

Transformers

How language models work

Mikhail Mikhasenko · Lecture 4A

Generative models

A **classifier** learns $p(y | x)$: which label for this input?

A **generative model** learns the distribution of the data itself, $p(x)$, and can **sample** new examples from it.

With a condition, it samples $p(x | \text{prompt})$:

- an image given a description
- a text given the start of a text

Language models are generative models of text.



“A painted picture of a crocodile in the house drinking tea and waiting for a turtle to visit”

A language model predicts the next token

Split text into tokens $x_1, \dots, x_T$. By the chain rule of probability,

$$p(x_1, \dots, x_T) = \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1}).$$

So one task is enough: **given the text so far, give a probability for every possible next token.**

Generate by sampling a token, appending it, and repeating.

GPT-2 (124 M parameters, 2019):

text so far	next token
The American flag is red, white, and	blue 0.93
When Mary and John went to the store, John gave a drink to	Mary 0.68
The capital of France is	the 0.08, now 0.05, ... Paris 0.03

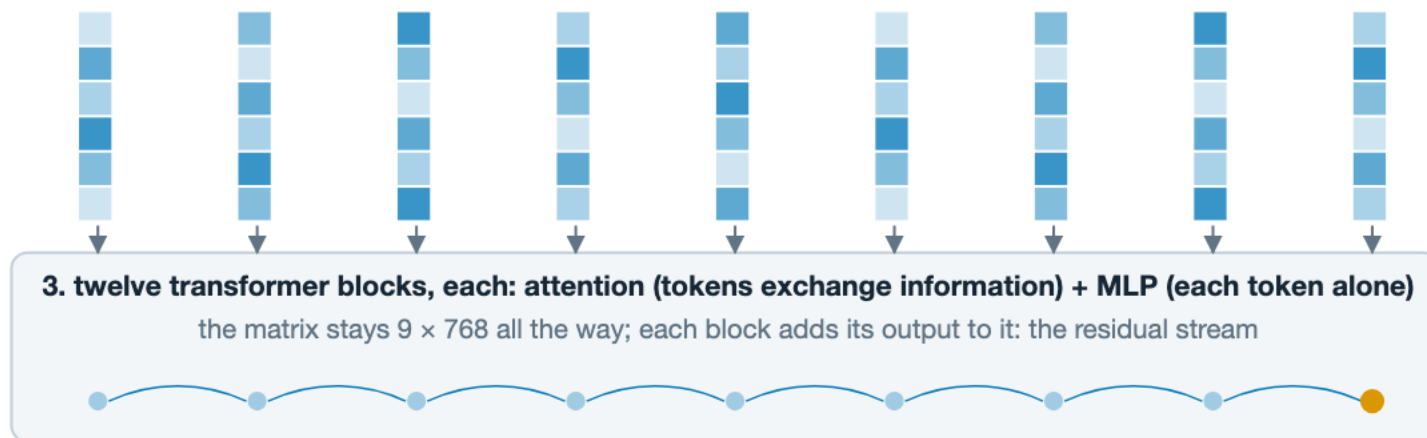
Llama 3.2 (1.2 B, 2024) gives Paris 0.39.

The whole machine on one slide

1. text → tokens



2. token ids → embedding vectors (one row of W_E each, 768 numbers)



4. last row × unembedding matrix → 50 257 logits → softmax → probabilities



5. sample a token, append it to the input, and run again: autoregressive generation

Why tokens?

Unit	Vocabulary	Length of a text	Problem
characters	~100 (bytes: 256)	long: one step per letter	the model must learn spelling before meaning
words	millions	short	new words, typos, names and other languages are unknown
subword tokens	50 k – 200 k	~4 characters per token in English	a compromise, learned from data

Frequent words become single tokens (`·the`, `·electron`); rare words are split into pieces (`·Bo` `ch` `um`).

Byte-pair encoding

Training the tokenizer

1. Start with single symbols (GPT-2: the 256 possible bytes).
2. Count all adjacent pairs in a large text.
3. Merge the most frequent pair into a new symbol.
4. Repeat until the vocabulary has the desired size.

GPT-2: 256 bytes + 50 000 merges + 1 end-of-text token = **50 257**.

Using it

Split new text into bytes, then apply the learned merges in the order they were learned.

Starting from bytes, nothing is ever “unknown”: any text, any language, any emoji can be encoded.

The tokenizer is trained **once, before** the network, and is not changed by gradient descent.

Byte-pair encoding, merge by merge

Byte-pair encoding, merge by merge

· marks a space red = symbol never seen in training

Start from single characters. Repeatedly find the most frequent adjacent pair in the training text and merge it into a new token.

Merges  40

+1 merge

Play

Reset

Training text

English physics text ▾

Training text

The electron and the proton attract each other. The energy of the electron in the atom is quantized, and the transition between two energy levels emits a photon. The photon energy equals the difference of the two energies. In the detector, the photon produces a shower, and the shower energy is measured by the calorimeter. The calorimeter measures the energy of electrons and photons; the tracker measures the momentum of charged particles. Together, the tracker and the calorimeter reconstruct the particles that were produced in the collision. The reconstruction of the collision is the first step of the analysis. In the analysis, we select the events, we measure the distributions, and we compare the measured distributions with the predicted distributions. The prediction

Tokenize new text

The transformer predicts the next token.
Thermodynamics of transformers!

The · t r a n s f o r m e r · p r e d i c t s · t h e · n e x t
· t o k e n · . The r m o d y n a m i c s · o f · t r a n s f
o r m e r s !

