Large Language Models in Data Science

Week 1: Concepts, Architecture, Motivation

Sebastian Mueller
Aix-Marseille Université
2025-2026

Session Overview

Lecture (1.5h)

1. Why study LLMs now?
2. From words to tokens
3. Transformer architecture
4. Training and inference
5. Capabilities, limits, ecosystem

Lab (1.5h)

- Tokenize and embed real text
- Inspect attention weights with HuggingFace
- Call OpenAI GPT models and estimate cost
- Bonus: visualize attention with BERTviz

Motivation: Data Science is Changing

- Analysts increasingly converse with their tools instead of writing boilerplate code.
- Reports, dashboards, and even experiments are drafted by generative models.
- Modern data workflows combine structured data with unstructured documents, code, and conversations.
- Understanding LLMs helps us design safer, more efficient assistants rather than treating them as black boxes.

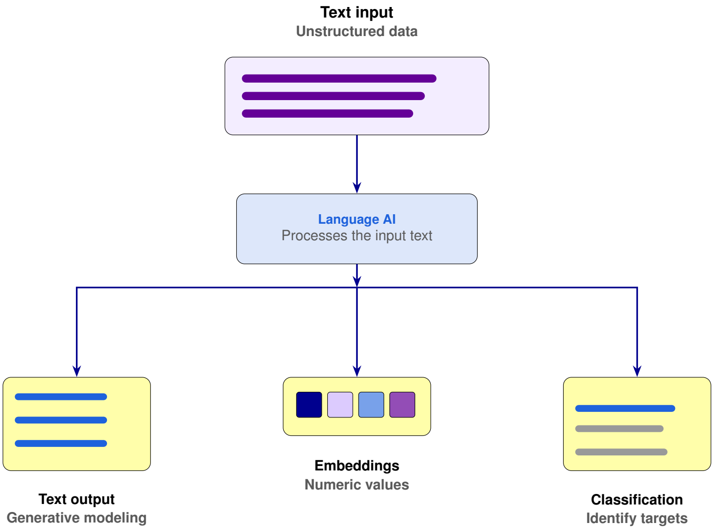
What is a Large Language Model?

- **Definition**: A neural network trained on massive text corpora to model the probability of token sequences.
- Given tokens $x_1, \ldots, x_{t-1}$, the model estimates

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

- During inference the model samples one token at a time, feeding each prediction back as context.
- LLMs power question answering, summarization, code generation, and dialogue systems across industry.

LLM Capabilities



Motivation: Machines Need Numbers

- Neural networks operate on numbers, not raw strings.
- Preprocessing must map text to numeric inputs while preserving meaning and structure.
- Tokenization, vocabularies, and embeddings create that bridge.

Core Concepts

1. Token

Unit of text fed to the model. Depending on the tokenizer it may be a word, subword, character, or byte.

2. Tokenization

Procedure that converts raw text into a sequence of tokens.

3. Vocabulary

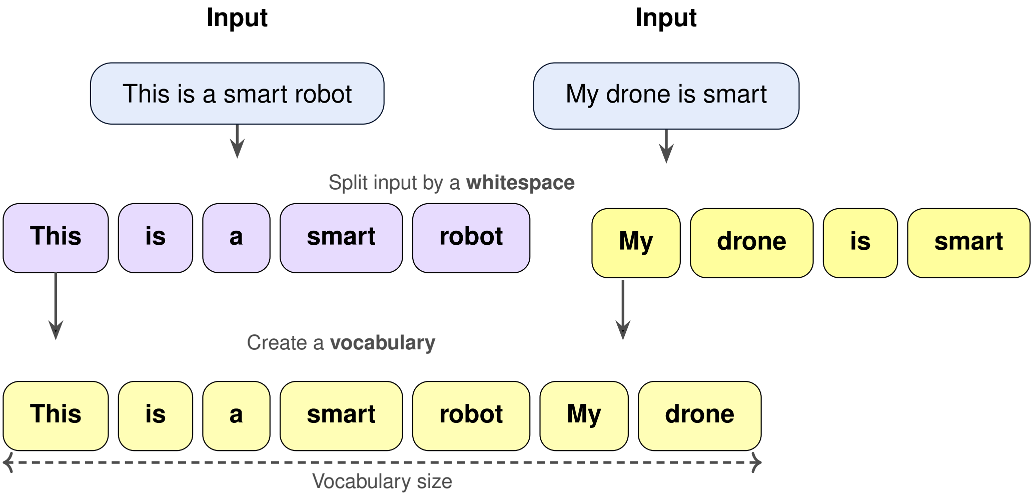
Finite set of tokens the model knows. Typical vocabularies contain $V \approx 50{,}000$ tokens.

