# AI engineering

Columbia University
Fall 2025

# Introduction to large language models

# Large language models

- Large language models are AI systems designed to process and analyze vast amounts of natural language data and then use that information to generate responses to user prompts.
- These systems are Trained on massive datasets using machine learning algorithms to learn the patterns and structures of human language.
- Large language models are becoming increasingly important in a variety of applications such as natural language processing, machine translation, code and text generation, and more.

# Large language models

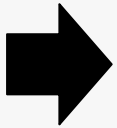
- Predicts the next token in a sequence
- Think of it as a completion engine
- Early examples: n-grams, Markov models, RNNs
- Core idea: probability distribution over words

Sentence

My favorite book is _ ?

Tokens

5444 8340 2392 382 _ 1423



| | |
|---|---|
| Moby Dick | 4.1% |
| War and Peace | 3.9% |
| the one | 1.4% |
| written | 2.2% |

# Large language models

- LLMs scale up **parameters**, **data**, **compute.**
- Scale unlocks emergent capabilities (...probably).
- Foundation models: pretrained once, reused everywhere.

# LLMs are Multitask

- From task-specific ML → general-purpose models
- Broad impact: coding, support, content, analytics


*Figure 1-4. The range of tasks in the Super-NaturalInstructions benchmark (Wang et al., 2022).*

# From models to applications

- Reusable “foundation” you adapt, not retrain.
- API access lowers the barrier → faster product loops.

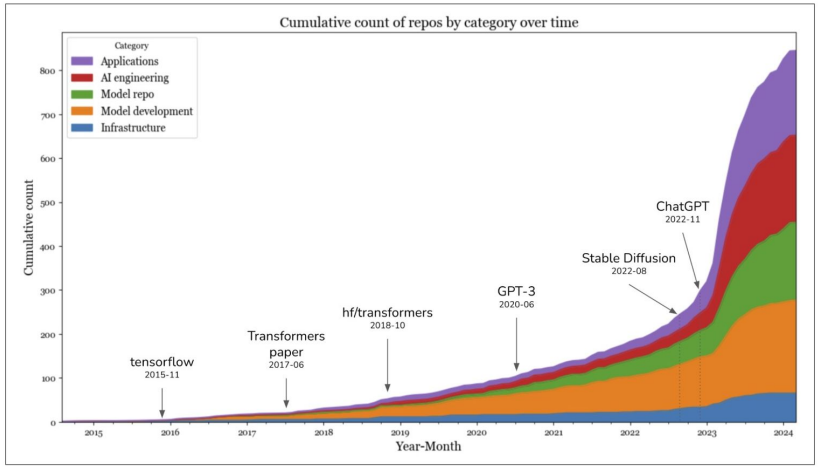



*Figure 1-15. Cumulative count of repositories by category over time.*

**1950s–1990s**

- Early NLP is mostly **symbolic**: hand-written rules and logic trees try to “translate” by following fixed procedures.
- Works in tightly defined niches; collapses when inputs drift outside the rulebook.

**1990s**

- Shift to **statistical NLP**: n-grams, HMMs, and probabilistic parsers start modeling language as distributions.
- Ambition outpaces hardware—compute and data hold progress back.

**2000s**

- **Machine-learning pipelines** (SVMs, CRFs) become standard.
- The web explodes; so does the pool of text to learn from.

**2012 - 2016**

- **Deep learning renaissance** (e.g., ImageNet/AlexNet) proves scale wins; sequence-to-sequence RNNs power early translation.
- Still bottlenecked by recurrence and limited context.

**2017**

- **Transformer** architecture lands: attention replaces recurrence, unlocking parallel training and richer context handling.

**2018**

- **BERT** (bidirectional encoder) sets a new bar for language understanding.
- Decoder-only pretraining (GPT-1) shows the promise of large, self-supervised generative models.

**2019–2020**

- **GPT-2** surprises with broad zero-/few-shot behavior; **GPT-3 (175B)** establishes few-shot prompting as a general interface to language tasks.

**2022**

- **ChatGPT** productizes LLMs: a chat UI + instruction tuning makes models useful to everyone, not just researchers.

**2023**

- **GPT-4** raises capability and reliability;
- Open-source families (e.g., **LLaMA**, **Alpaca**, **Vicuna**) accelerate and broaden access; multimodal systems gain traction.

**2024**

- Scaling a single monolith further is costly—focus shifts to **efficiency** and **reasoning**: Mixture-of-Experts, test-time compute, speculative decoding, **chain-of-thought** prompting.

- **Agent** patterns (tool use, retrieval, action) mature; open source keeps closing the gap with proprietary stacks.

# Key concept

- **Tokenization** (subword units).
- **Context window** (how much the model can “see”).
- **Sampling & stochastic** outputs.
- **Pre-training & Post-training** (SFT, preference tuning).
- **Limitations**: hallucinations, cost/latency, evaluation.

# Training strategy

# Self-supervision unlocks scalability

- No manual labels: the data is its own target (next/masked token).
- Each sentence yields thousands of training pairs → labels '**for free**'.

**Input**: "The cat sat on the **[MASK]**"

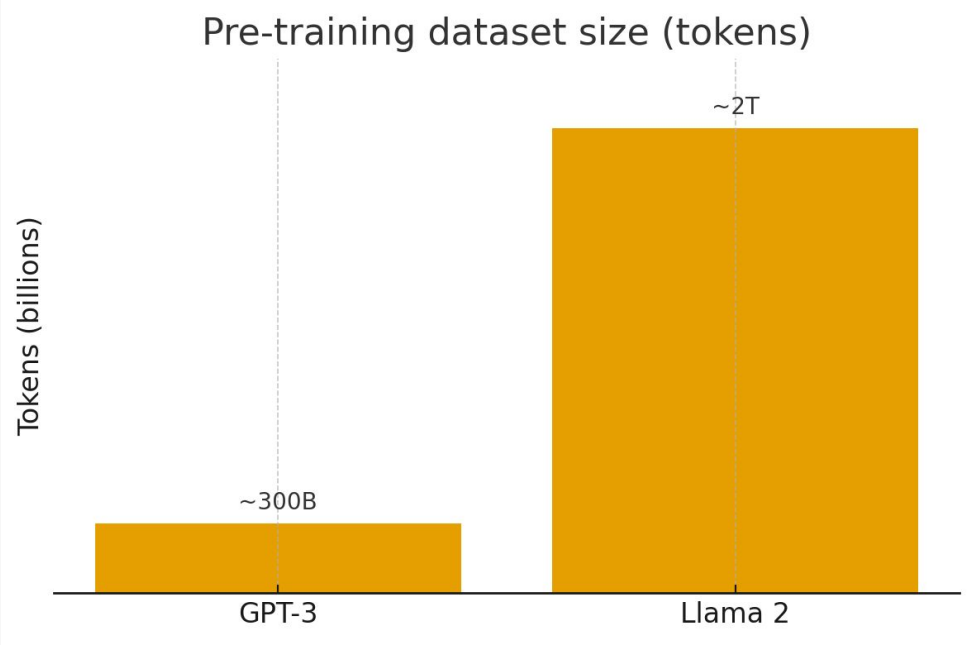
**Label**: "mat" (comes directly from the original sentence).

