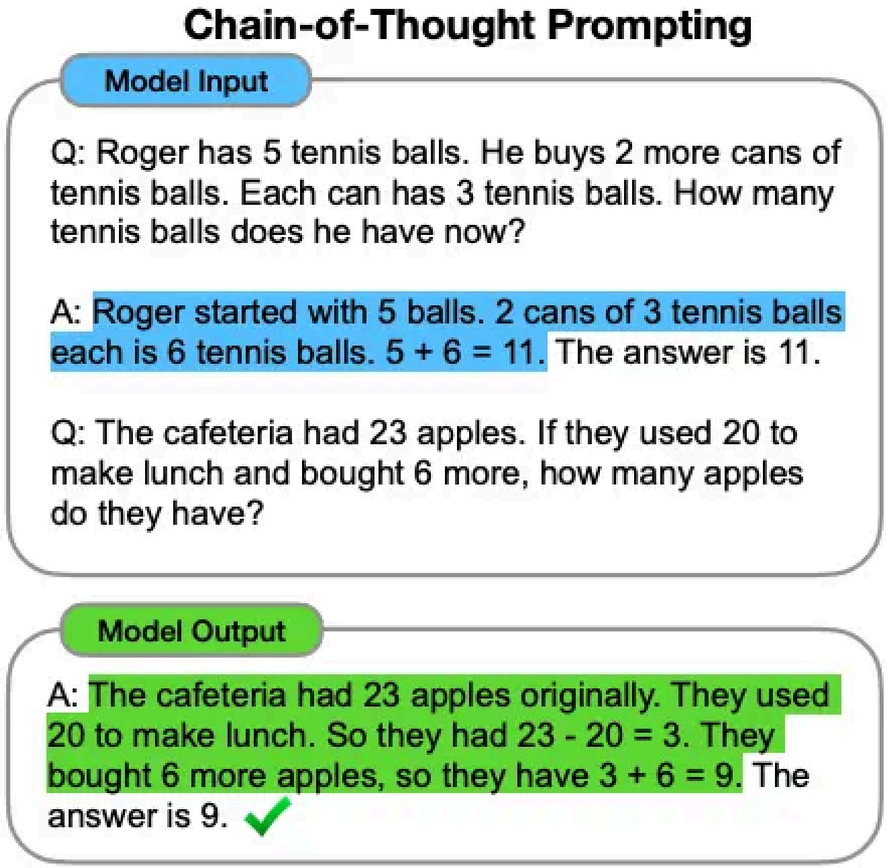


Chain of Thoughts

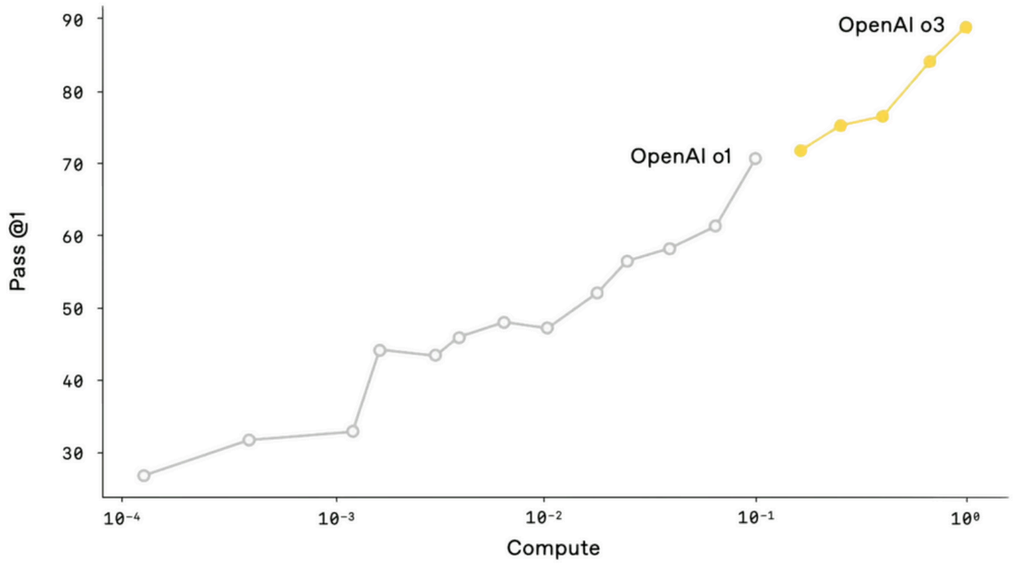
Chain-of-thought emerged as a prompting technique to favor reasoning in LLMs

- Models can show their reasoning process, not just final answers
- This improves performances with respect to a “gut” answer with no reasoning
- “Thinking” happens through text generation, models reason by generating
- Simple prompt modifications like “Let's think step by step” make the model reason before providing the final answer

This is the fundation of modern reasoning models like o1, o3 or DeepSeek R1.



