The Transformer Architecture

Session 1 · 3 hours · Lecture + live-coding lab

AUTHOR
Lambert Fatoux

Other Formats
RevealJS

The Transformer Architecture

Session 1 · 3 hours

From input embeddings to output tokens: what is a decoder only block in a transformer ?

Learning outcomes (today)

1. Trace a token from **ID → embedding → attention → logits**
2. Explain **why causal masking** exists (and what breaks without it)
3. Read and check tensor **shapes** (`B, T, C`) like a production habit
4. Implement a **decoder-only transformer block** in PyTorch and verify the mask

> **Produced:** a shape-verified decoder block

Course doctrine (hold all semester)

> No generative component is **working** until it is scored on a **held-out golden set** with **cost** and **latency** tracked.

An impressive demo on the training distribution = the canonical self-deception.

In scope	Out of scope
Working **with** and **around** models	Training foundation models
Architecture enough to debug systems	Transformer-internals research

Why Session 1 exists (practice)

You will ship RAG, agents, and eval harnesses later.

Without architecture intuition you cannot answer:

- *Why is my context window "full" after 4 PDFs?* → **tokens × embeddings**
- *Why did the model "see" the answer in the prompt?* → **attention / leakage**
- *Why is first-token latency high?* → **full forward pass; KV-cache later helps**
- *Why can't I fine-tune everything?* → **parameter count lives in these matrices**

Today = the mental model for every later cost and failure mode.

Architectures

Horizon of generative models

- Before jumping right into generative models' architectures, what about the models themselves, what can you do with them ?
- We'll see models for:
  - text
  - images
  - videos
  - sound
  - poses estimation based models
  - actions (how does this make sense ?)

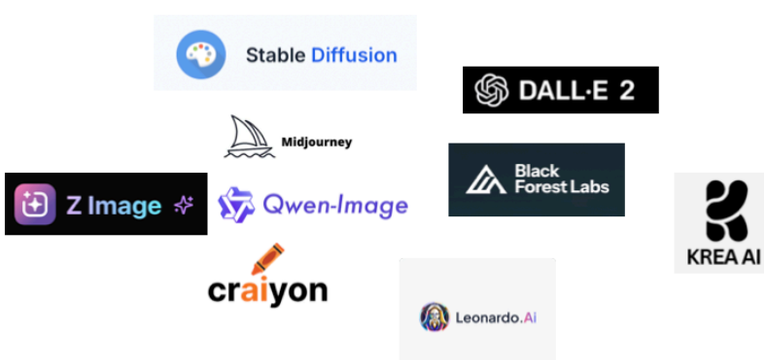
Text models

Text models leaderboard

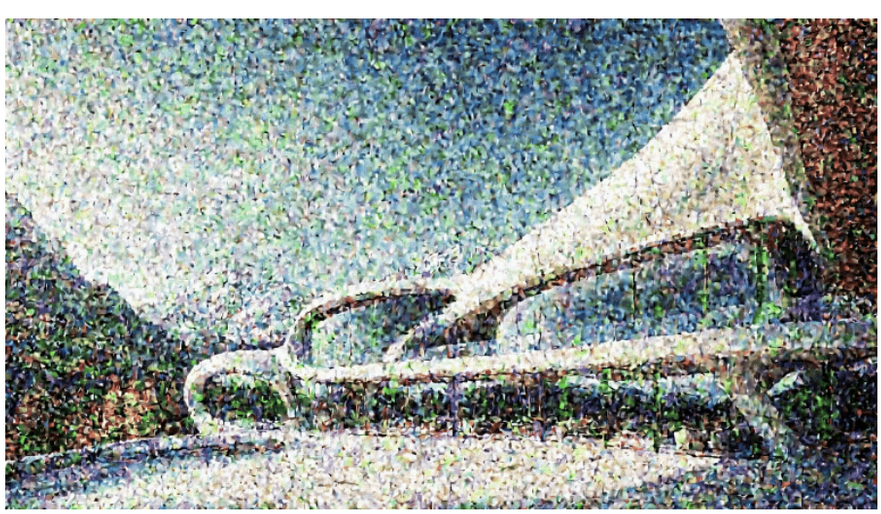
Show Filters | Models | Labs

Rank	Overall Rank	Model	Score	Rank Spread	Votes	Price $/M	Context
1	17	kimi-k3-max Moonshot · Kimi K3 license	1485 ±5	7 – 30	20 987	N/A	N/A
2	19	glm-5.3-max Z.ai · MIT	1483 ±6	7 – 37	10 960	$1.40 / $4.40	1M
3	29	glm-5.3-flash Z.ai · MIT	1475 ±7	14 – 49	10 038	$0.06 / $0.20	1M
4	38	glm-5.2-max Z.ai · MIT	1472 ±5	21 – 51	36 798	$1.40 / $4.40	1M
5	44	mimo-v2.5-pro Xiaomi · MIT	1467 ±4	28 – 53	60 919	$0.43 / $0.87	1.1M
6	47	glm-5.1 Z.ai · MIT	1466 ±4	30 – 56	48 901	$1.40 / $4.40	202.8K
7	50	deepseek-v4-pro-high-20260… DeepSeek · MIT	1463 ±7	28 – 68	9 008	$1.32 / $3.96	N/A
8	52	kimi-k2.6 Moonshot · Modified MIT	1460 ±5	41 – 68	37 502	$0.95 / $4	262.1K
9	56	glm-5 Z.ai · MIT	1458 ±4	46 – 72	27 605	$1 / $3.20	204.8K
10	57	deepseek-v4-pro DeepSeek · MIT	1457 ±4	48 – 71	54 130	$1.32 / $3.96	1M