Learned merges

40	· + s → ·s	4x
39	· + is → ·is	4x
38	t + on → ton	5x
37	e + d → ed	5x
36	·energ + y → ·energy	5x
35	·T + he → ·The	5x
34	· + w → ·w	5x
33	· + d → ·d	5x
32	t + r → tr	6x
31	o + r → or	6x
30	h + o → ho	6x
29	a + l → al	6x
28	·o + f → ·of	6x
27	·measu + re → ·measure	6x
26	·meas + u → ·measu	6x
25	·mea + s → ·meas	6x
24	·me + a → ·mea	6x
23	·ener + g → ·energ	6x

Vocabulary

30 chars + 40 merges = 70

New text

72 chars → 53 tokens

Training text

845 chars → 458 tokens

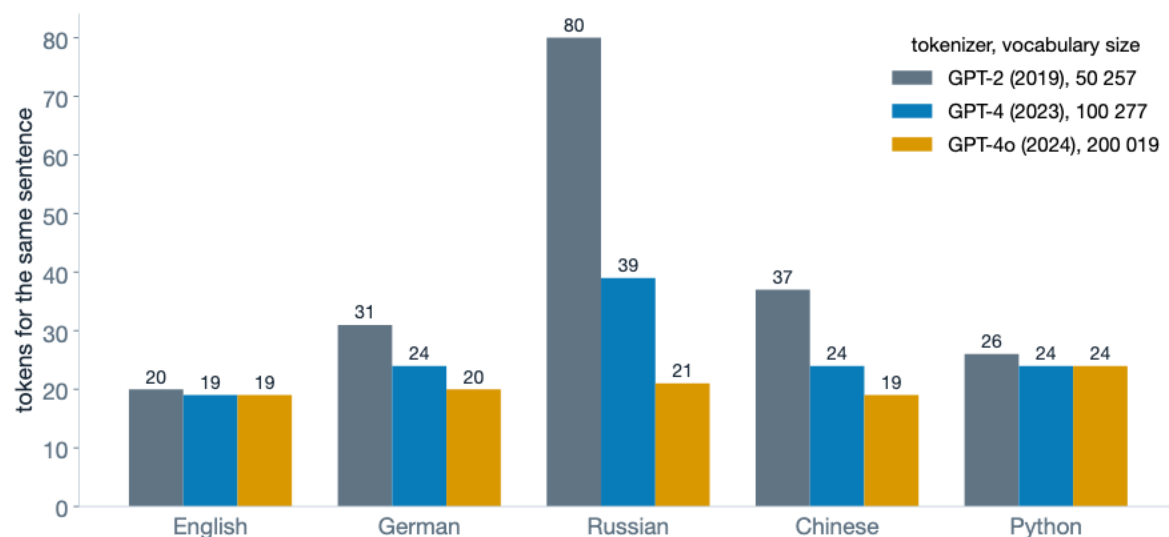
Chars per token (training)

1.84

Unknown symbols

2

Real tokenizers



text

GPT-2 tokens

strawberry

st raw berry

·Bochum

·Bo ch um

·Energieerhaltungssatz

9 pieces

·12345678

·123 45 678

·SolidGoldMagikarp

1 token

The model never sees letters: counting the r's in *strawberry* is hard.

Embedding: a token becomes a vector

Each token id i selects one row of the **embedding matrix** W_E :

$$\mathbf{e}_i = W_E^\top \mathbf{1}_i \in \mathbb{R}^d, \quad W_E \in \mathbb{R}^{V \times d}.$$

A one-hot vector times a matrix: a lookup table, trained by backpropagation like any other weight.

GPT-2: $V = 50\,257$, $d = 768$, so W_E has **38.6 M** of the 124 M parameters.

Training moves tokens that are used in similar contexts to similar vectors:

- nearest to **Monday**: Tuesday, Thursday, Wednesday, Friday
- nearest to **electron**: electrons, photon, neutron
- **king - man + woman** $\approx$ **queen**
- **Paris - France + Germany** $\approx$ **Berlin**

The same idea as AlexNet's fc7 embedding in Lecture 3B.