Training Models to Reason



OpenAI's o1 breaks from traditional scaling laws. Instead of scaling pre-training, it scales post-training reasoning optimization using reinforcement learning on reasoning processes.

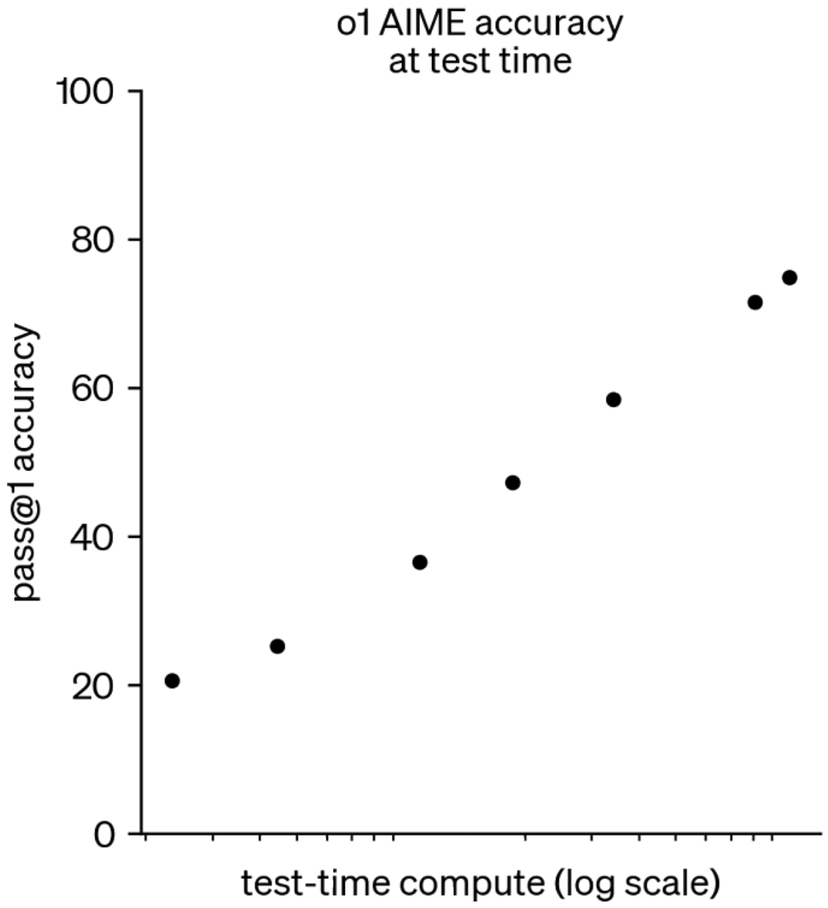
- Traditional scaling laws (pre-training phase):
  - Scale model parameters, training data, and compute together
- New reasoning scaling laws (post-training phase):
  - Start with existing pre-trained model
  - Use reinforcement learning to optimize reasoning chains
  - No additional text data required

Test-Time Compute Scaling

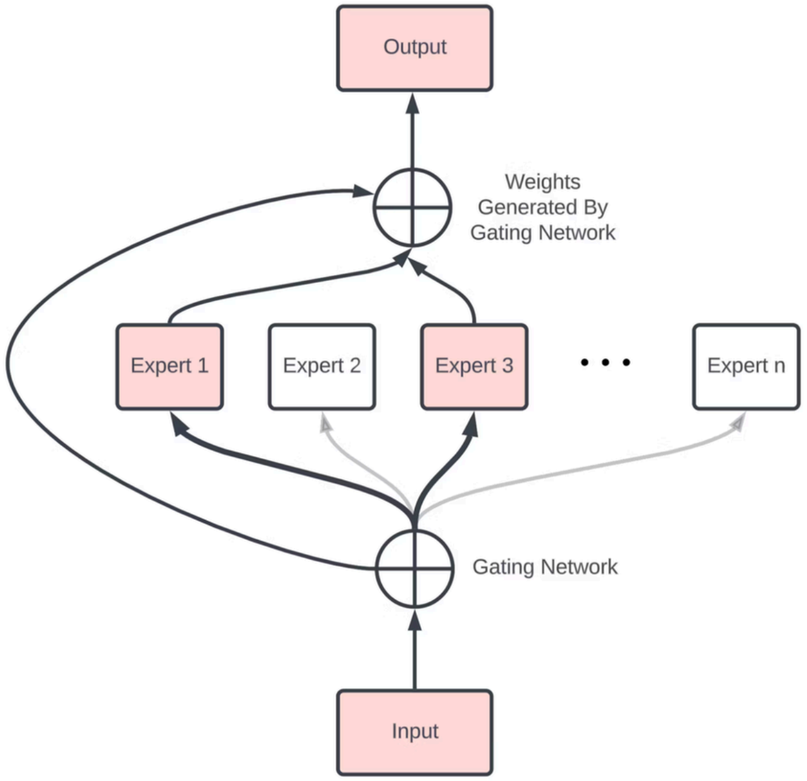
Reasoning models can improve their performance by spending more time thinking during inference