Image models



Diffusion mechanism



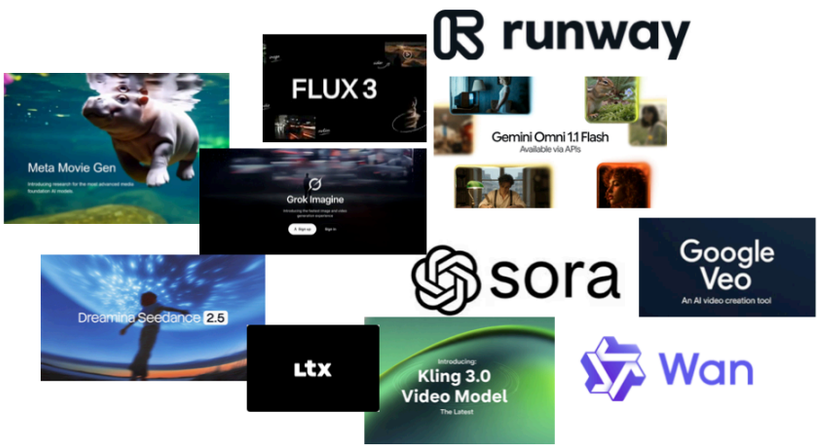
Text-to-Image leaderboard

Show Filters

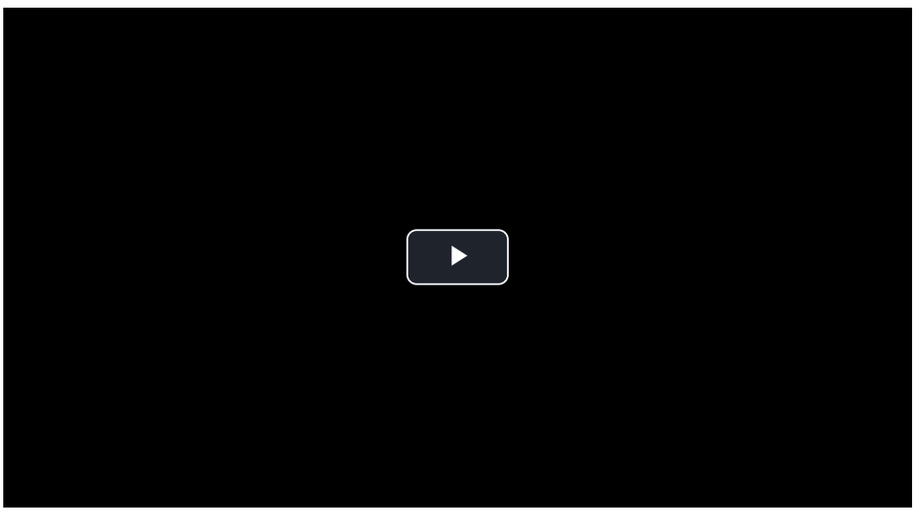
Models | Labs

Rank	Rank Spread	Model	Score	Votes
1	1 ↔ 1	gpt-image-2.5-sunburst OpenAI · Proprietary	1423 ±9 Preliminary	8 791
2	2 ↔ 2	gpt-image-2.5-flare OpenAI · Proprietary	1401 ±9 Preliminary	8 014
3	3 ↔ 3	gpt-image-2 (medium) OpenAI · Proprietary	1381 ±4	82 862
4	4 ↔ 4	mai-image-2.6 Microsoft AI · Proprietary	1334 ±6	16 310
5	5 ↔ 6	grok-imagine-image-2.0 (low) SpaceXAI · Proprietary	1302 ±9	5 369
6	5 ↔ 6	reve-2.1 Reve · Proprietary	1301 ±8	7 582
7	7 ↔ 8	muse-image Meta · Proprietary	1276 ±5	31 686
8	7 ↔ 9	reve-2.0 Reve · Proprietary	1270 ±6	14 625
9	8 ↔ 13	gemini-3.1-flash-image (nano-banana-2) [web-search] Google · Proprietary	1260 ±4	47 988
10	9 ↔ 13	seedream-5.0-pro Bytedance · Proprietary	1256 ±4	76 831

Video models



Video example (Flux 3)



Audio models

Generate audio from text

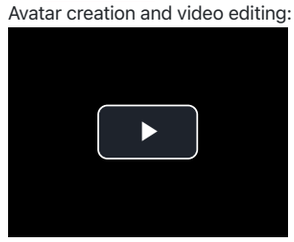
WellSaid

Kokoro: 00:00 00:20

Audio8: 00:00 00:10 00:00 00:08

Others

- Muse from Meta, personnal assistant -> ⚠️ privacy
- OpenMuse: same but open source and self hosted
- Jev TypeSafe AI: alternative to standard LLMs for strctured outputs
- Avatar creation and video editing:



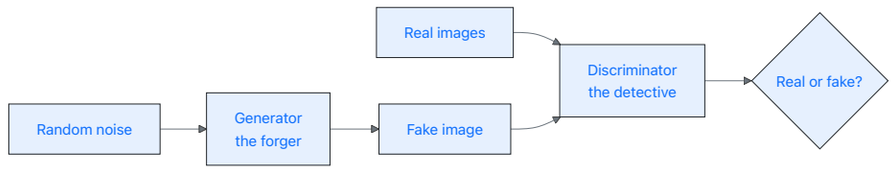
- Hyperframes

: generate videos from HTML

Before transformers: other generative families

Family	How it generates	Famous for	Today
RNN / LSTM (1986 / 1997)	Reads and writes one token at a time, carrying a memory	Google Translate (2016), Siri, autocomplete	Comeback as Mamba, xLSTM
GAN (2014)	A generator tries to fool a detector	Fake faces, deepfakes, StyleGAN	Fast image upscaling, still used inside image pipelines
Energy-based (1982 →)	Scores how "plausible" an input is, then slides toward low energy	Hopfield nets, Boltzmann machines	Ideas live on in diffusion models and LeCun's JEPA
Transformer (2017)	Attention over all previous tokens	GPT, Claude, Llama	Dominant for text, also inside image and video models