Example: “Prediction”

Raw text	`Prediction`
Tokens	`["Pred", "iction"]`
Ids	`[4792, 1526]`

Subword splits let the model handle rare words by composing common pieces.

Bag of words - Whitespace Tokenization



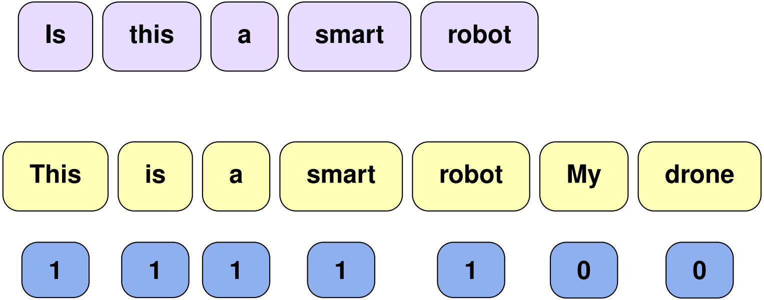
Vector representation

This | is | a | smart | robot

This	is	a	smart	robot	My	drone
1	1	1	1	1	0	0

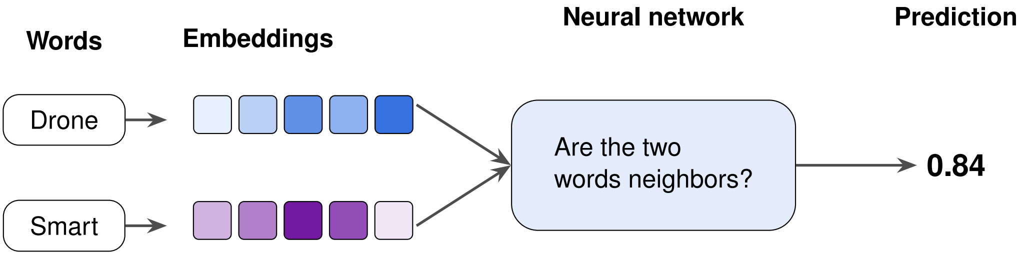
Bag of words is created counting the number of occurrences of each word in the vocabulary.

Vector representation



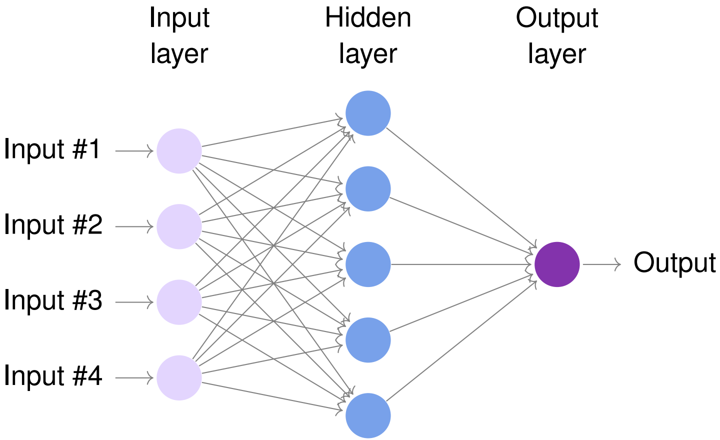
Bag of words is created counting the number of occurrences of each word in the vocabulary.

Dense Vector Embeddings - word2vec



The neural network is trained to predict if two words are neighbors (appear in similar contexts).

Neural Networks



The neural network learns the weights of the connections to optimize its predictions.