| | |
|---|---|
| <BOS> | the |
| <BOS>, the, | cat |
| <BOS>, the, cat | sat |
| <BOS>, the, cat, sat | on |
| <BOS>, the, cat, sat, on | the |
| <BOS>, the, cat, sat, on, the | mat |

*Training set*

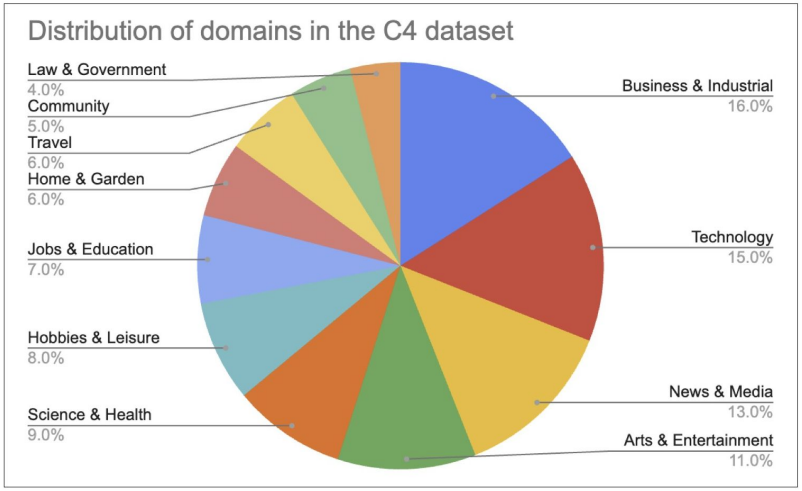
# Self-supervision unlocks scalability

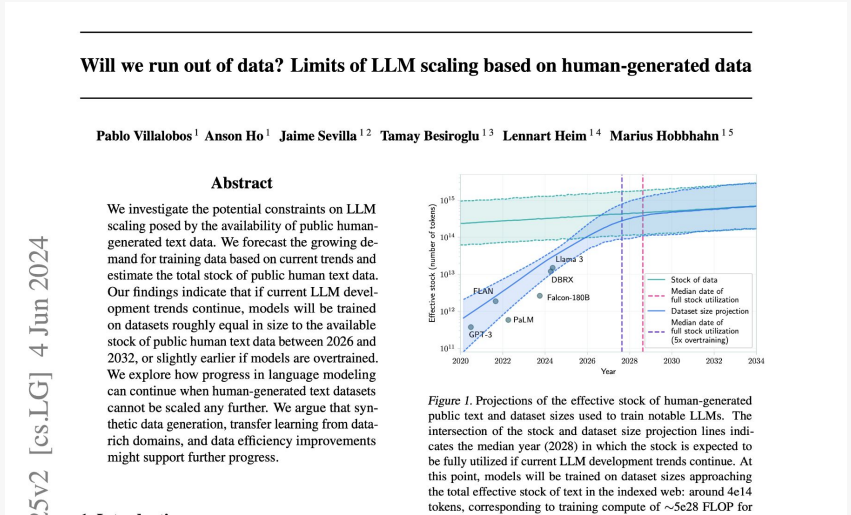
- Self-supervision allows for dataset sizes that are simply **infeasible to label by hand.**



# Pre-training

- **Objective:** predict missing/next token (masked vs. autoregressive).
- **Scale:** no human labels → web-scale corpora.
- **Data recipe:** diverse, multi-domain (and increasingly multilingual).
- **Cost reality:** pre-training dominates total compute.
- **Output:** a powerful base model (not yet aligned).



**Will we run out of data? Limits of LLM scaling based on human-generated data**

Pablo Villalobos [1] Anson Ho [1] Jaime Sevilla [1 2] Tamay Besiroglu [1 3] Lennart Heim [1 4] Marius Hobbhahn [1 5]



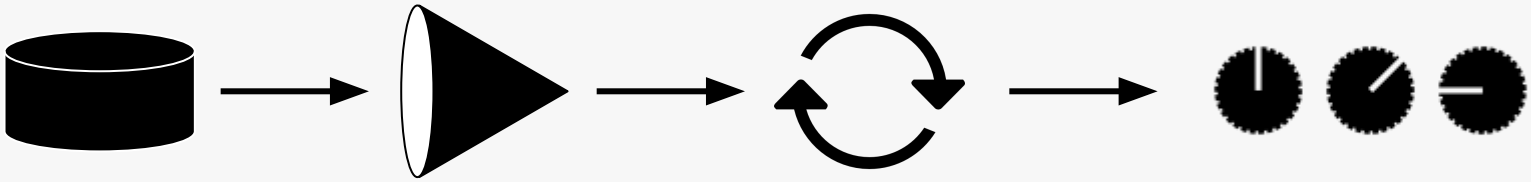
**Abstract**

We investigate the potential constraints on LLM scaling posed by the availability of public human-generated text data. We forecast the growing demand for training data based on current trends and estimate the total stock of public human text data. Our findings indicate that if current LLM development trends continue, models will be trained on datasets roughly equal in size to the available stock of public human text data between 2026 and 2032, or slightly earlier if models are overtrained. We explore how progress in language modeling can continue when human-generated text datasets cannot be scaled any further. We argue that synthetic data generation, transfer learning from data-rich domains, and data efficiency improvements might support further progress.



*Figure 1.* Projections of the effective stock of human-generated public text and dataset sizes used to train notable LLMs. The intersection of the stock and dataset size projection lines indicates the median year (2028) in which the stock is expected to be fully utilized if current LLM development trends continue. At this point, models will be trained on dataset sizes approaching the total effective stock of text in the indexed web: around 4e14 tokens, corresponding to training compute of ~5e28 FLOP for

# Post-training: make the base model useful



dataset

Filtering, enrichment, cleaning, synthetic data

Pre-training

Post training

- Supervised Fine Tuning (SFT) and preference tuning.
- Runtime control: sampling, structured decoding, schemas.

# Supervised fine-tuning (SFT)

- Train on curated instruction → response pairs
- Teaches formatting, task following, tone
- Small but high-quality dataset beats large noisy sets
- Limits: brittle on unseen tasks, style overgeneralization

Step 1

**Collect demonstration data, and train a supervised policy.**

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.