GANs: a forger vs a detective



- Two networks train **against each other**: the generator improves until the discriminator can no longer tell
- Very sharp images, but **unstable training** and *mode collapse* (the forger learns one good face and repeats it)
- 👉 Refresh thispersondoesnotexist.com: a new StyleGAN face every time. Play with a GAN in your browser: GAN Lab

> 💡 🎲 **Fun fact (but I'm not sure it's true)**
> Ian Goodfellow had the idea in a Montreal bar in 2014 and coded it the same night

Energy-based models: rolling downhill

- Learn an **energy** function: low energy for plausible data (a real face), high energy for nonsense (random pixels)
- Generate by starting from noise and **rolling downhill** toward low energy
- Elegant, but sampling is slow: many small steps for each sample
- Diffusion models (Session 7) follow the same intuition: they learn **which direction** makes noise more plausible

> 💡 🎲 **Fun fact**
> The **2024 Nobel Prize in Physics** went to John Hopfield and Geoffrey Hinton, for the Hopfield network (1982) and the Boltzmann machine (1985). Both are energy-based models borrowed from statistical physics.

RNN and LSTM: one word at a time

- **RNN:** reads a sentence word by word and updates a **hidden state**, a running summary of everything read so far
- Problem: the summary fades. After ~20 words, the start of the sentence is almost forgotten (**vanishing gradient**)
- **LSTM** (Hochreiter & Schmidhuber, 1997) adds **gates** that decide what to keep, forget and output: memory over hundreds of steps
- Still **sequential even in training**: to learn from a known sentence, word 100 waits for word 99's state. A transformer processes all 100 positions at once. Both still generate one token at a time

> 💡 🎲 Fun fact
>
> Karpathy's tiny character-level RNN produced fake Shakespeare, fake Wikipedia pages and fake Linux source code that *almost* compiles. And RNNs are back: **Mamba** (2023) and **xLSTM** (2024, from Hochreiter's own team) process very long sequences with a fixed-size memory.

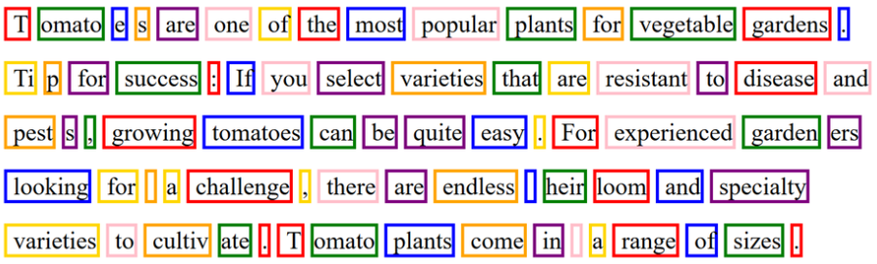
Large Language Models (LLMs)

- Goal is to generate text based on some input (generally also text)
- How do we do in practice ?
  - Try to predict the next *word* based on the previous ones
  - Use that new sequence of words to predict the second next word
  - and so on...

> This idea has been around for a long time now, statistical models were developped with information theory concepts decades ago !

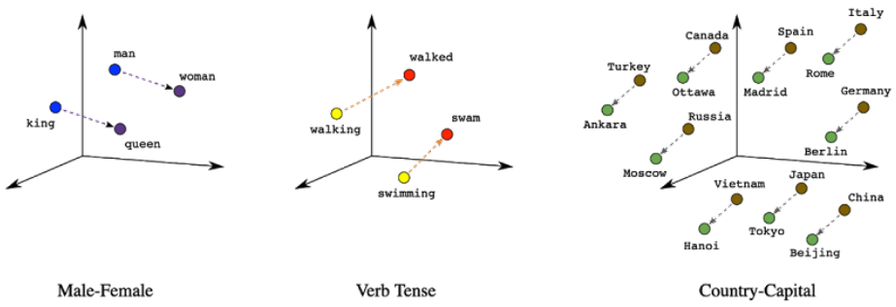
Tokens

- In fact models that we'll talk about do not use *words* as objects but **tokens** and **embeddings**
- This helps to group semantics concepts together (**normal**, **normal**ly, **normal**ity, ...)
- Some tokens do not even represent words but also punctuation or emojis: " ";"!", 👍 etc.
- In english, usually 4 characters/token but it depends on how you **tokenize** a word



Embeddings

- Once words have been converted to tokens, they must somehow get a numerical representation that the classical operators (sum, substraction, multiplication, etc.) can use
- Doing so provides **embeddings**, concretely vectors living in a **latent space**
- This allows to perform mathematical operations that have a semantic meaning



Embedding example with lower dimension

How to get embeddings ?