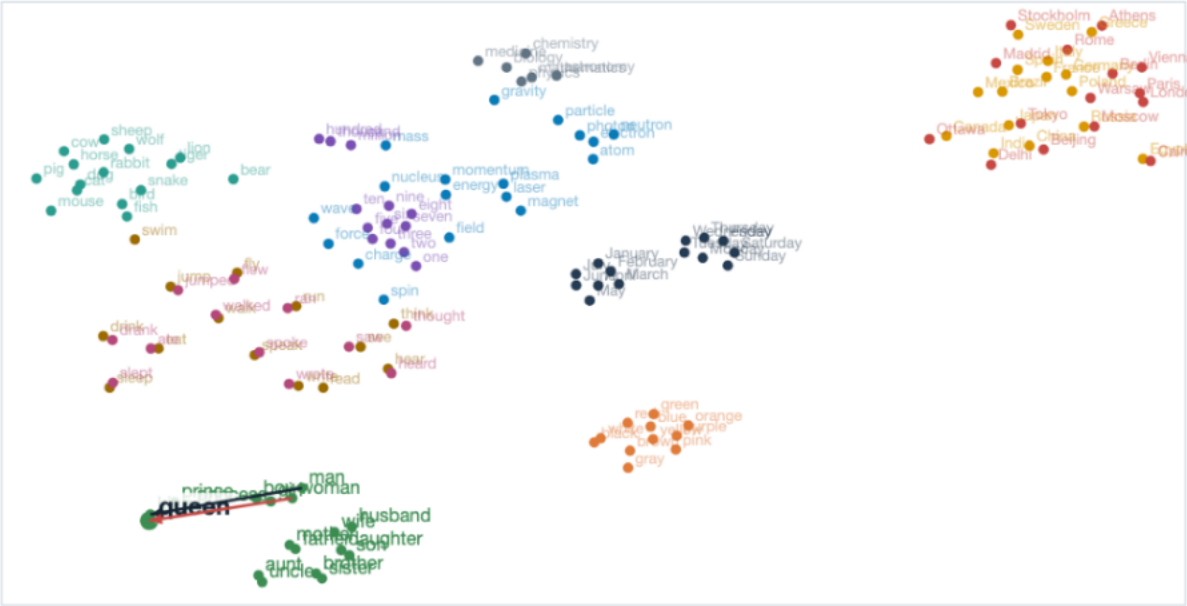
Token embeddings of GPT-2

Token embeddings of GPT-2

each word is a vector of 768 learned numbers · click a word

The map is a 2D t-SNE projection of the vectors. Vector arithmetic below is done in all 768 dimensions.

Analogy king – man + king – man + woman Compute



Vector arithmetic Closest words (cosine)
king – man + woman queen 0.71 princess 0.60 girl 0.49 wife 0.49

Where is each token?

Attention (next slides) compares tokens by content only. Shuffle the input and each token gets the same result: “dog bites man” = “man bites dog”.

So add information about the **position** t to each token vector:

$$\mathbf{x}_t = \mathbf{e}_{\text{token}(t)} + \mathbf{p}_t$$

model

position encoding

Transformer (2017)

fixed sines and cosines,
 $p_{t,2k} = \sin(t/10000^{2k/d})$

GPT-2 (2019)

learned vector per position,
up to 1024

Llama, most models since
2023

RoPE: rotate $\mathbf{q}$ and $\mathbf{k}$ by an
angle $\propto t$

RoPE makes the attention score depend on the **distance** between two tokens.

Why attention?

The meaning of a token depends on other tokens, sometimes far away:

- the **bank** of the river / the **bank** raised its rates
- The **electron** was accelerated in the field, and **it** ...
- The American **flag** is red, white, and ...

Each position must **collect information from the relevant other positions**, whose number and location change from text to text.

A dense layer has one fixed weight per input position: it cannot route information by content.

Recurrent networks (before 2017) passed a single state vector along the text: slow, one step at a time, and forgetful over long distances.

Attention: every token asks a question, every token offers an answer, and the weights are computed from the data.

Query, key, value

From each token vector $\mathbf{x}_i$ compute three vectors with learned matrices:

$$\mathbf{q}_i = W_Q \mathbf{x}_i, \quad \mathbf{k}_i = W_K \mathbf{x}_i, \quad \mathbf{v}_i = W_V \mathbf{x}_i$$

- **query**: what am I looking for?
- **key**: what do I contain?
- **value**: what do I pass on if selected?

Similarity is the dot product $\mathbf{q}_i \cdot \mathbf{k}_j$, the same similarity score as a CNN kernel on an image patch.

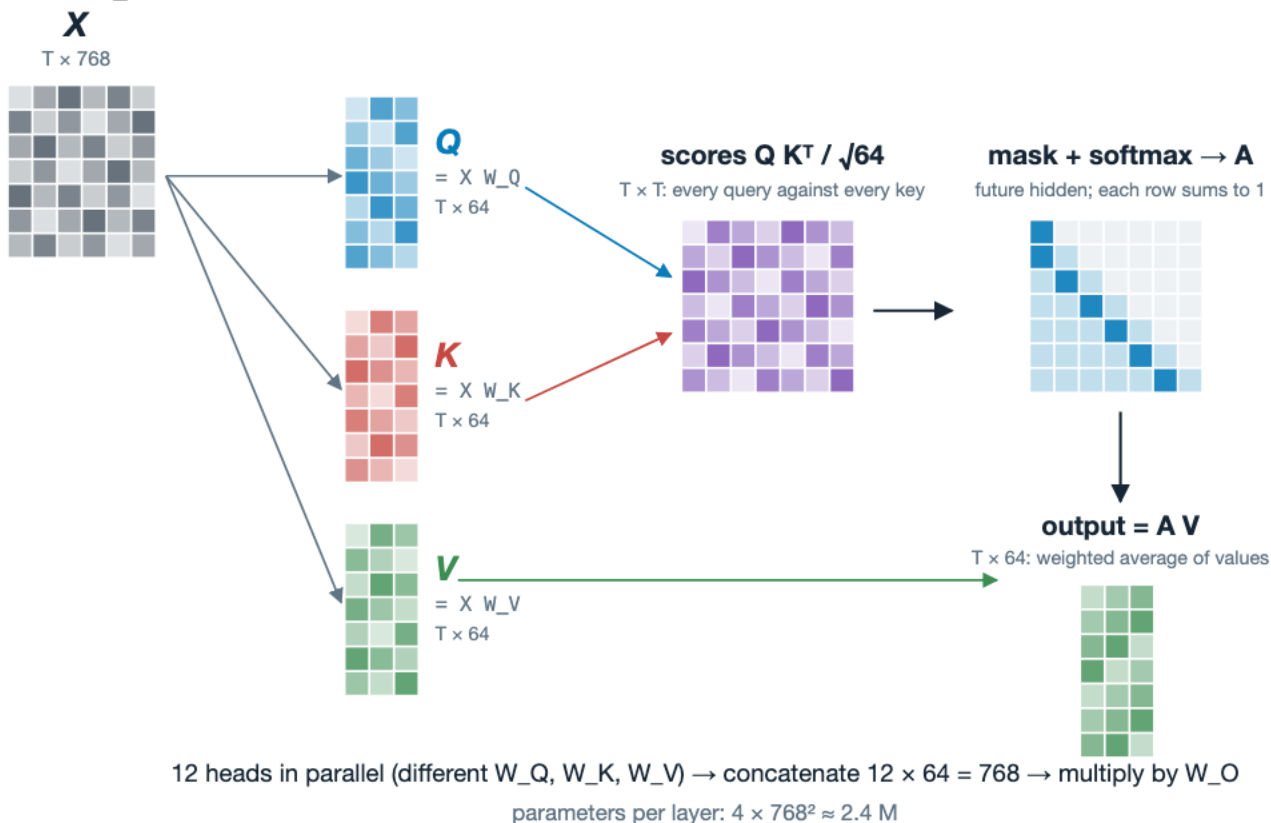
The output for token i is a weighted average of values:

$$\mathbf{o}_i = \sum_j a_{ij} \mathbf{v}_j, \quad a_{ij} = \frac{e^{\mathbf{q}_i \cdot \mathbf{k}_j / \sqrt{d_k}}}{\sum_{j'} e^{\mathbf{q}_i \cdot \mathbf{k}_{j'} / \sqrt{d_k}}}$$

Weights $a_{ij} \geq 0$, $\sum_j a_{ij} = 1$: a softmax over all tokens.

GPT-2: $\mathbf{x} \in \mathbb{R}^{768}$, $\mathbf{q}, \mathbf{k}, \mathbf{v} \in \mathbb{R}^{64}$ per head.

Scaled dot-product attention



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V, \quad M_{ij} = \begin{cases} 0 & j \leq i \\ -\infty & j > i \end{cases}$$

Dividing by $\sqrt{d_k}$: for random vectors $\text{Var}(\mathbf{q} \cdot \mathbf{k}) = d_k$, and large scores would saturate the softmax.

Attention by hand: which bank?

Attention by hand: which bank?

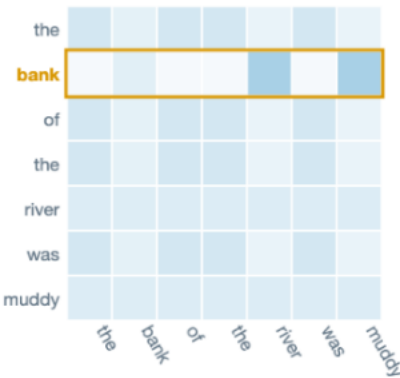
click a word to make it the query

Each word has a query q , a key k and a value v . The query of “bank” looks for topic words; their values say *river* or *money*.

Sentence the bank of the river was muddy Query strength $|q|$ 3.0 ☐ causal mask (decoder)

the bank of the river was muddy

Attention pattern A (rows sum to 1)

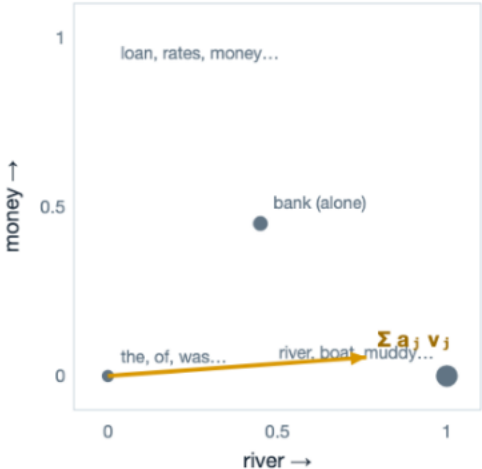


query (row) → key (column)

Query “bank”: $q = (3.0, 0.0)$

key	k	$q \cdot k / \sqrt{2}$	softmax	
the	(0,1)	0.00		0.04
bank	(0.5,0.3)	1.06		0.12
of	(0,1)	0.00		0.04
the	(0,1)	0.00		0.04
river	(1,0)	2.12		0.35
was	(0,1)	0.00		0.04
muddy	(1,0)	2.12		0.35

Values and output for the query



Query	Output $\sum a_j v_j$	Reading of “bank”	Largest weight
bank	(0.76, 0.06)	river bank	river (0.35)

Compare the river and the loan sentences. Then switch on the causal mask for “the bank of the river”: “bank” can no longer see “river”.

Multi-head attention

One attention pattern can express one kind of relation. So run h heads in parallel, each with its own $W_Q^{(k)}, W_K^{(k)}, W_V^{(k)}$ of size $d \times d/h$:

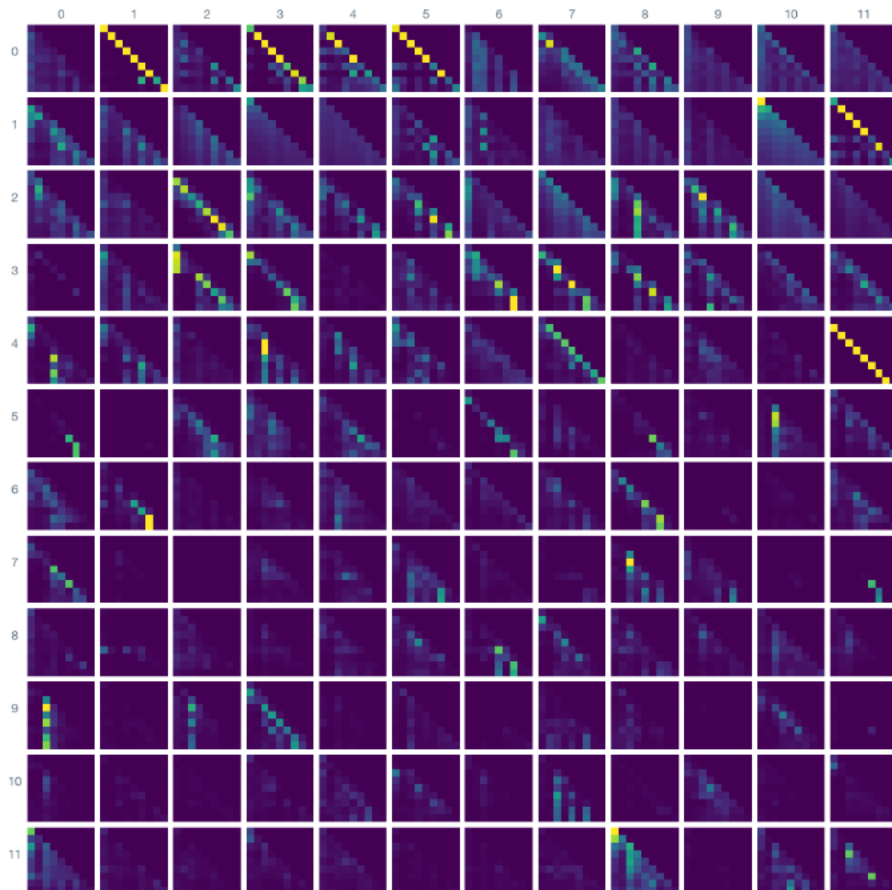
$$\text{MHA}(X) = [\mathbf{o}^{(1)} \dots \mathbf{o}^{(h)}] W_O$$

Heads specialize: previous word, subject of the verb, matching bracket, earlier copy of the current word...

	GPT-2 small	GPT-3	DeepSeek-R1
layers	12	96	61
heads per layer	12	96	128
width d	768	12 288	7 168
attention patterns	144	9 216	7 808

Parameters per attention layer: $4d^2$ (plus biases).

What GPT-2's heads look at



All 144 attention patterns of GPT-2 for “The American flag is red, white, and” (rows: layers 0–11, columns: heads 0–11).

Some roles are known:

- **L4 H11**: the previous token
- **L5 H5, L6 H9**: induction heads (next slide)
- **L9 H9, L9 H6**: move a name to the output (“John gave a drink to ... **Mary**”)

Most heads have no simple description.

Inside GPT-2: 144 attention heads

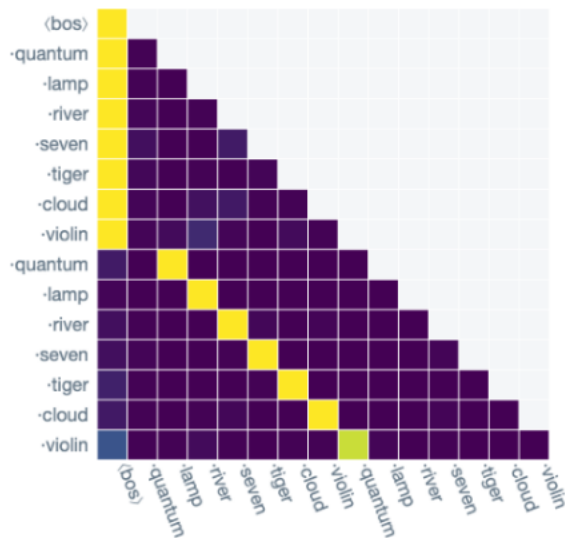
Inside GPT-2: 144 attention heads

12 layers × 12 heads · click a thumbnail

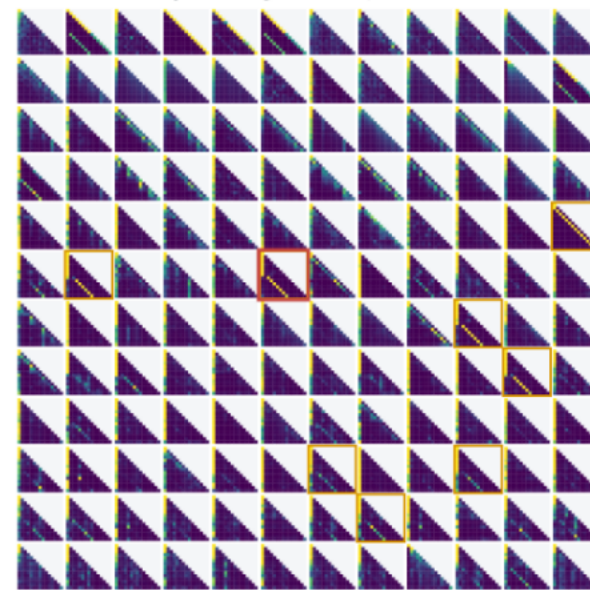
Real attention patterns of GPT-2 small. Row = the token doing the looking (query), column = the token looked at (key). <bos> is a start token.