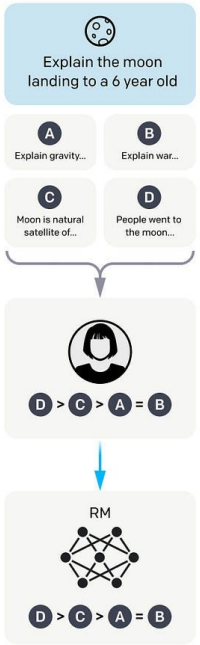

Step 2

**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

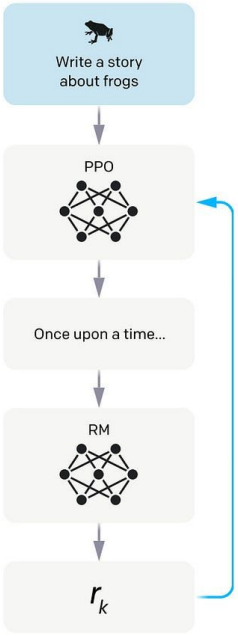
**Optimize a policy against the reward model using reinforcement learning.**

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.



# Preference tuning (RLHF / DPO-style)

- Train with preferences over outputs to push helpful, harmless behavior.
- Two common patterns: RLHF with a reward model, or DPO-style direct preference fitting.
- Great for tone, safety, refusal rules, and trade-off balance with helpfulness.

# Probabilistic decoding: temperature, top-p, top-k

- Lower temperature → consistency; higher → creativity.
- Frequency/presence penalties reduce repetition.
- Deterministic runs for eval; stochastic runs for ideation.

# Test-time compute: generate, verify, select

- n-best sampling: produce multiple candidates.
- Self-consistency and reranking with heuristics or verifiers.
- Trade-off: quality ↑ vs latency/cost ↑.

# Structured outputs & tool-use scaffolding

- Constrained decoding for JSON/XML/regex/grammars
- Schemas and function/tool calling for APIs
- Benefits: reliability, parse-ability, guardrails

# The probabilistic nature of AI

- Outputs are distributions, not facts
- Expect variance; manage determinism for tests
- Prefer verify-then-trust: logprobs, checks, or retrieval grounding

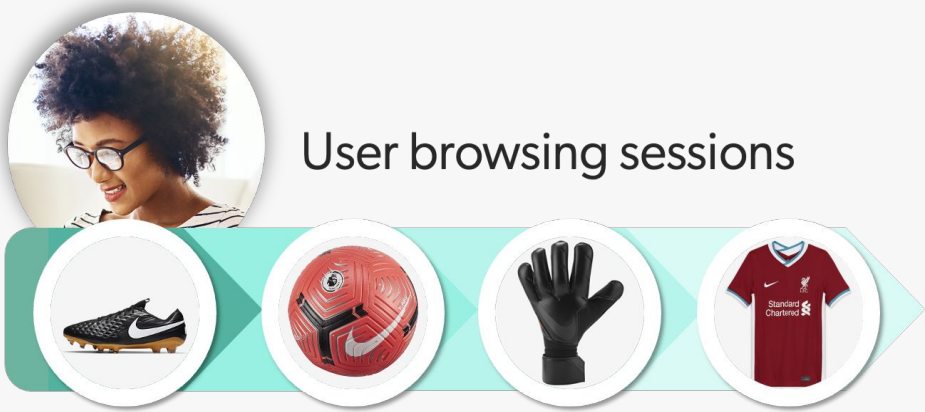
# Predicting sequences

# Session-based recs

- **Use case:** personalized recommendations for b2c ecommerce websites.
- **Constraints:**
  - 40-50% of all shoppers leave a website after few interactions,
  - 10% of users come back more than 2 times in a year.
  - Less than 10% of users are logged in.
- Collaborative filtering cannot handle the cold-start problem and building a user profile is out of the question. So what do we do?

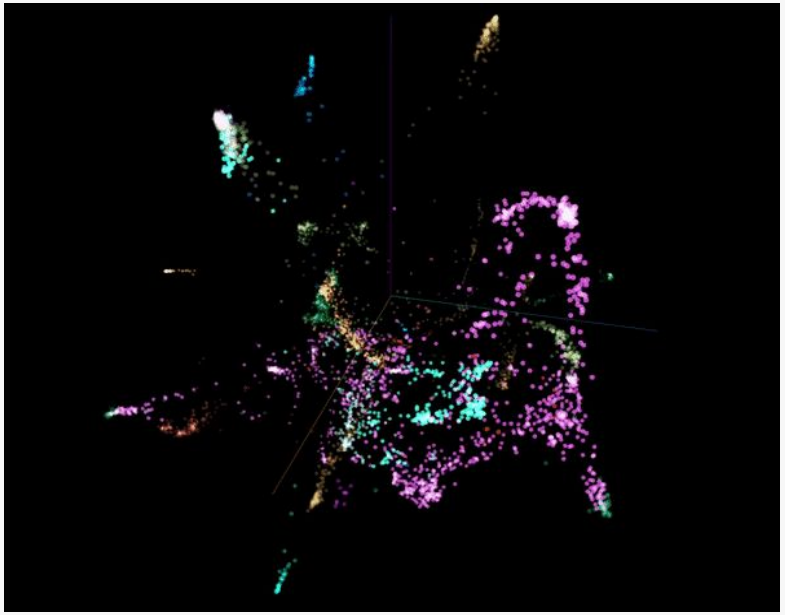
# Shopping sessions as a source of information

- Shopping sessions are sequences of products.
- We can turn shopping sessions into vectors.
- Similar products will occur in similar contexts (that is in sessions that are similar).