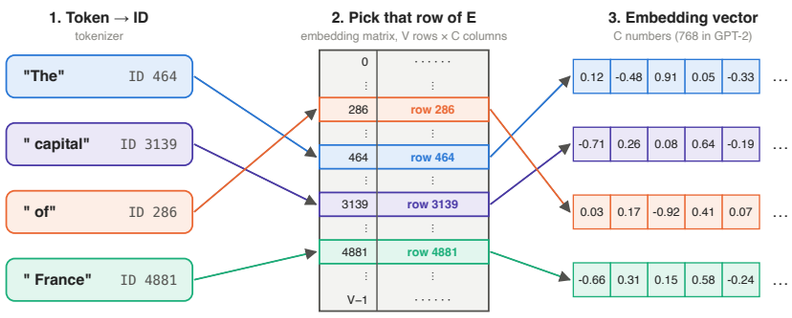
- **Lookup table** (inside every LLM): one big matrix, one row per token. The token ID simply picks its row. It starts random and is learned during training
- **Word2Vec / GloVe** (2013): learn a vector per word by guessing a word from its neighbours: *"you shall know a word by the company it keeps"*
- **Contextual embeddings** (BERT, GPT): after the transformer layers, "bank" gets a different vector in *river bank* and *bank account*
- **Sentence embeddings** (sentence-transformers, embedding APIs): one vector for a whole sentence or document, ready for search (Session 5)

```
from transformers import AutoModel, AutoTokenizer

tok = AutoTokenizer.from_pretrained("gpt2")
model = AutoModel.from_pretrained("gpt2")
```

```
ids = tok("The capital of France", return_tensors="pt").input_ids  # [[464, 3139, 286, 48
E = model.get_input_embeddings()                                   # Embedding(50257, 768
vectors = E(ids)                                                   # shape (1, 4, 768)
```

Token → embedding: a table lookup



E starts random and is learned during training: tokens used in similar contexts end up with nearby rows.

Model representation of text

- In order for a model to perform text tasks, it must
  - decompose sentences into smaller chunks called **tokens**
  - get a representation of a token into an **embedding**
  - get the **next** embedding
  - transform that next embedding **back into a token** that can be readable by a human

This is what we call a **decoder-only** language model

> GPT2: embeddings have a dimension of 768 GPT3: embedding dimension of 12,288 Claude Fable: not publicly available, probably around 16,384

The big picture (decoder-only LM)

Here is a schema of a typical model:

```
tokens  →  embeddings  →  N × Transformer blocks  →  logits  →  next token
  IDs          + pos          (attention + FFN)      vocab        sample
```

Concrete prompt:

```
User: "The capital of France is"
Model predicts: " Paris"  (then " ." …)
```

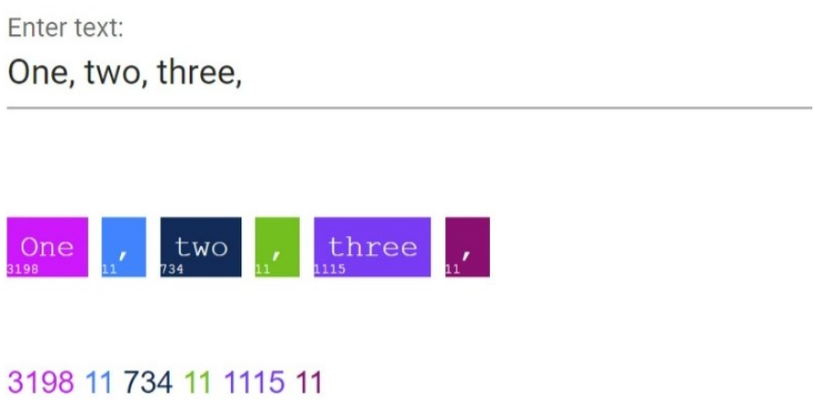
For a conversationnal model, what you want is:

- Each step: look at **previous** tokens only, pick the next one.
- That is **autoregressive** generation — ChatGPT, Claude, Llama all do this.

> But what is concretely this **Transformer Block** with attention and FFN ?

From embeddings to probability

- Once we have embeddings of the **previous tokens**, we want to get the **next most likely token** to appear in order to complete the sentence
- How could we do so ?
- Here is an example of what we want in the end:



#	probs	next token ID	predicted next token
0	46.44%	1440	four
1	7.48%	290	and
2	7.31%	1936	five
3	2.66%	393	or
4	2.54%	2237	six
5	2.09%	1115	three
6	1.86%	3863	maybe
7	1.62%	345	you
8	1.23%	257	a
9	0.92%	530	one

Example of next token probabilities

What has been tried

- Simple Multi Layer Perceptron (MLP) -> no links between words, each token is processed individually
- RNN, LSTM -> vanishing gradient, short memory, etc.

Nothing seemed to really crack the problem at hand, until...

Transformer architecture to the rescue

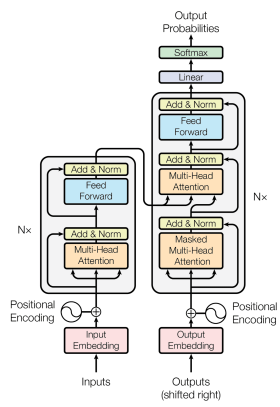
- *Attention is All You Need*, Introduced in 2017 by Google a new network architecture: **The Transformers**
- Significant improvements in the *understanding* of relationships between words (check out this interactive visualization)

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: the load-bearing object

> Attention = soft routing: each token builds a **weighted summary** of other tokens.

For one query position ("is" in *The capital of France is*):

- Query ("is") asks: who has the answer?
- Keys (all tokens) advertise: what I contain
- Values (all tokens) carry: what I contribute

Intuition: the model learns to put high weight on **France** when predicting **Paris**.

Transformers: key takeways

- For a given embedding vector of token i, the dot product $q_i \cdot k_j$ is going to be computed for every other token j
- This tells us **how much information** does token j contain relevant to token i
- The goal is to **update the vector representation of token** i with a new representation that also contains information about the full context

Scaled dot-product attention (one head)

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

Symbol	Role	Shape (per head)
`Q`	Queries	`(B, T, d_k)`
`K`	Keys	`(B, T, d_k)`
`V`	Values	`(B, T, d_k)`
`QKᵀ`	Similarity scores	`(B, T, T)` ← **this is the `T²` cost**
softmax	Weights sum to 1	still `(B, T, T)`

Why divide by √d_k? Keeps scores from exploding → softmax stays usable.