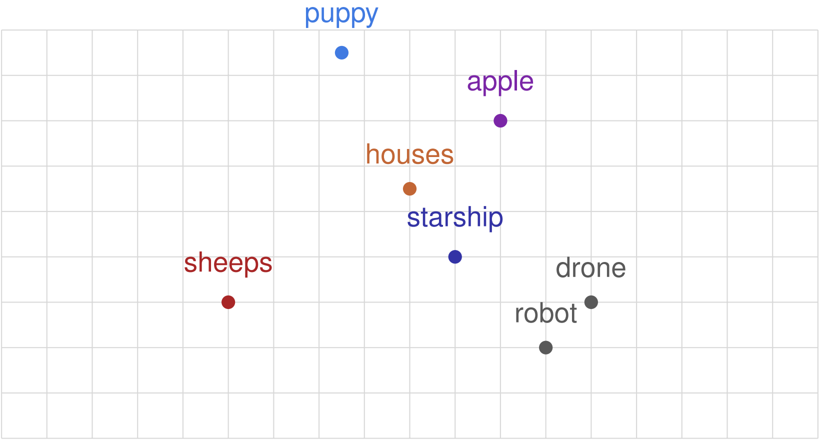
Embedding Dimensions

	sheeps	puppy	houses	apple	robot
animal	.90	.94	-0.56	-0.71	.01
machine	-0.11	.71	-0.32	-0.15	.90
human	.19	.36	.31	.29	-.87
⋮	⋮	⋮	⋮	⋮	⋮
plural	.94	-0.82	.94	-0.51	-0.11
fruit	-0.51	-0.91	-0.5	.90	-0.51

Number of dimensions

The values in the embedding vectors represent different **latent** features of words. The dimensions do not have an explicit meaning.

2D Embedding Visualization



Zipf’s Law in Language

- Word frequency in natural language follows a power law distribution (Zipf’s Law).
- A few words are extremely common, while many words are rare.
- Example: In English, ”the” is the most common word, while ”quokka” is rare.
- Implication: Tokenizers must balance vocabulary size to cover common words while handling rare words via subword units.

Embeddings: Turning Tokens into Vectors

- **Embedding matrix** $E \in \mathbb{R}^{V \times d}$ maps token ids to d-dimensional vectors.
- For token id x_i, lookup yields $\mathbf{e}_i = E[x_i] \in \mathbb{R}^d$.
- Embeddings capture semantic similarity: similar words live near each other in vector space.
- These vectors are the starting point for all subsequent Transformer computations.

Recaps of Embeddings

Why embeddings?

- Language models cannot process raw text: we must map tokens to vectors in $\mathbb{R}^d$.
- Embeddings capture semantic similarity in geometry: similar words close vectors.

Embedding domains:

- Words, subwords, sentences, documents
- Also used in other modalities: images, audio, graphs

Similarity in Embedding Space

- **Cosine similarity** measures angle between vectors:

$$\text{cosine\_sim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}.$$

- Values range from -1 (opposite) to 1 (same direction).
- Used to find similar words, sentences, or documents.
- Nearest neighhor reveals semantic and functional relationships.
- Beware domain shift; neighbors change with training data

Static vs Contextual Embeddings

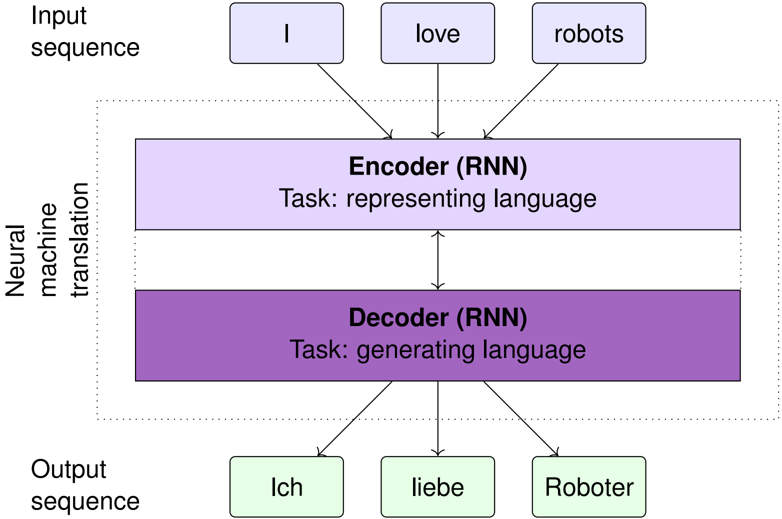
Static embeddings *(e.g., word2vec, GloVe, FastText)*

- Each word has a single fixed vector, independent of context
- Cannot handle polysemy (e.g., `bank` = river vs finance)
- Cannot handle out-of-vocabulary words.