Text Layer Head Known heads

Layer 5, head 5 — induction



All 144 heads (rows: layers 0–11, columns: heads 0–11)



gold frame = head with a known role

GPT-2 next token

" quantum" 31% "\n" 11% " Quantum" 4%

Last token "violin" attends to

quantum 0.73 <bos> 0.20 river 0.03

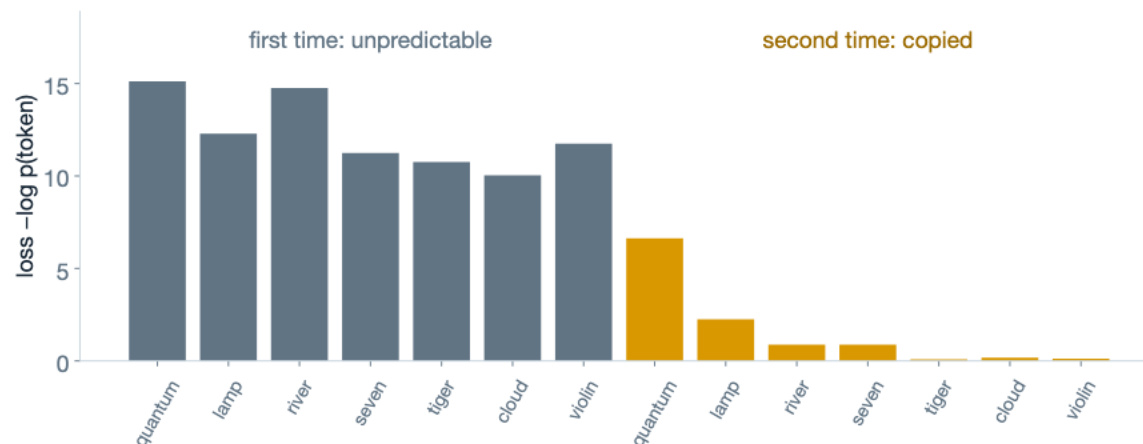
Induction heads: learning inside the prompt

A two-head circuit:

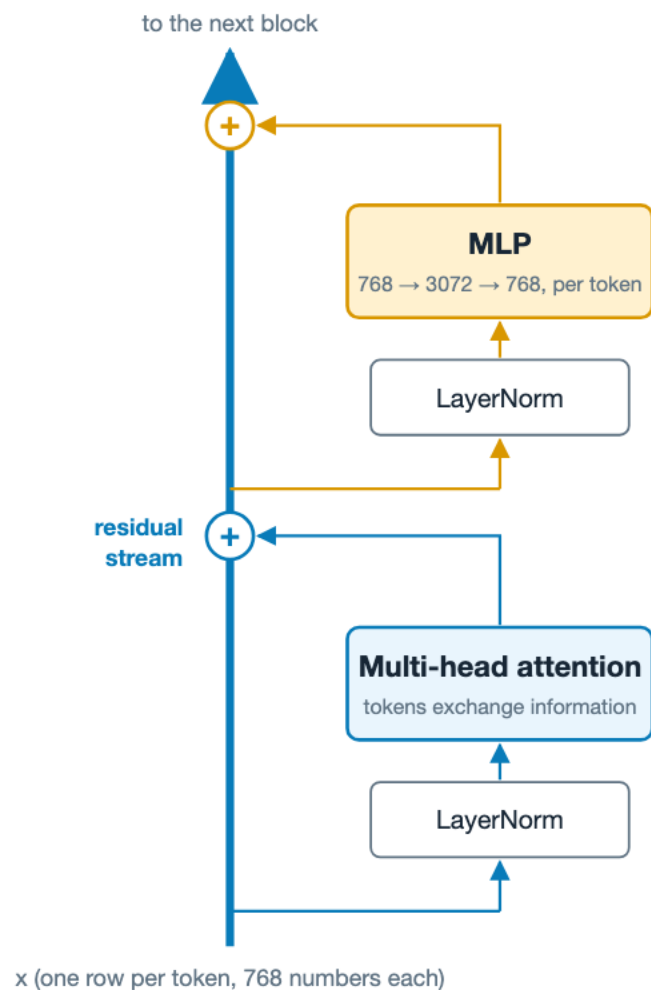
1. a **previous-token head** writes into each token “the word before me was A”;
2. an **induction head** at the current token B searches for earlier tokens whose previous word was B, and copies what followed.

Pattern: ... A B ... A $\rightarrow$ predict B.

Random words shown twice: GPT-2’s loss falls from **11.8** to **0.73** nats per token.



The transformer block



Attention: tokens exchange information.

MLP: each token processed on its own, $768 \rightarrow 3072 \rightarrow 768$ with GELU. Two thirds of the block's weights; much of the model's factual knowledge seems to live here.

LayerNorm: normalize each token vector,

$$\text{LN}(\mathbf{x}) = \gamma \odot \frac{\mathbf{x} - \mu}{\sigma} + \beta.$$

Residual connections: each block *adds* to $\mathbf{x}$, as in ResNet (Lecture 3B):
 $\mathbf{x} \leftarrow \mathbf{x} + \text{Attn}(\text{LN}(\mathbf{x})), \quad \mathbf{x} \leftarrow \mathbf{x} + \text{MLP}(\text{LN}(\mathbf{x}))$

GPT-2 small stacks 12 blocks, GPT-3 96.

The residual stream: watching a prediction form

	The	electron	has	a	negative
embed	theless	electron	lives	mathemat	negative
layer 1	ories	electron	been	few	negative
layer 2	latest	electron	been	lot	negative
layer 3	latest	electron	been	lot	pressure
layer 4	latest	electron	been	lot	spiral
layer 5	resa	electron	been	lot	spiral
layer 6	resa	microscope	been	lot	impact
layer 7	resa	microscope	been	tendency	impact
layer 8	resa	microscope	been	tendency	value
layer 9	resa	microscope	been	tendency	value
layer 10	resa	microscope	been	resonance	value
layer 11	following	microscope	been	magnetic	spin
layer 12	first	microscope	been	high	charge

input position (each cell: most likely NEXT token read out from the residual stream)

Each block adds a little to a running vector per token: the **residual stream**.