Practice: long context hurts latency because of that `(T, T)` matrix (plus memory).

Concrete mini-example (3 tokens)

Sequence: `["The", "cat", "sat"]` — predicting after `"sat"`.

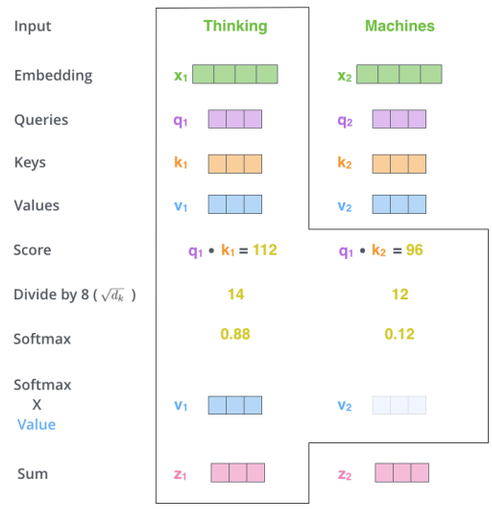
Illustrative attention weights for query `"sat"`:

Key →	The	cat	sat
weight	0.1	0.6	0.3

Output ≈ `0.1·V_The + 0.6·V_cat + 0.3·V_sat`

The model "looks at" **cat** most — useful for guessing *on the mat* vs *in the car*.

Illustration



Source: jalammar.github.io

Position: order is not free

Embeddings alone do **not** know that token 1 is before token 5.

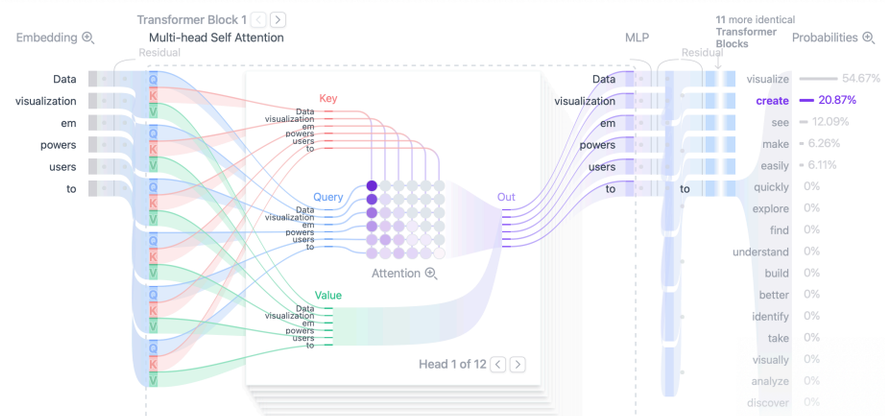
```
"dog bites man"  ≠  "man bites dog"
same bag of embeddings without position → model is lost
```

Approach	Idea	Practice note
Absolute (sin/cos or learned)	Add a vector per index `0..T-1`	Simple; length limits feel hard
Relative / RoPE (modern LMs)	Encode relative distance in attention	Better length extrapolation story

You need the idea, not the research paper: every production LM injects position somehow.

Product translation: RAG stuffing 50 chunks into context only helps if attention *actually* routes to the right spans (rerankers exist for this — Session 5).

Visualization



From on attention head to full model

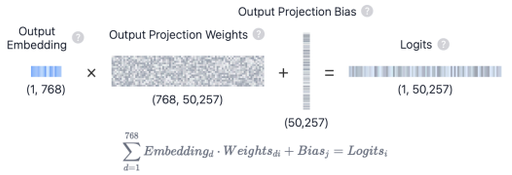
- Modern models use **mutliple attention heads** that can run in parallel in order to grasp multiple concepts from vectors

. . .

- These parallel computations are extremely fast on GPU hardware (using PyTorch, Tensorflow, Cuda)

. . .

- In the end we end up with a single vectors that is passed to a final MLP



Multi-head attention

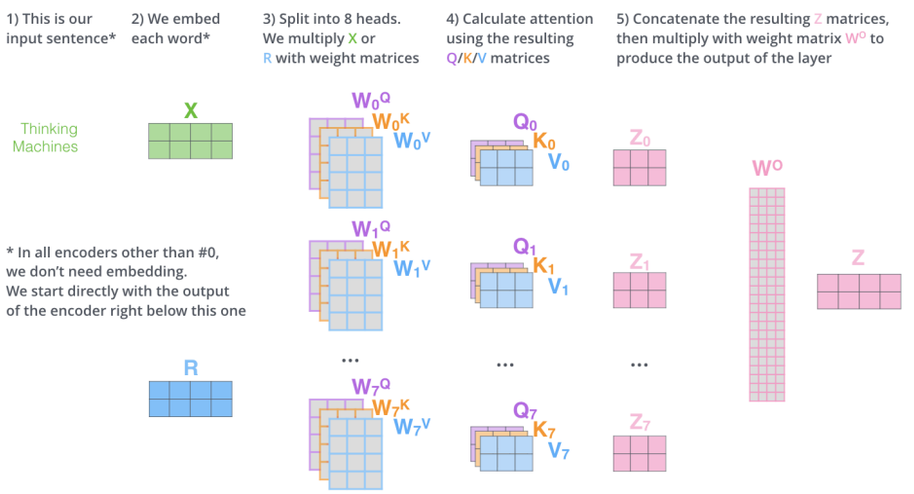
One head = one soft routing pattern.
Several heads = several patterns in parallel.

```
Head 1: syntax (subject ↔ verb)
Head 2: entities (France ↔ capital)
Head 3: copy / formatting
...
Concatenate → project back to d_model
```

Why multiple heads?	Practice consequence
Different relationships at once	`n_heads` is a hyperparameter in every LLM config
`d_model` split across heads	`d_k = d_model / n_heads` must divide evenly

Config literacy: when a model card says `n_heads=32, d_model=4096`, you now know what that means.