Contextual embeddings *(e.g., ELMo, BERT, GPT)*

- Embeddings depend on surrounding context (sentence, paragraph)
- Can disambiguate meaning dynamically
- Use subword tokenization robust to unknown words

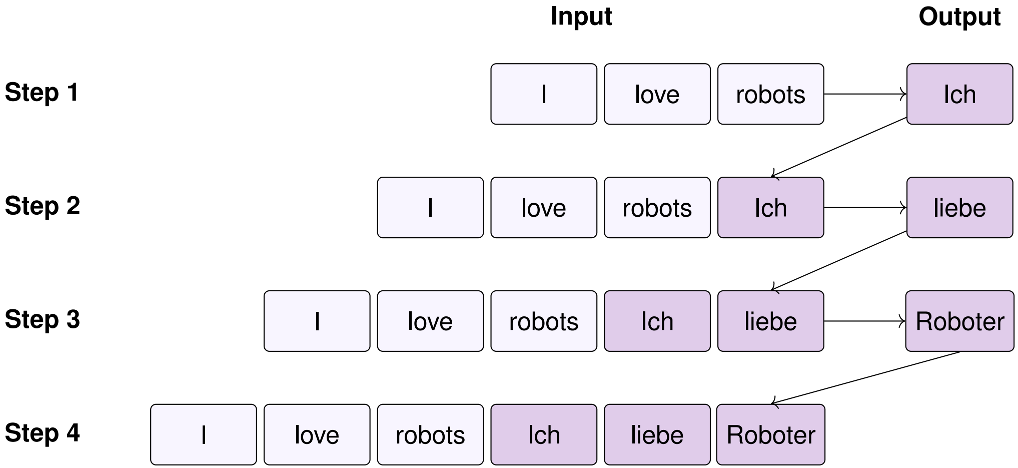
Neural Machine Translation with Encoder-Decoder RNN



Motivation: Sequential Data

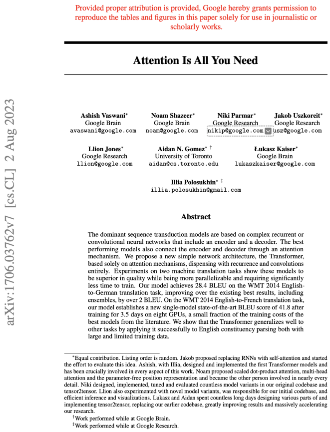
- Language is inherently sequential: word order matters.
- Meaning of words depends on context (previous words).
- RNNs (Elman, 1990; Hochreiter & Schmidhuber, 1997) process tokens one at a time, maintaining a hidden state that summarizes past context.
- Variants like LSTMs and GRUs address vanishing gradients, enabling longer-range dependencies.
- RNNs were the dominant architecture for NLP before Transformers.

Autoregressive Nature of Decoder RNN



Motivation: Beyond Recurrent Networks

- Recurrent Neural Networks process tokens sequentially, limiting parallelism and context length.
- Transformers (Vaswani et al., 2017) introduced self-attention, enabling long-range dependencies and scalable training, parallelizations on GPUs.
- Result: state-of-the-art across NLP tasks, vision-language models, and code generation.



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

arXiv:1706.03762v7 [cs.CL] 2 Aug 2023

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.

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

Transformer Architecture Overview

- Key innovation: **self-attention** mechanism allows each token to attend to all others in the sequence.
- Enables parallel processing of tokens, unlike RNNs.
- Composed of stacked layers of attention and feed-forward networks, with residual connections and normalization.

Self-Attention: What and Why

- **What**: Each token forms *queries*, *keys*, and *values*: $Q = HW_Q$, $K = HW_K$, $V = HW_V$.
- **Computation**:

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

- **Why softmax?** Turns similarity scores into a probability distribution to weight values.
- **Why scale by $\sqrt{d_k}$?** Prevents saturation of softmax for large d_k, stabilizing training.