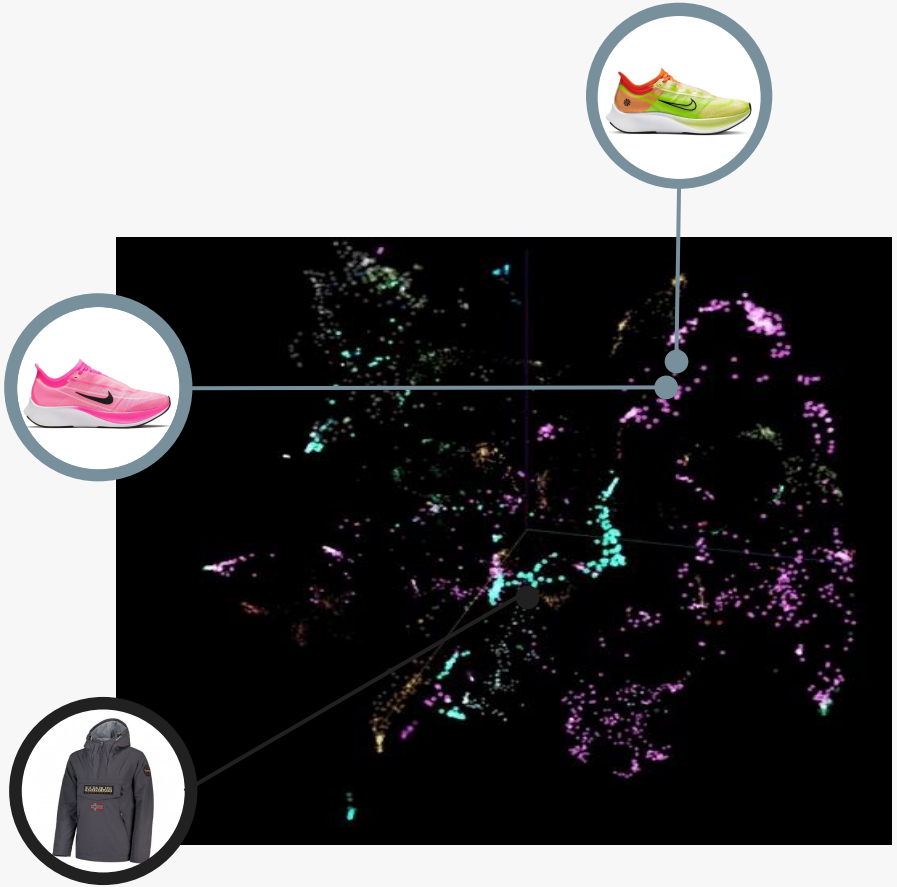
# Prod2Vec

- When a prod2vec model is trained, similar products appear to be closer in the embedding space.
- The 3D T-SNE projection shows how products related to the same sport activity cluster together (for a catalog in the sport apparel vertical).

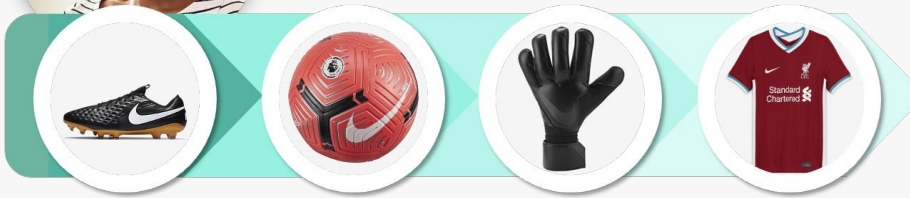


# Prod2Vec

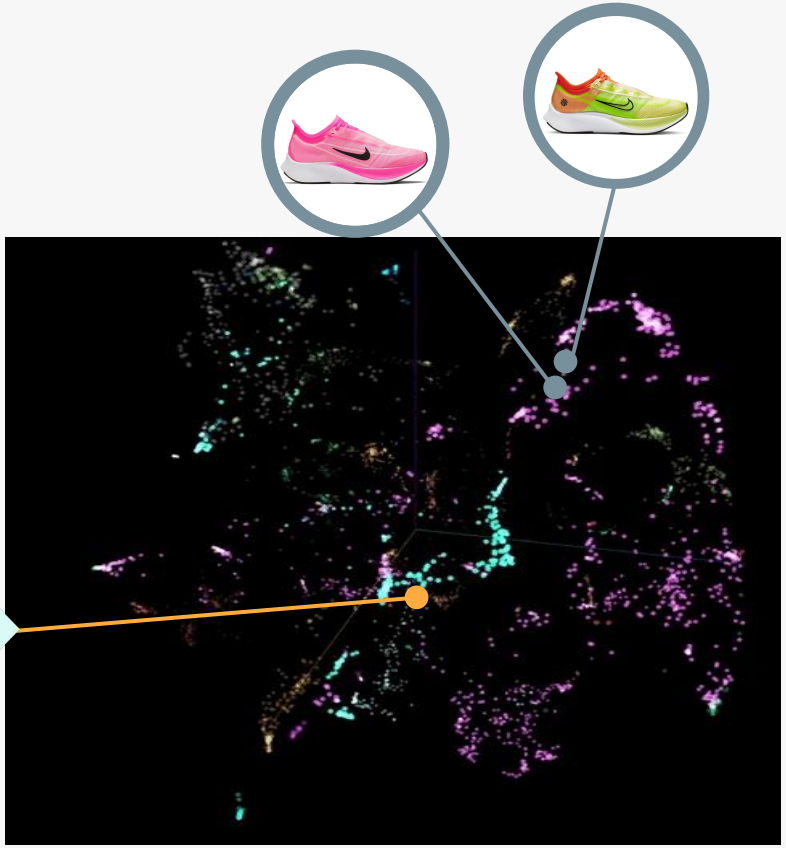
- Embedding space: products which are “semantically” similar will end up close in the embedding space.
- Prediction: when shopper is visiting item A, retrieve embedding for A and do KNN to retrieve similar items.
- Training: skip-gram (same as word2vec).



# Session-base RS



Prediction: when shopper is visiting item A, retrieve embedding for A and do KNN to retrieve similar items



# Deep Dive into Language Models

# Tokenization

- Tokenizers map text to ids tied to a model's vocabulary.
- Strategies: bytes, characters, words, subwords.
- Subword tokenizers (BPE, WordPiece, SentencePiece) balance vocabulary size with open-vocabulary coverage.
- Practical tip: inspect token splits and counts for your real inputs.

lowercase and CAPITALIZATION

🤗

show_tokens False None elif == >= else: two tabs:" " Three tabs: " "

12.0*50=600

# Representing words as vectors: bag of words

**Corpus**

A collection of text documents

↓

**Tokenize**

Divide the text into smaller units called tokens, usually words or phrases

↓

**Count word frequencies**

Create a vocabulary of unique words and count the number of times each word appears in each document.

↓

**Encode the data**

Encoding the text data as numerical values by creating a vector for each document, with each element of the vector representing the frequency count of a particular word in the document

*Corpus:*

The dog is happy. The child makes the dog happy. The dog makes the child happy

↓

*Tokenization:*

D1: [The] [dog] [is] [happy]
D2: [The] [child] [makes] [the] [dog] [happy]
D3: [The] [dog] [makes] [the] [child] [happy]