Multi-head schema



Source: jalammar.github.io

Residual stream + LayerNorm

A transformer block is **not** "replace `x` with attention."
It is **add a refinement**:

```
x = x + Attention(Norm(x))
x = x + FFN(Norm(x))
```

Piece	Job	Why you care
Residual `+`	Keep original signal; train deep stacks	Explains why deep LMs are trainable
LayerNorm / RMSNorm	Stabilise activations	Pre-norm (modern) vs post-norm (classic)
FFN / MLP	Per-token nonlinear mix (`C → 4C → C`)	Often **most parameters** in the block

Practice: LoRA (Session 3) often targets attention and/or FFN projections — those matrices are where adaptation lives.

Causal mask: can you see the future ?

Decoder-only LMs must **not** see the future.

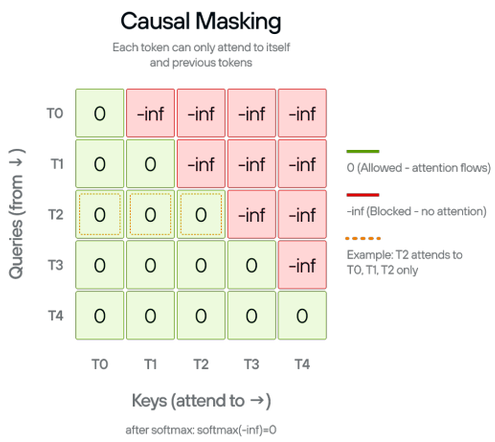
```
Position:  0     1     2     3
Token:    The  capital  of  France
```

```
When predicting token 2 ("of"),
legal context = The, capital
ILLEGAL     = of, France   ← future
```

Mask sets illegal scores to **−∞** before softmax → weight **0**.

```
Attention mask (lower-triangular):
     0  1  2  3
0    ✓  ✗  ✗  ✗
1    ✓  ✓  ✗  ✗
2    ✓  ✓  ✓  ✗
3    ✓  ✓  ✓  ✓
```

Causal Masking



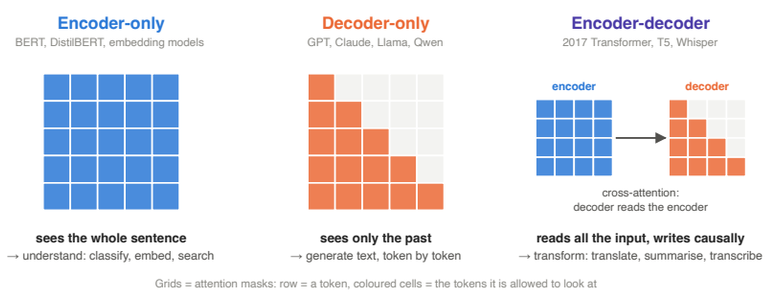
What breaks without a causal mask?

- Training leakage: model "cheats" by reading the answer token
- Nonsense generation: autoregressive sampling no longer matches training
- Eval self-deception: loss looks great; generation is garbage

Lab goal: prove with a probe that future positions get **zero** attention weight.

> If you only remember one diagram from today, remember the lower-triangular mask.

Three families of transformers



- Same building blocks (attention, FFN, residuals): the **mask** decides the family
- Today's lab model `distilbert-base-uncased` is an **encoder**: great for embeddings, it cannot generate text
- Decoder-only won for chat: one simple recipe that scales, and it can still classify, translate or summarise when asked in the prompt

> 💡 🎲 **Fun fact**
>
> The 2017 *Attention is All You Need* transformer was an **encoder-decoder** built for translation (English → German and French). GPT (2018) kept only the decoder, BERT (2018) only the encoder.

From hidden states to tokens

```
logits = h @ W_LM^T          # LM head: (B, T, V)
probs  = softmax(logits)     # distribution over vocabulary
next   = argmax or sample    # pick one ID → decode to text
```

Concrete:

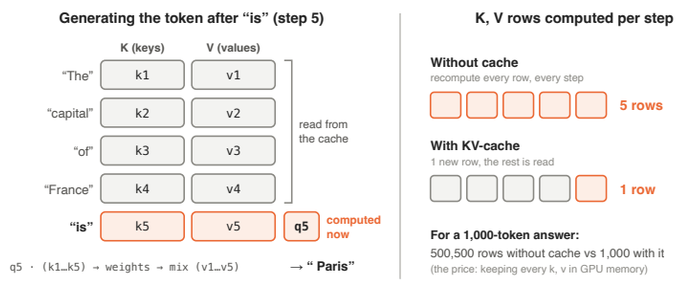
```
Context: "The capital of France is"
Logits peak at ID for " Paris"
Decode → append → repeat
```

Decoding choice	Behaviour	Practice

Greedy argmax	Deterministic, dull	Good for structured tasks
Temperature sampling	Creative / risky	Chat, images-of-text prompts
Top-p / top-k	Truncate tail	Production default family

Inference vs training (KV-cache preview)

- **Training**: full sequence in parallel with the causal mask (big matmuls, high VRAM)
- **Naive generation**: recompute keys and values of the whole prefix for every new token, painfully slow
- **KV-cache**: past keys and values never change (causal mask!), so store them and only compute the new row