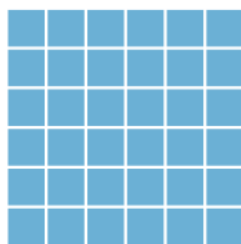
Logit lens: read the stream out early with the final unembedding.

After “negative”, GPT-2’s guess moves from *pressure* to *impact*, *value*, *spin* and finally **charge** (0.47) in the last layer.

Encoders and decoders

Encoder only

BERT (2018)



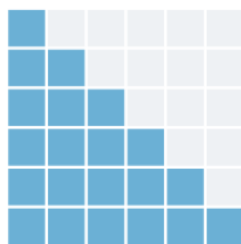
attention mask: row = query, column = key

sees the whole text in both directions

understanding: classify, embed, search

Decoder only

GPT, Llama, Claude (2018 →)



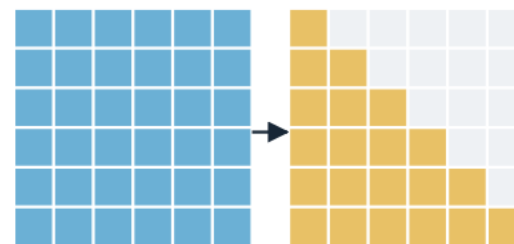
attention mask: row = query, column = key

each token sees only the past

generation: next token, chat, code

Encoder-decoder

original Transformer (2017), T5



encoder: full

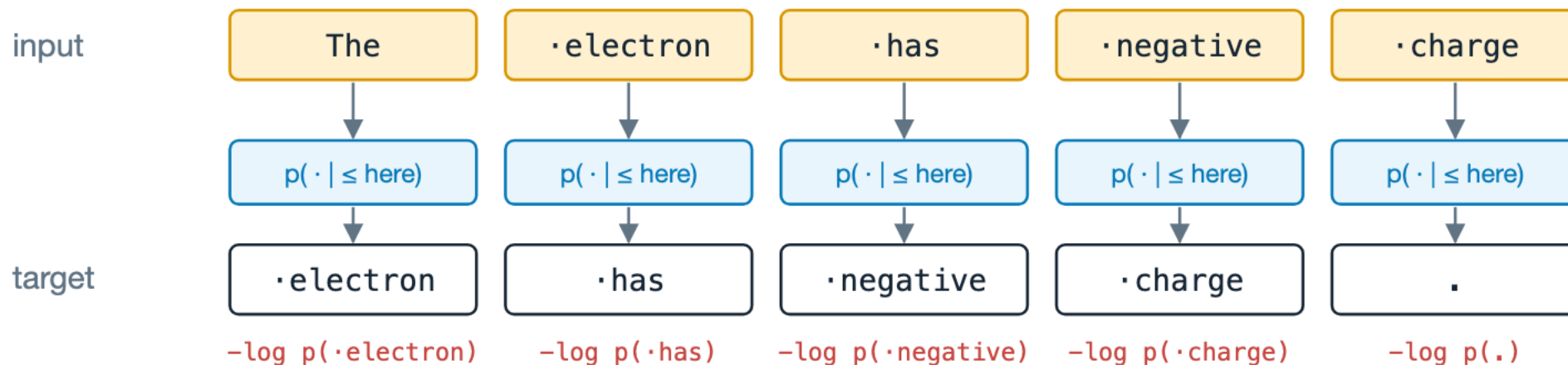
decoder: causal

decoder attends to encoder output

translation, speech to text

The 2017 Transformer translated: an **encoder** read the German sentence, a **decoder** wrote the English one and looked at the encoder through cross-attention. GPT kept only the decoder, and one objective, next-token prediction, turned out to cover almost every task.

Training: predict every next token



loss = average over all positions: one text of T tokens gives T training examples, computed in one pass

$$L(\theta) = -\frac{1}{T} \sum_{t=1}^T \log p_{\theta}(x_{t+1} | x_{\leq t})$$

The causal mask lets one forward pass train all T predictions at once: no labels needed, the text is its own target.

training text

GPT-2 (2019)	~40 GB of web text, ~10 B tokens
GPT-3 (2020)	~300 B tokens
Llama 3 (2024)	~15 T tokens

From logits to text

Unembedding: logits $\mathbf{z} = W_U \mathbf{h}_T \in \mathbb{R}^{50257}$, then

$$p_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

- $T \rightarrow 0$: always the most likely token (greedy)
- $T = 1$: the model's distribution
- **top- k** : keep the k most likely tokens
- **top- p** : keep the smallest set with total probability $\geq p$

GPT-2, “The capital of France is ...”

T = 0: the capital of the French Republic, and the capital of the French Republic is the capital of the French Republic...

T = 0.7: still the biggest producer of oil and gas. (Reuters)...

T = 1.5: Barville , “A vice801 noted.” Mr Wallace wanted ahead standpoint...

Choosing the next token

Choosing the next token

real GPT-2 (124 M) next-token probabilities

The model returns a logit z_i for all 50 257 tokens. Sampling uses $p_i \propto \exp(z_i/T)$, optionally keeping only the top- k tokens or the smallest set with probability $\geq p$.

Prompt The capital of France is ...