↓



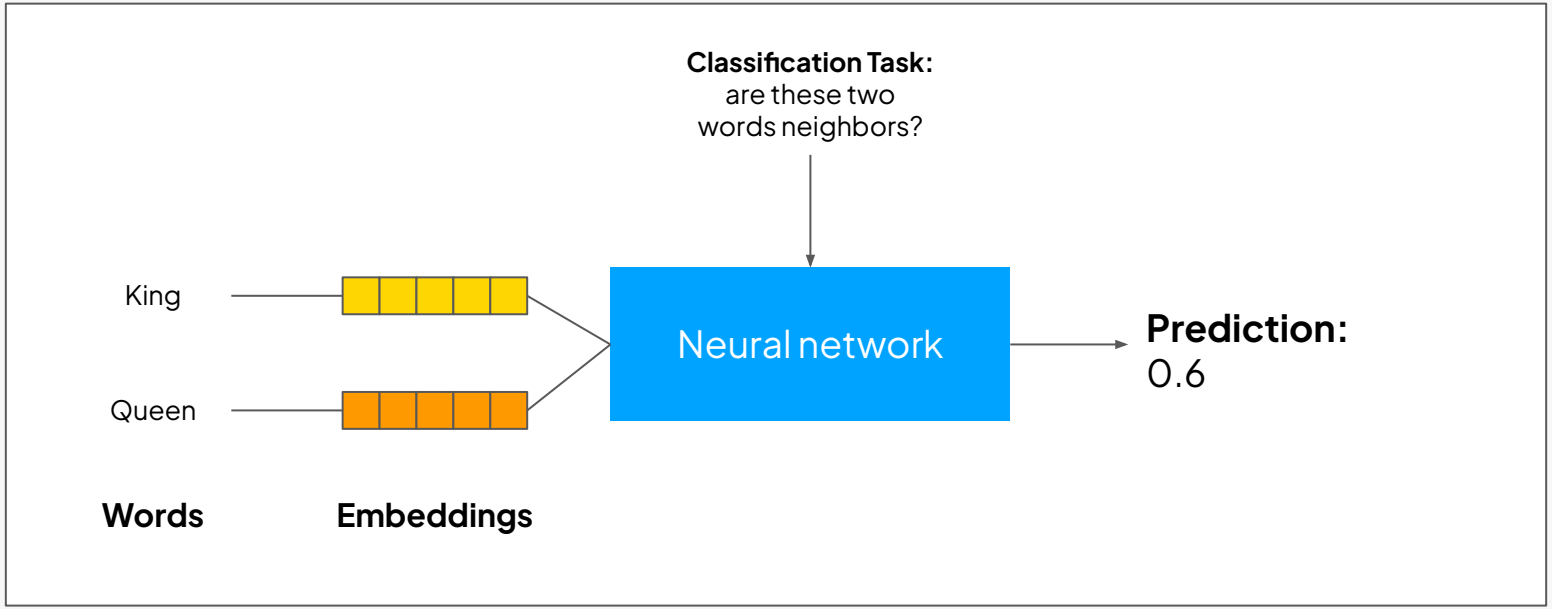
| Documents | Counting word frequencies |
|---|---|
| D1 | the: 1, dog: 1, is: 1, happy: 1 |
| D2 | the: 2, dog: 1, makes: 1, child: 1, happy: 1 |
| D3 | the: 2, child: 1, makes: 1, dog: 1, happy: 1 |

↓

| Encode | child | dog | happy | is | makes | the | BoW Vector representations |
|---|---|---|---|---|---|---|---|
| D1 | 0 | 1 | 1 | 1 | 0 | 1 | [0,1,1,1,0,1] |
| D2 | 1 | 1 | 1 | 0 | 1 | 2 | [1,1,1,0,1,2] |
| D3 | 1 | 1 | 1 | 0 | 1 | 2 | [1,1,1,0,1,2] |

# Dense vector embeddings

- Meaning and context: dense vectors capture distributional similarity (“you shall know a word by the company it keeps”).



# Dense vector embeddings

- Geometry encodes meaning: Use cosine similarity as a proxy for semantic relatedness.

$$\cos\theta = \frac{u \cdot v}{\|u\| \, \|v\|}$$

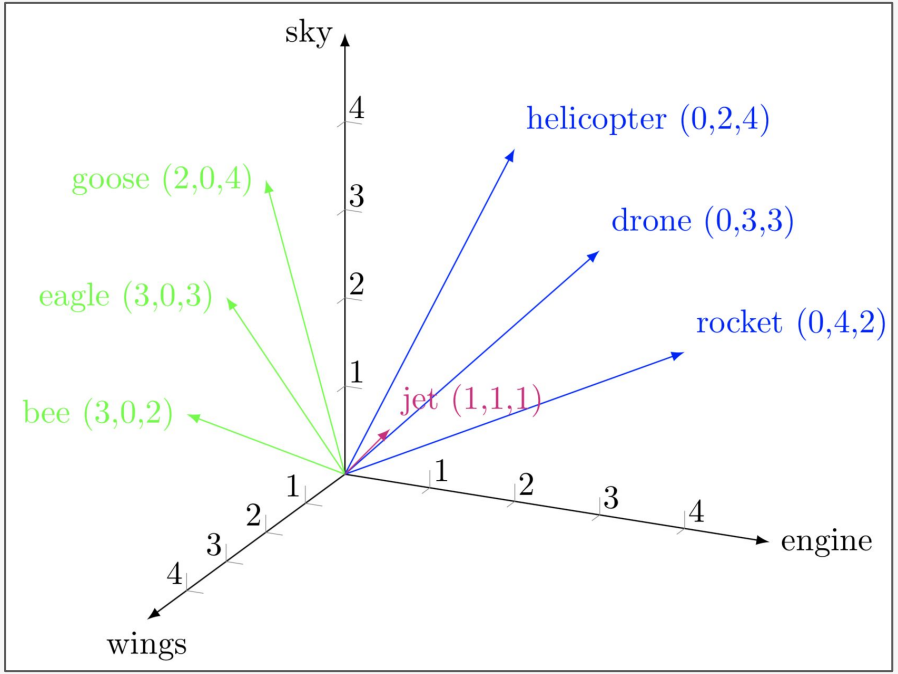
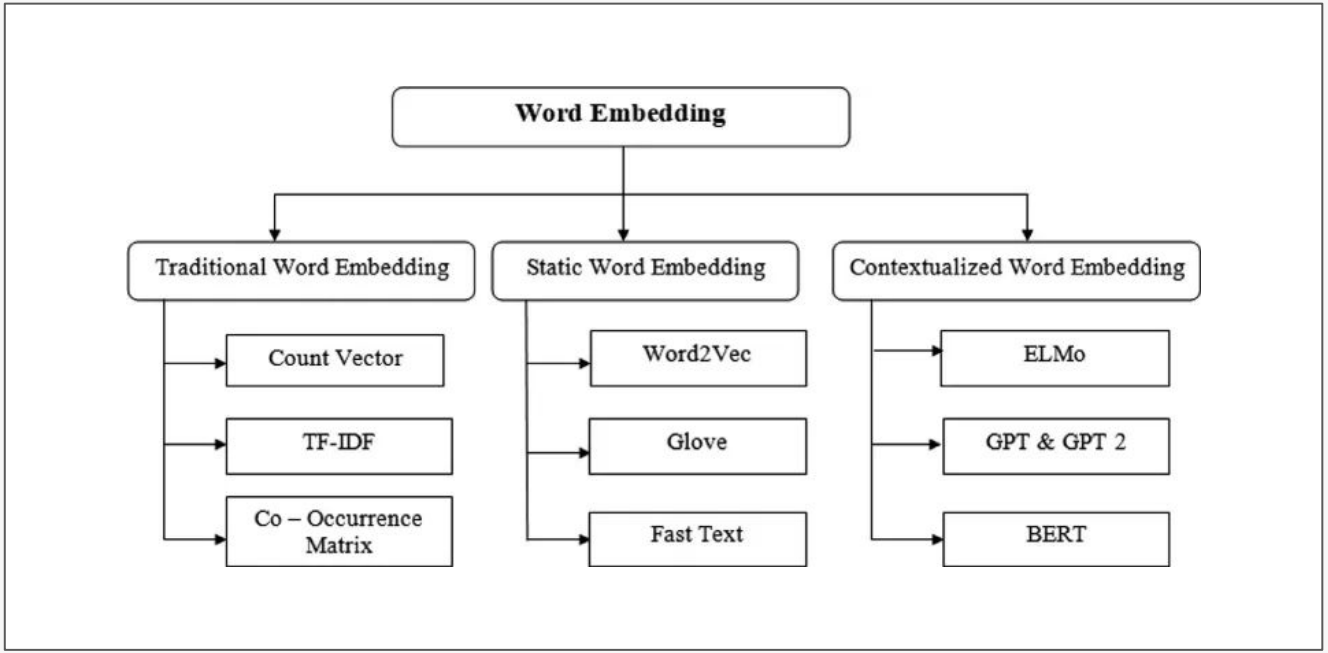