KV-cache: the memory bill

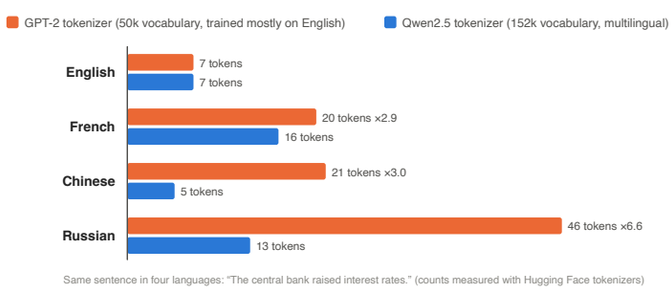
Per token: **2** (K and V) × layers × KV heads × head dim × **2 bytes** (fp16)

Model	Layers	KV heads × dim	Per token	Full context
GPT-2 small	12	12 × 64	36 KB	1k tokens → 36 MB
Qwen2.5-0.5B	24	2 × 64	12 KB	32k tokens → 384 MB
Llama 3.1 8B	32	8 × 128	128 KB	128k tokens → **16 GB**

- A full 128k context of Llama 3.1 8B needs as much memory as **the model weights themselves** (16 GB)
- Modern models share KV heads between query heads (**GQA**): with 32 KV heads instead of 8, that cache would be 64 GB
- **Time to first token** = process the whole prompt and fill the cache; **tokens per second** = read the cache for each new token

Tokenisation: a practical cost meter

The tokenizer fixes the length **T** before the model even runs, and you pay **per token**.



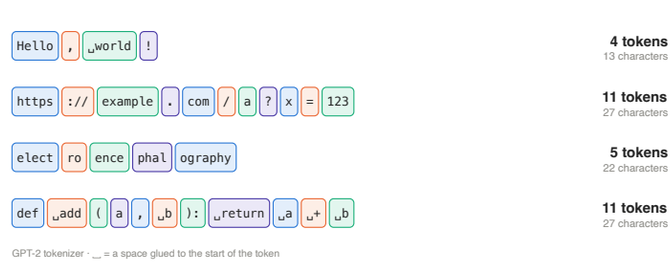
> ⓘ 💼 Business corner
>
> Serving customers in Russian with an English-centric tokenizer: **6.6× the tokens** for the same message, so 6.6× the API bill, slower answers and a context window that fills 6.6× faster. Petrov et al. (2023) measured gaps of **up to 15×** between languages. Check token counts in *your* languages before signing a pricing plan.

Tokenisation checkpoint

Your guess: how many tokens for each line?

```
Hello, world!
https://example.com/a?x=123
electroencephalography
def add(a, b): return a + b
```

. . .



The question is not “how many characters?” but **“how many vocabulary pieces?”** URLs and code are expensive; plain English is cheap.

Vocabulary size V

The vocabulary size **V** sits at both ends of the model: the **embedding table** (`V × C`) on the way in, the **logits** (`B × T × V`) on the way out.

Model	V	C	Embedding table	Share of all parameters
GPT-2 small	50,257	768	38.6M	**31%**
Qwen2.5-0.5B	151,936	896	136M	**28%**
Llama 3 8B	128,256	4,096	525M (×2: in + out)	**13%**

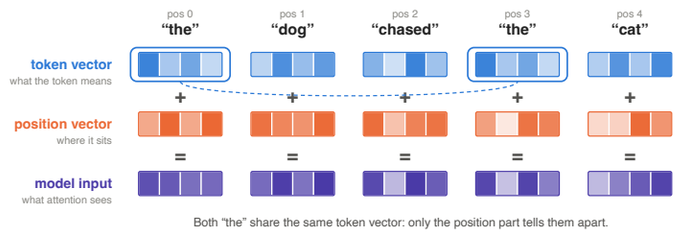
. . .

- **Bigger V:** fewer tokens per text (Russian: 46 → 13 tokens), but a bigger table and a bigger softmax
- **Smaller V:** cheap table, but long sequences, and attention costs grow with T^2

. . .

> 💡 🎲 Fun fact
>
> **SolidGoldMagikarp** (2023): GPT-3’s vocabulary contained Reddit usernames that almost never appeared in its training text. Their embedding rows were never really trained, and asking the model to repeat ” SolidGoldMagikarp” produced insults, evasions or the word “distribute”.

Positional information: what goes in



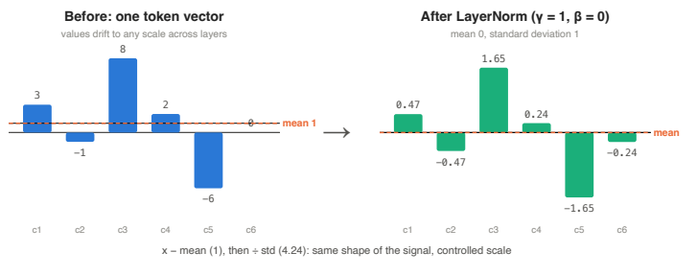
- Each input vector = **what** the token is + **where** it sits
- Without position, “the dog chased the cat” and “the cat chased the dog” look identical to attention
- **RoPE** technique is often used today: *rotate* query and key vectors by an angle proportional to the position.