Temperature T

1.00

top-k

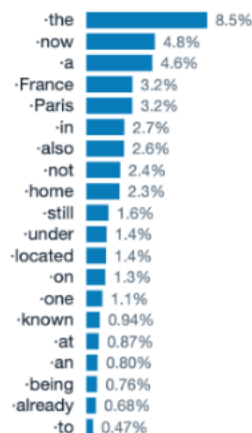
200

top-p

1.00

Sample 50

"The capital of France is ..."



all other 50237 tokens

54.3%

samples out of 50



Tokens that can be drawn

200 + ~50 000

Most likely

" the" 8.5%

Entropy (top 200)

4.56 bits

Unmodified p at T = 1

" the" 8.5%

GPT-2 continuation, T = 1.0: The capital of France is also the largest city in the world. ↵ ↵ About half of France is now part of the EU, one fifth is part of the EU

Counting parameters

GPT-2 small	parameters
token embedding $50\,257 \times 768$	38.6 M
position embedding 1024×768	0.8 M
attention, $12 \times (4d^2 + \text{biases})$	28.3 M
MLP, $12 \times (8d^2 + \text{biases})$	56.7 M
layer norms	0.04 M
total	124.4 M

Per block: $4d^2 + 8d^2 = 12d^2$. For a deep model

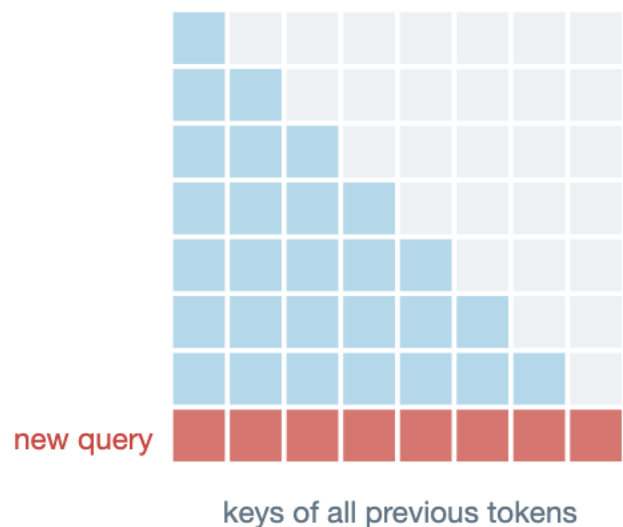
$$N \approx 12 L d^2.$$

GPT-3: $12 \times 96 \times 12\,288^2 \approx 174 \text{ B}$, reported 175 B.

Generating one token costs about $2N$ floating-point operations: 250 M for GPT-2, 350 G for GPT-3.

The cost of long texts: the KV cache

Generating token 8: only the new row is computed



KV cache

keys and values of earlier tokens never change:
store them instead of recomputing them.

cost per new token $\propto$ tokens so far (not their square)

memory per token = $2 \times \text{layers} \times \text{heads} \times d_{\text{head}}$ numbers

GPT-2 small: $2 \times 12 \times 12 \times 64 = 18\,432$ numbers ≈ 37 kB (16-bit)

DeepSeek-R1 with standard attention: ~ 4 MB per token

with multi-head latent attention (MLA): ~ 70 kB per token

The full attention pattern has T^2 entries: a 100 000-token context compares every pair of words of a book. When generating, earlier keys and values do not change, so they are cached. The cache, not the weights, often limits how many users and how long a context a server can handle; DeepSeek's multi-head latent attention compresses it 57-fold.

GPT-2 under the hood: marimo notebook

`GPT2_under_the_hood.py`:

1. tokenize any text with three tokenizers
2. next-token probabilities and generation
3. attention pattern of any layer and head
4. the logit lens for your prompt
5. embedding neighbours and analogies
6. timing with and without the KV cache
7. **a tiny GPT written from scratch**, trained on the text of this course

From the `Lecture_4A` folder:

```
1 python3.12 -m venv .venv
2 .venv/bin/python -m pip install -r requirements.txt
3 .venv/bin/marimo edit GPT2_under_the_hood.py
```

CPU only; the first start downloads GPT-2 (~550 MB). The tiny GPT (0.8 M parameters) trains in about a minute.

[Static export of the notebook](#)

Beyond text

Anything that can be cut into tokens:

- **images:** 16×16 pixel patches (Vision Transformer, 2020)
- **audio:** short frames of a spectrogram (Whisper, 2022)
- **proteins:** amino acids (AlphaFold 2, ESM)
- **particle physics:** particles or detector hits in an event (jet tagging, tracking)

Shared embedding spaces connect them: CLIP maps images and captions to the same vectors.

2017 Transformer: attention is all you need

2018 GPT, BERT

2019 GPT-2: 1.5 B parameters

2020 GPT-3: 175 B; scaling laws; ViT

2022 ChatGPT

2024–25 reasoning models, open weights (Llama, DeepSeek-R1)

Summary

1. **Tokenizer**: text $\rightarrow$ integers (BPE, trained once)
2. **Embedding**: integers $\rightarrow$ vectors, plus position
3. **Attention**: tokens read from each other,
 $\text{softmax} \left(QK^T / \sqrt{d_k} \right) V$, causal in a decoder
4. **MLP**: each token processed alone
5. Blocks **add** to the residual stream, L times
6. **Unembedding** + softmax $\rightarrow$ next-token probabilities
7. **Sampling** with temperature, repeat

Trained on one task, next-token prediction, with the cross-entropy, backpropagation and Adam of the previous lectures.

Lecture 4B: from a text predictor to a chat assistant and an agent: context windows, latency, “thinking”, tools and harnesses.

End

Thank you for the attention