Image from Corpus Linguistics and Statistics with R
http://www.springer.com/gp/book/9783319645704

# Types of embeddings

- **Token embeddings:** one vector per vocabulary token at the model input.
- **Contextual word embeddings:** the vector for a word depends on its sentence.
- **Sentence or document embeddings:** a single vector for the whole text.
- Why it matters: generalizes beyond exact word overlap, enables semantic search, clustering, retrieval for LLM apps.

# Types of word embeddings



# Training static embeddings (Word2Vec)

- Training static embeddings (Word2Vec)
- Predict context from a center word (skip-gram) or vice versa (CBOW).
- Learn two embedding tables; score with dot products.
- Optimize logistic loss with negative sampling and backprop.
- Result: one vector per word type. Great for intuition, limited on polysemy.

# Embeddings in practice (inside an LLM)

- You download: tokenizer, embeddings, transformer blocks, output head.
- Embeddings are necessary but not sufficient. Most capability lives in attention and MLP layers.
- LM head is often tied to the token embedding matrix.
- For search: use purpose-built sentence/document encoders; for generation: use the full LLM.

# LLMs, Sequence Modeling, and Post-Training

# Architectures to predict sequences

**n-grams → learn from counts.** Models estimate probability from counts; simple and fast, but suffer from sparse statistics as n grows, and can't model long contexts or generalize beyond observed n-grams.

**RNNs (2014) → learn from sequences.** Recurrent nets replace fixed n with a hidden state, letting the model in principle use unbounded context. Practically, they struggle with vanishing/exploding gradients and long-range dependencies.

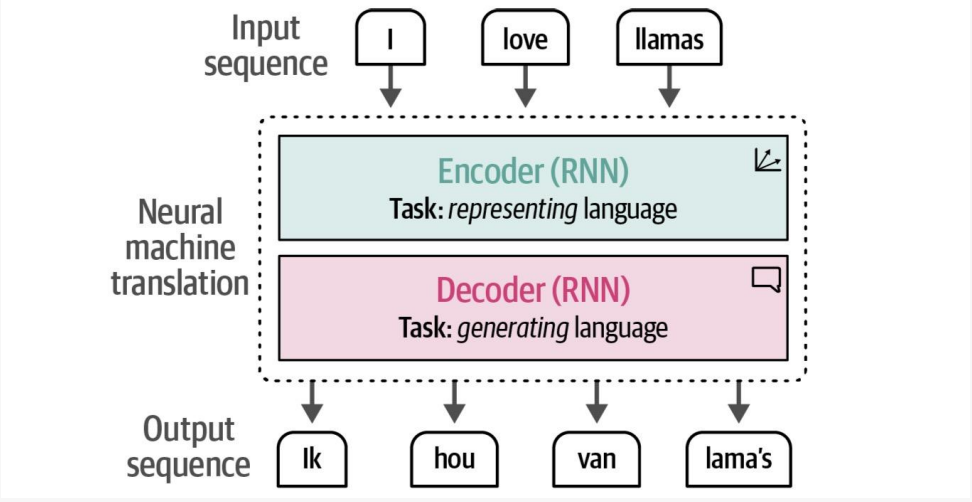
**LSTMs → mitigate vanishing gradients.** LSTMs add gates and persistent cell states, making RNN training more stable and improving long-range patterns. They power early state-of-the-art seq2seq systems, but are still sequential.

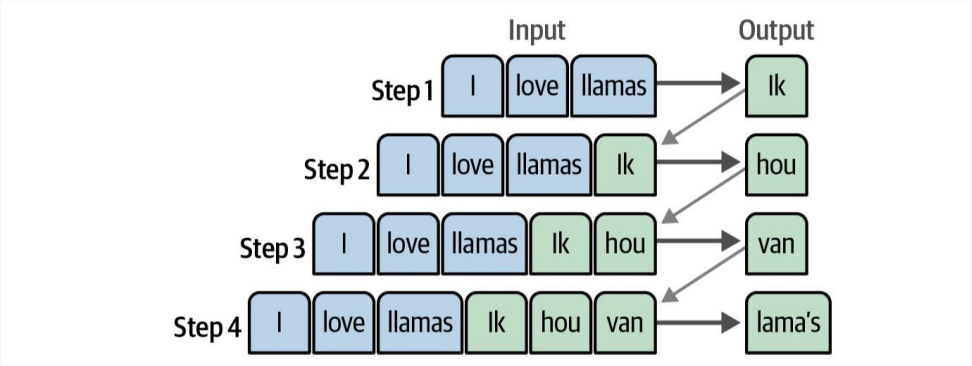
**Attention (2015) → break the "thought-vector" bottleneck.** Vanilla seq2seq compresses an input into one vector; attention lets the decoder soft-search over all encoder states each step.

**Transformers (2017) → scale with self-attention.** Self-attention replaces recurrence with direct token-to-token interactions; multi-head attention captures multiple relationships; positional encodings restore order; and the architecture trains in parallel over tokens.

# Encoder–decoder

- **Encoder**: reads the input sequence step by step and compresses it into a hidden representation (context vector).
- **Decoder**: takes this representation and generates the output sequence, one token at a time.
- **Training**: decoder is guided with the correct previous word (teacher forcing).
- **Inference**: decoder relies on its own predictions (autoregressive loop).




*Images from Jay Alammar, Maarten Grootendorst*

# Context-thought bottleneck

- **Seq2seq compresses the whole input into one fixed-size vector:** it works on short, simple inputs BUT drops details on long, complex sentences.
- **Symptoms**: wrong word order, missing entities, brittle agreement across clauses.
- **Motivation for attention**: let the decoder look back at all encoder states, not just one summary.