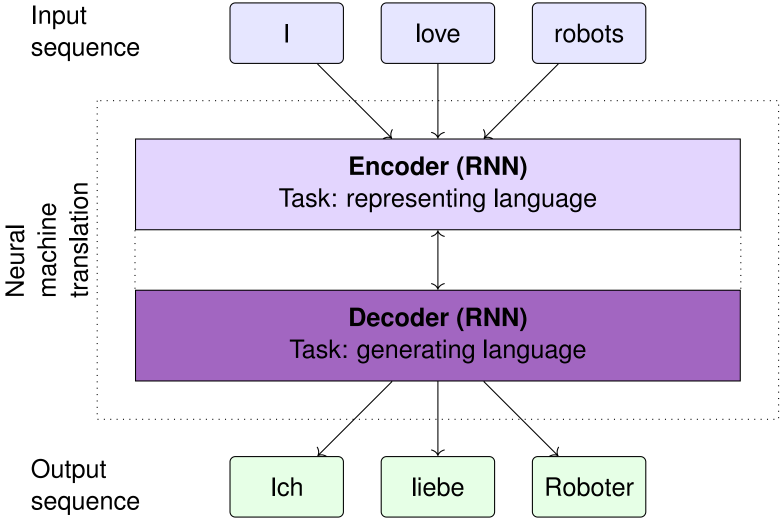
Tiny intuition

'The cat sat' $\rightarrow$ stronger weights to nearby/related tokens (e.g., 'cat' when processing 'sat').

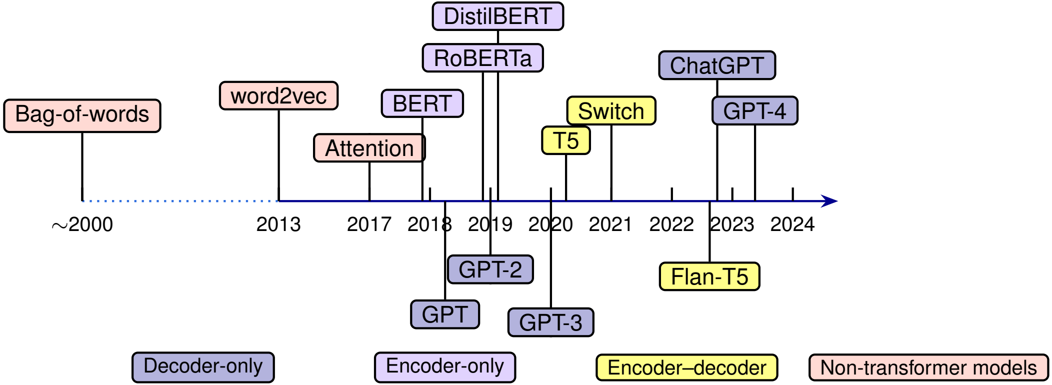
The Transformer at a Glance

- **Encoder–Decoder (original paper)**: encoder builds contextual representations; decoder attends to both past outputs (self-attention) and encoder outputs (cross-attention).
- **Encoder-only (BERT family) - representation models**: uses only bidirectional self-attention; effective for understanding tasks.
- **Decoder-only (GPT family) - generative models**: uses only causal/masked self-attention with a mask to prevent peeking ahead; simpler and efficient for generation.
- Common building blocks across both: embeddings + positional info, scaled dot-product attention, multi-head projections, residual connections, layer normalization, feed-forward networks.

Neural Machine Translation with Encoder-Decoder RNN



LLM Model Timeline (2000–2024)



Further Reading on Transformers

- Highly recommended: The Illustrated Transformer (Jay Alammar) for step-by-step visuals.
- Optional deeper dive: Attention in Transformers (Josh Starmer) for basic understanding of attention.
- Explore attention live: poloclub.github.io/transformer-explainer

Inference and Decoding Strategies

- Generation proceeds token by token; decoding choices shape behavior.
- **Temperature**: scales logits to control randomness.
- **Top-*k*** and **top-*p* (nucleus)** sampling: restrict candidate tokens to most probable set.
- **Max tokens**: upper bound on generated length; context window includes prompt + output.
- Engineering considerations: caching key/value tensors, speculative decoding, batching requests.