Attention interpretation limits

- Some heads clearly specialise: punctuation, previous token, subject ↔ verb, copying a name seen earlier
- Attention maps are a good **debugging lead**: “the model never looked at the contract clause”
- But they are **not an explanation**: information also flows through residuals, FFN layers and dozens of other heads

> 💡 🎲 Fun fact
>
> **Golden Gate Claude** (Anthropic, 2024): researchers found an internal *feature* for the Golden Gate Bridge and turned it up. The model then claimed to *be* the bridge. Interpretability works on features, not just on attention weights. See here

Layer normalization



$$\mathrm{LayerNorm}(x) = \gamma \frac{x - \mathrm{mean}(x)}{\sqrt{\mathrm{var}(x) + \varepsilon}} + \beta$$

- Same idea as `StandardScaler`, but **per token**, across its C features, at **every** layer
- γ and β are learned: the model can still choose its own scale and offset
- Keeps values in a stable range so 30+ stacked layers can train

Transformers applications

What can you do with such models ?

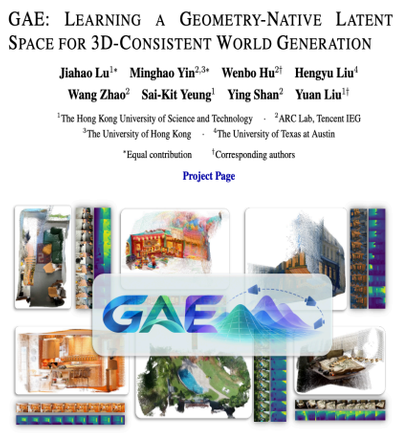
- All you favorite LLMs revolve around the same idea: Claude Opus 5.5, GPT Astra 6, GLM 5.3,

DeepSeek, Gemini, ...

- One of the first application was to translate documents, now you see why

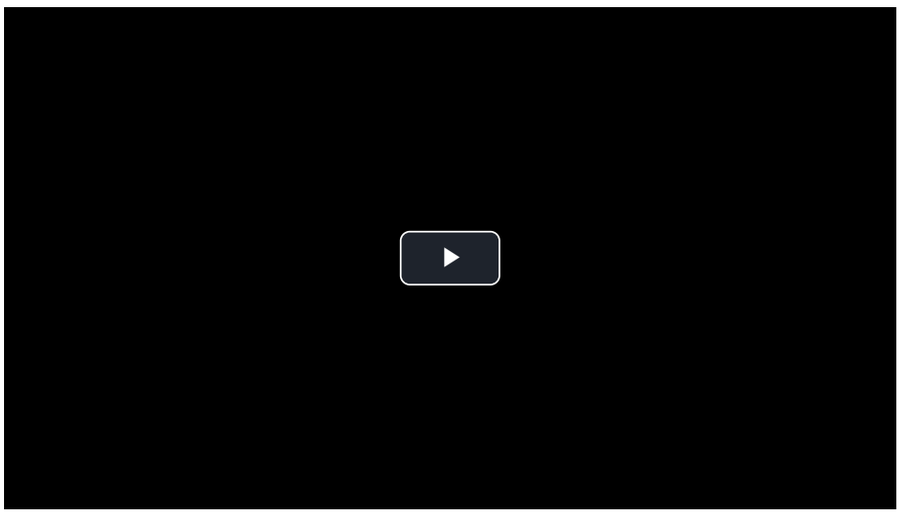
MarigoldV2

. . .



GAE

. . .



Speed matters

. . .

- Generative models often lack of long term memory: we've seen why today !

. . .

- Found solutions: establish a **knowledge database** that a model can request, **create a 3D map** for image and video models, and so on

Practice session

Lab task 1: hands on tokens and embeddings

- Use an already trained model to get tokens and embeddings from english text
- The model you'll be using is imported from the `transformers` library
- transformer model `distilbert-base-uncased` only has 66,362,880 trainable parameters!

```
from transformers import AutoModel, AutoTokenizer

MODEL_NAME = "distilbert-base-uncased"  # small, runs on CPU

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
model = AutoModel.from_pretrained(MODEL_NAME, output_hidden_states=True)
```

Lab task 2: Single causal attention head

- Build a single causal attention head using basic pytorch objects (no `nn.Transformer` ;))
- Compute the attention score using Q, K matrices:

```
scores = q @ k.transpose(-2, -1) / math.sqrt(self.C)
```

- Apply a causal mask with negative infinity to the attention scores before calling softmax

```
causal_mask = torch.tril(torch.ones(seq_len, seq_len, dtype=torch.bool))
```

Lab task 3: complete the attention path

- Compute scaled scores
- normalise them with softmax, multiply by V
- concatenate the heads and apply the output projection.

At every stage, compare the observed shape with the expected shape written on the slide.

Lab task 4: add residual and MLP paths

Use pre-normalisation:

```
x = x + attention(LayerNorm(x))
x = x + MLP(LayerNorm(x))
```

The value returned by the block must have the same B × T × C shape as its input

Common lab pitfalls

Symptom	Likely cause
`softmax` sees NaNs	Forgot to mask with `-inf` (used `0` instead)
Shape `(B, T, T)` vs heads	Missing reshape/transpose for multi-head
Loss of residual	Overwrote `x` instead of `x = x + …`
`d_model % n_heads != 0`	Illegal head split

How today's concepts show up next week

Later topic	Hook from Session 1
BPE & Chinchilla (S2)	`Embedding dim` and `vocabulary size` are economic variables
LoRA (S3)	adapters on Q/K/V/FFN matrices
Structured decoding (S4)	softmax over constrained token sets
RAG (S5)	large `embedding dim` → cost; attention may ignore your chunk
Agents (S6)	each thought/tool step = more forward passes
Eval (S8)	architecture bugs look like "model quality"

Checklist

Before you leave, you can explain:

- ☐ Token ID → embedding row
- ☐ Why position is added / injected
- ☐ What Q, K, V do in one sentence each
- ☐ Why attention cost grows with `T²`
- ☐ What the causal mask forbids
- ☐ Why residuals keep depth trainable
- ☐ How logits become the next token