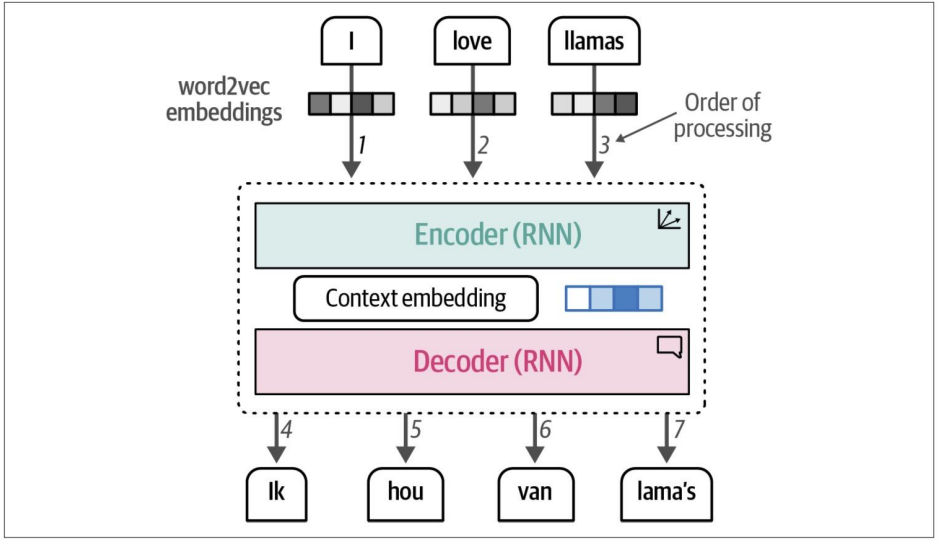
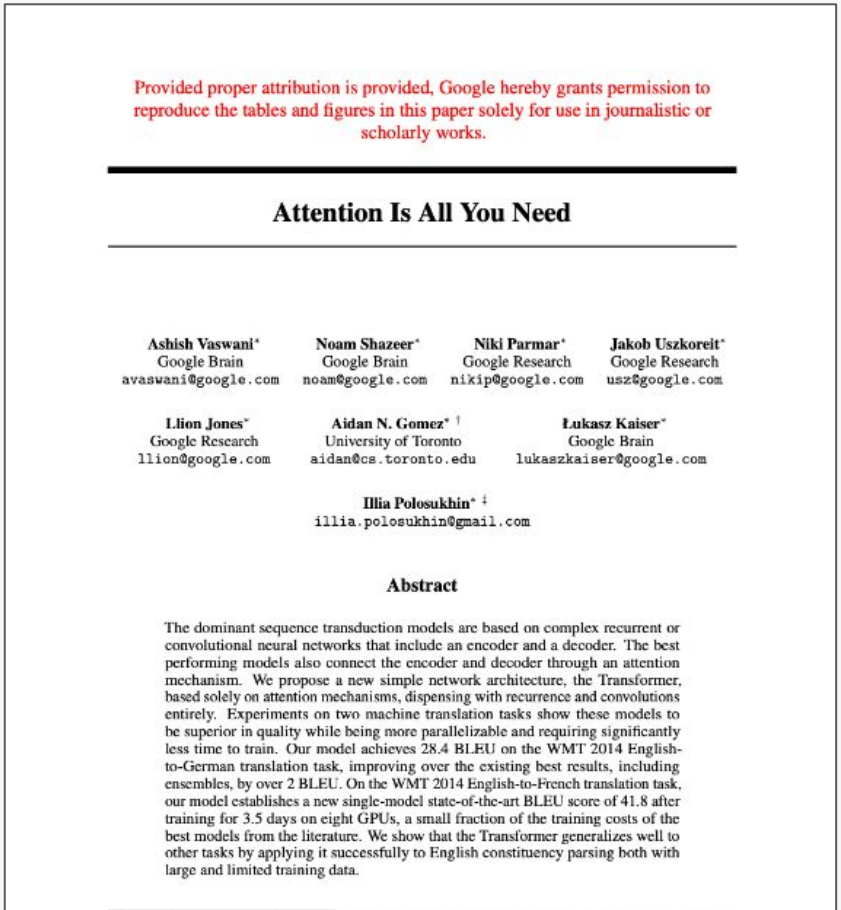



*Figure 1-13. Using word2vec embeddings, a context embedding is generated that represents the entire sequence.*

*Images from Jay Alammar, Maarten Grootendorst*

# Attention is all you need

- In 2017, attention replaces recurrence in sequence models.
- **Key idea:** tokens attend to tokens directly (no more RNN loop).
- Attention unlocks parallel training, longer-range dependencies, clean scaling.
- This architecture enabled **BERT, GPT, T5** and the foundation model era.





**Attention Is All You Need**

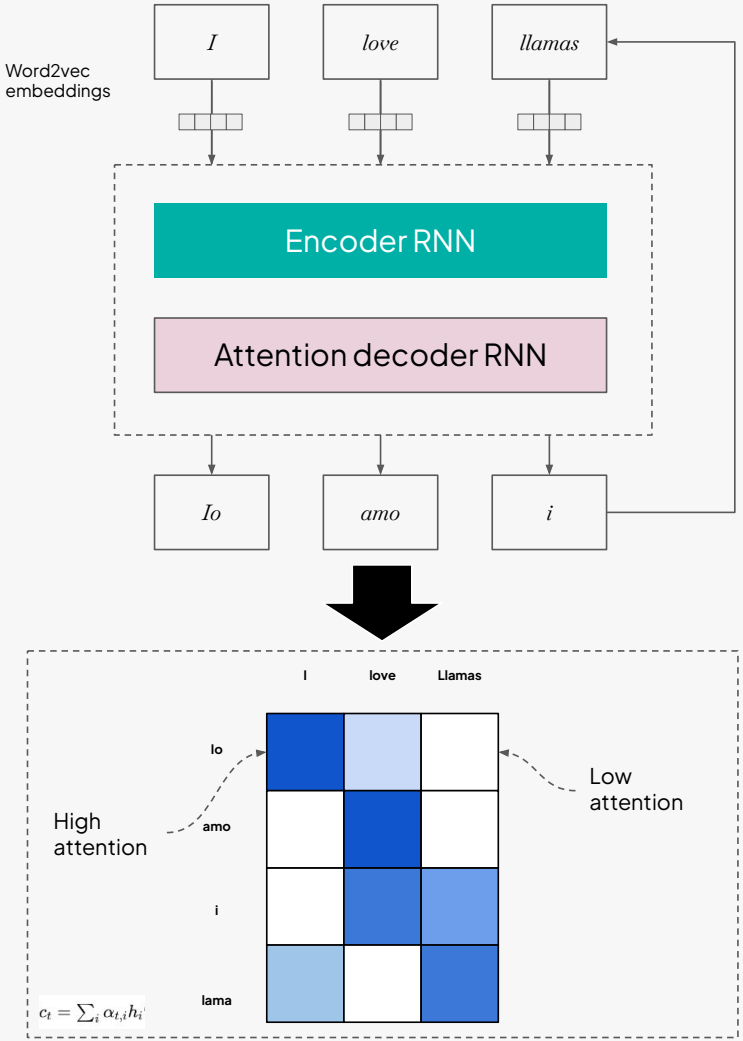
**Ashish Vaswani**[*] Google Brain avaswani@google.com
**Noam Shazeer**[*] Google Brain noam@google.com
**Niki Parmar**[*] Google Research nikip@google.com
**Jakob Uszkoreit**[*] Google Research usz@google.com
**Llion Jones**[*] Google Research llion@google.com
**Aidan N. Gomez**[*] [†] University of Toronto aidan@cs.toronto.edu
**Łukasz Kaiser**[*] Google Brain lukaszkaiser@google.com
**Illia Polosukhin**[*] [‡] illia.polosukhin@gmail.com