What LLMs Do Well

- Few-shot learning: adapt to new tasks using examples in the prompt.
- Chain-of-thought prompting: articulate intermediate reasoning steps.
- Code generation and refactoring for Python, SQL, and beyond.
- Summarization, translation, explanation across domains.
- Data-wrangling assistance: regex creation, feature brainstorming, doc parsing.

Limits, Risks, Practices

- **Hallucinations**: confident but fabricated answers without grounding.
- **Inconsistency**: outputs vary with phrasing, temperature, and randomness.
- **Context window limits**: finite memory (e.g., 8k–200k tokens) affects long documents.
- **Bias and toxicity**: inherited from training data, requiring audits and guardrails.
- **Cost and latency**: API tokens and GPU serving are not free; optimize prompts and batching.
- **No persistent memory**: model forgets state across sessions unless you build storage.
- **Data privacy**: be cautious with sensitive info; follow provider policies.
- **Best practices**: clear prompts, few-shot examples, temperature tuning, output validation.
- **Human-in-the-loop**: always review critical outputs; LLMs assist, not replace, human judgment.

Where to Get Models (Hosted APIs)

Major providers (managed, pay-per-use; fast start)

Provider	Example Models (2025)	Access
OpenAI	o4-mini, o3, GPT-4o	Hosted API (reasoning, vision)
Google DeepMind	Gemini 2.5 Pro / Flash	Gemini API (free tier + paid)
Anthropic	Claude Sonnet 4.5	Claude API / Console
xAI	Grok-4, Grok-3	API (OpenAI-compatible)

When to choose

- Minimal ops, latest frontier models, strong tooling & SLAs.
- Cons: recurring cost, data governance/contracts, rate limits.

Where to Get Models (Open Weights & Hubs)

Open models (self-host or use hosted endpoints)

Provider	Example Models (2025)	Access
Meta	Llama 3.1, Llama 4 (Scout/Maverick)	Open weights (license)
Mistral	Mistral Large/Medium/Small, Pixtral	Open weights & Mistral API
Microsoft	Phi-4, Phi-4-reasoning	Open weights (HF)
Hugging Face Hub	Falcon, Mixtral, Phi, etc.	Hub + Inference API/Endpoints

When to choose

- Control, on-prem/privacy, cost efficiency at scale.
- Cons: you manage serving, scaling, evals, safety layers.

Calling an API

```python
from openai import OpenAI
client = OpenAI()

resp = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {"role": "system", "content": "You are a data assistant."},
        {"role": "user", "content": "Tell me a joke about matrix inversion."}
    ],
    temperature=0.6,
    max_tokens=120,
)
print(resp.choices[0].message.content)
```

Token Economics

- Providers charge per 1,000 tokens (roughly 750 words). Both input (prompt) and output tokens count.
- Example: if input = 500 tokens and output = 200 tokens at \$0.002 per 1K tokens, cost $\approx$ \$0.0014.
- Track usage: log prompts, responses, and costs for transparency and budgeting.

Key Takeaways

- LLMs model $P(x_t \mid x_{<t})$ and rely on tokenization, embeddings, and Transformers to operate.
- Self-attention + positional encodings enable parallel processing and long-range context.
- Training spans massive pretraining, then alignment (SFT, RLHF, DPO) for safety and usefulness.
- Real-world use demands awareness of strengths, limitations, providers, and cost models.
- Lab will reinforce these concepts through hands-on tokenization and API exercises.

Further Reading

Concept	Reference / Why it matters
LLM overview	How Transformer LLMs Work (deeplearning.ai) concise intro to components.
Transformer basics	The Illustrated Transformer (Jay Alammar) visual intuition for attention.
Tokenization	OpenAI `tiktoken`; HF Tokenizers counts, costs, subwords.
Attention math	Attention Is All You Need (Vaswani et al., 2017) original paper.
APIs (OpenAI)	OpenAI Platform Docs endpoints, auth, safety.
APIs (Gemini)	Gemini API Docs free tier + multimodal.
APIs (Claude)	Anthropic Claude Docs reasoning, tool use.
APIs (Grok)	xAI Grok API OpenAI-compatible endpoints.