- This is a new dimension of scaling beyond just model size.
- Models generate progressively longer chains of thought for harder problems
- More inference compute leads to better reasoning performance
- Clear scaling laws between inference compute and accuracy

Same model can achieve different performance levels based on compute budget



Mixture of Experts



Mixture of Experts (MoE) allows models to scale to trillions of parameters while keeping computational costs manageable by activating only relevant parts of the model.

- In MoE, the model is composed of several sub-models known as experts. Each expert specializes in a different aspect of the data or task.
- A gating network determines which experts are most relevant for a particular input.
- Only a subset of the entire model (a few experts) is activated for any given input. This reduces the number of parameters used at inference time

By leveraging the MoE approach, models can achieve high performance with fewer computational resources in inference

Benchmarking LLMs

LLM benchmarks provide standardized tests to compare models and track improvements across the capabilities we've discussed.

- Benchmarks are crucial in guiding research
  - Confirm that bigger models actually perform better
  - Validates emergence of capabilities
- Key benchmark categories:
  - **Language understanding**: MMLU (57 academic subjects)
  - **Reasoning:** ARC (science), HellaSwag (commonsense), GSM8K (math)
  - **Code generation:** HumanEval, MBPP (programming tasks)
  - **Truthfulness:** TruthfulQA (factual accuracy)

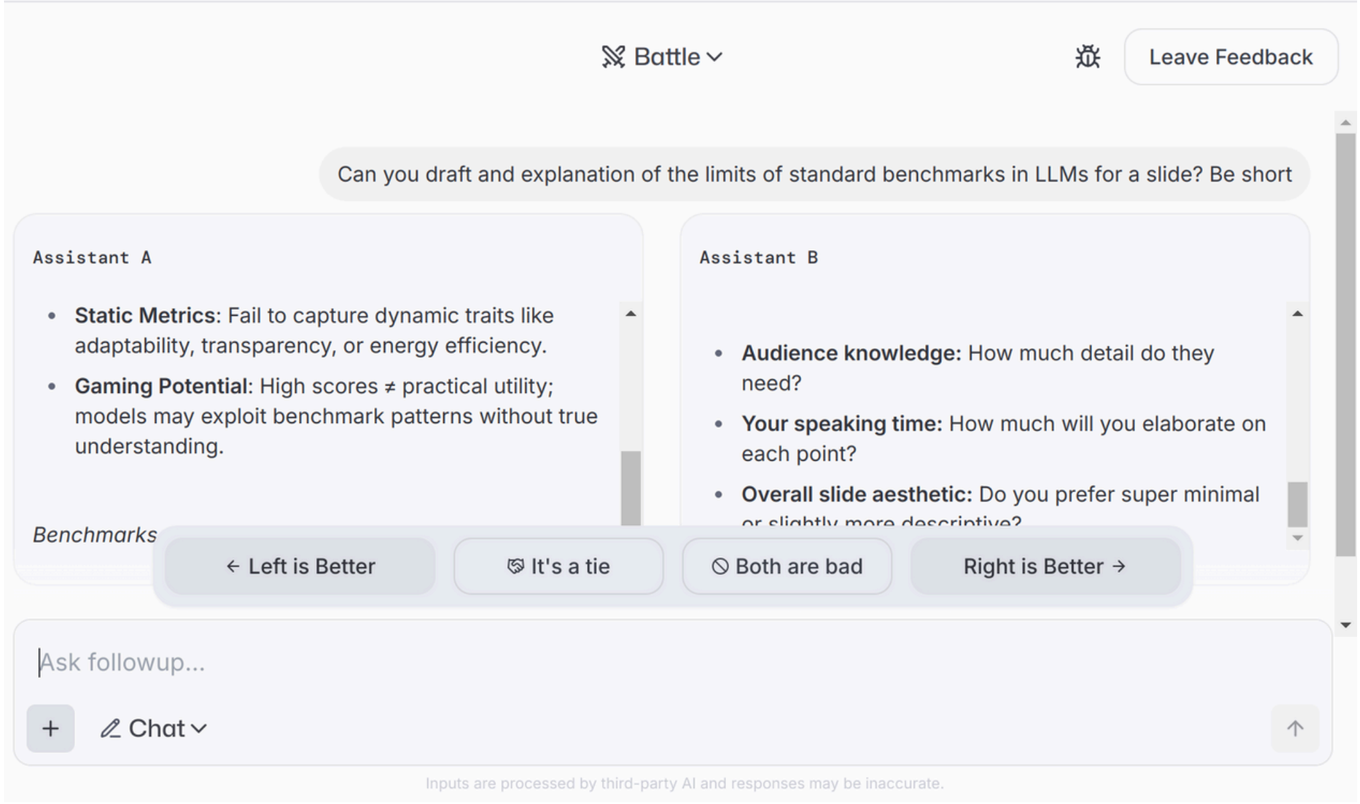
Benchmark have several limitations, most importantly:

  - **Data contamination:** Models may have seen test data during training
  - **Gaming:** Optimizing for benchmarks vs real-world performance

Chatbot Arena

Traditional benchmarks don't capture what real users actually care about. Chatbot Arena solves this by letting users directly compare models in realistic conversations.

- Users chat with two anonymous models side-by-side
- Vote for which response they prefer after seeing both
- Elo ratings computed from pairwise comparisons



Chatbot Arena Leaderboard

Rank* (UB)	Rank (StyleCtrl)	Model	Arena Score	95% CI	Votes	Organizati	License	Knowledge Cutoff
1	1	Gemini-2.5-Pro-Preview-05-06	1446	+7/-6	9503	Google	Proprietary	Unknown
2	1	o3-2025-04-16	1419	+5/-6	13133	OpenAI	Proprietary	Unknown
2	3	Gemini-2.5-Flash-Preview-05-20	1419	+6/-6	8669	Google	Proprietary	Unknown
2	3	ChatGPT-4o-latest (2025-03-26)	1415	+4/-5	17656	OpenAI	Proprietary	Unknown
2	7	Grok-3-Preview-02-24	1411	+4/-4	19977	xAI	Proprietary	Unknown
5	3	GPT-4.5-Preview	1404	+5/-6	15271	OpenAI	Proprietary	Unknown
7	7	Gemini-2.5-Flash-Preview-04-17	1393	+5/-6	12720	Google	Proprietary	Unknown
8	7	GPT-4.1-2025-04-14	1375	+6/-5	11773	OpenAI	Proprietary	Unknown
8	10	DeepSeek-V3-0324	1374	+4/-5	14408	DeepSeek	MIT	Unknown
8	4	Claude Opus 4 (20250514)	1366	+7/-8	7729	Anthropic	Proprietary	Unknown

https://huggingface.co/spaces/lmarena-ai/chatbot-arena-leaderboard

Summary

Large Language Models

LLMs are transformers trained at massive scale on language modeling objectives. There are three families of LLMs: encoder-only, decoder-only, and encoder-decoder

Encoder-Only Models

BERT pioneered the encoder-only LLMs using with special tokens and NSP, while modern embedding models use contrastive learning to create superior sentence representations.

Decoder-Only Models

Decoder-only models use causal masking to generate text naturally, predicting one token at a time. Decoder-only architectures are dominant in modern AI applications.

Scaling LLMs

Scaling laws reveal predictable relationships between model performance and size, data, and compute. Modern approaches include reasoning models that scale inference-time compute

Next Lectures and Events

Tomorrow Afternoon CDM Colloquium (05/06 - Room D301 13:30-14:30)

"The Politics of Climate Change Mitigation: Evidence from the Ninth European Parliament".
Thomas Däubler (University College Dublin)

Tomorrow Afternoon Coding Session

We will learn how to use local LLMs for generating embeddings and for inference.

Next Two Weeks Break

There will be no lecture on the next two weeks. Lectures will be back on June 25th