**Abstract**

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

# Attention in seq2seq

- Attention fixes the problem of one single “thought vector” dropping details on long inputs.
- The decoder now computes a weighted look-back over all encoder states.
- Benefit: soft alignments, better long sentences, partial interpretability.

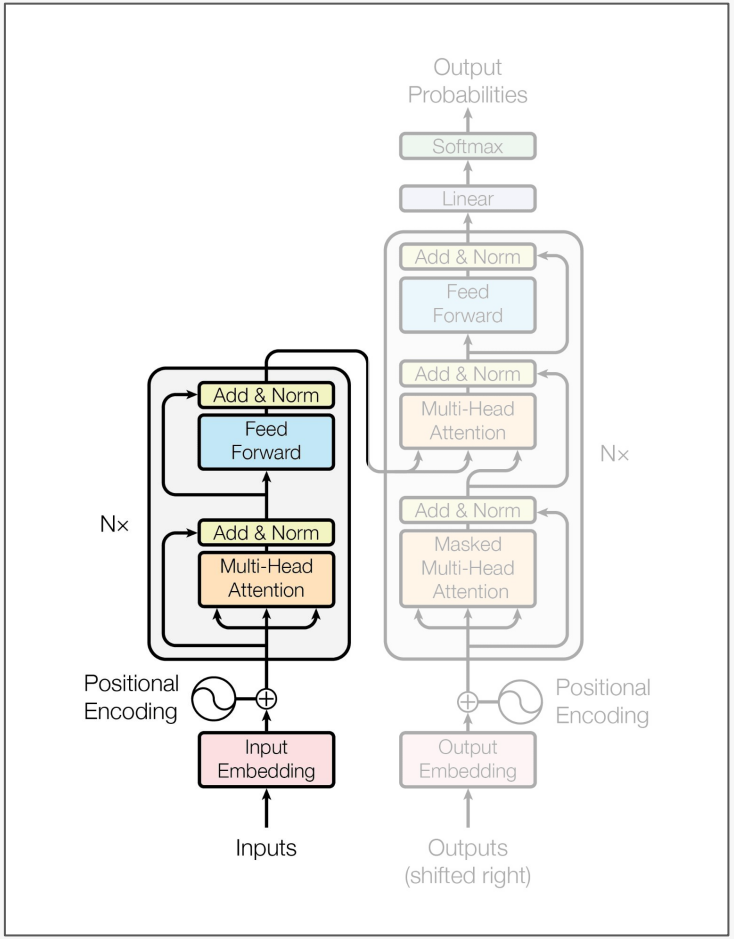


# Transformer encoder

- Builds global, order-aware token states (not one summary), fixing long-range context.
- Each word looks at the whole sentence so nothing important is forgotten.

---

- Input embeddings + positional info.
- Multi-head self-attention (bidirectional).
- Feed-forward network (per token).
- Residual connections and LayerNorm.
- **Output:** contextual token representations.

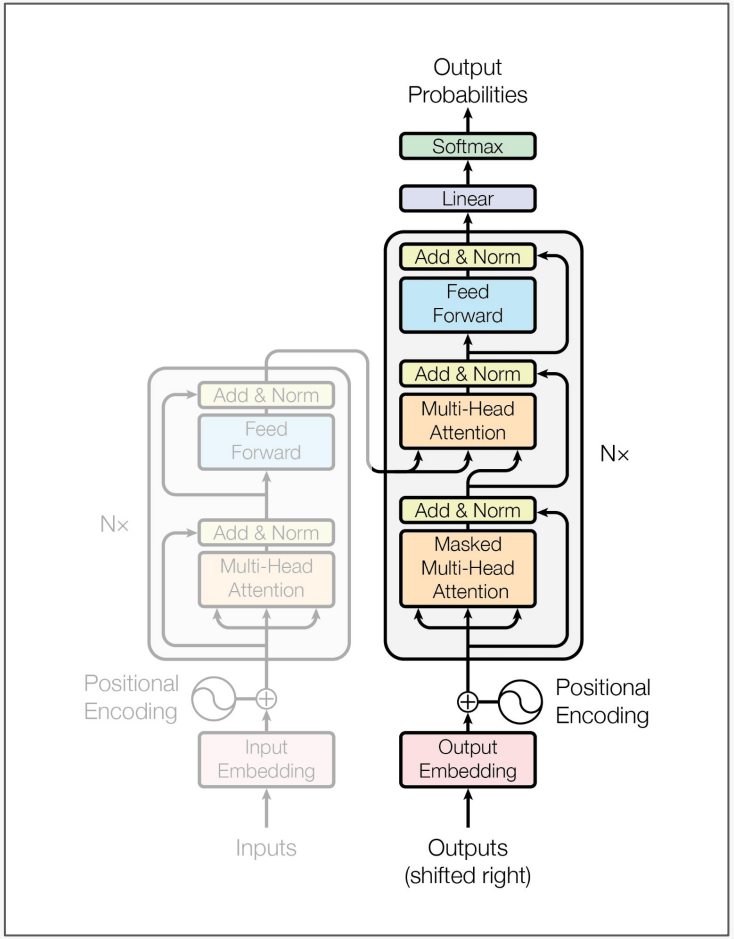


# Transformer decoder

- Generates causally while retrieving source info via cross-attention for faithful outputs.
- It writes the next word and checks the original input so the answer stays on topic.

---

- Masked self-attention over generated tokens.
- Cross-attention to encoder outputs.
- Feed-forward network.
- Residuals and LayerNorm.
- **Output:** next-token logits for autoregressive generation.



# Putting it all together

- Encoder builds contextual representations; decoder generates with masked self-attention and cross-attention, enabling parallel training and strong long-range modeling.

---

- Encoder stack produces source representations.
- Decoder stack generates targets using masked self-attention + cross-attention.
- Parallelizable training, strong long-range modeling.
- Families: encoder-only (BERT), decoder-only (GPT), encoder-decoder (T5).