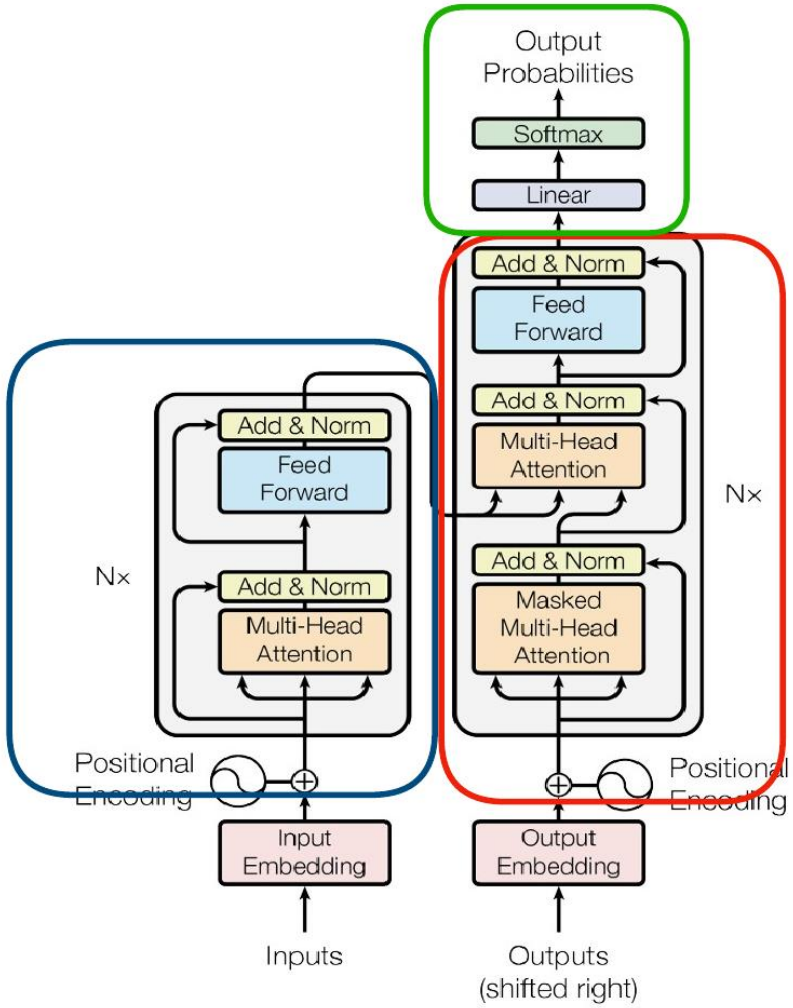
Machine Learning I
60629A

Transformers, part 2: from transformers to LLMs

— Week #12



Today's plan



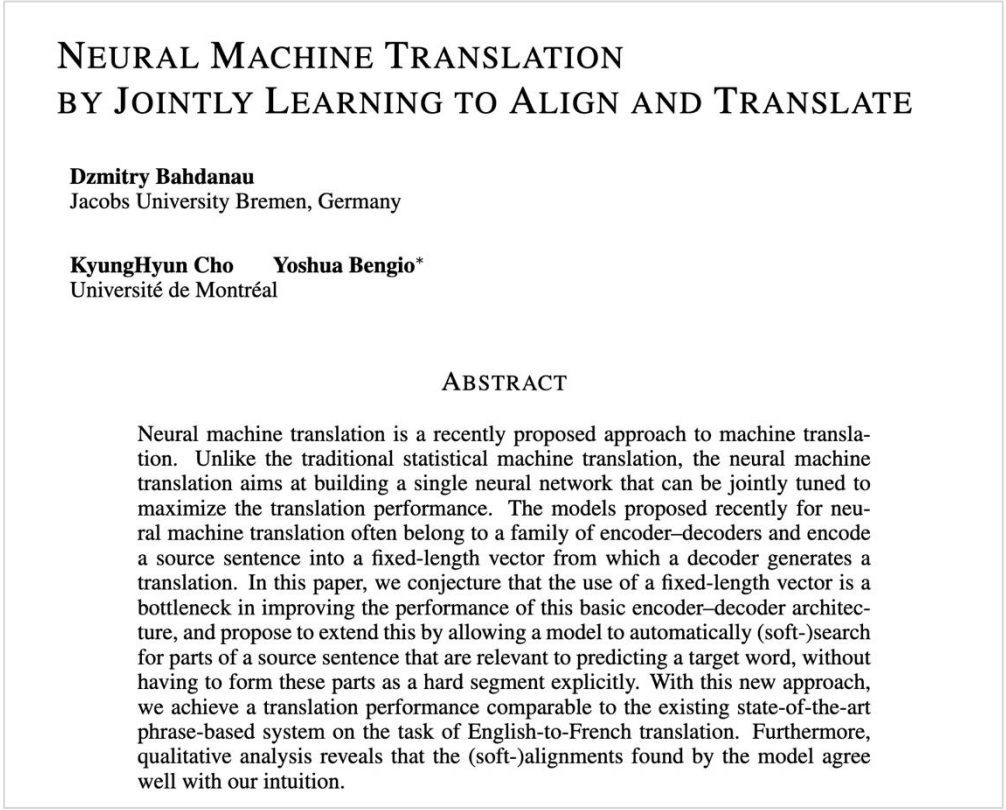
Recap of last week	15 min
Finishing the transformer	25 min
Training a language model	20 min
Break	10 min
From a language model to an assistant	30 min
LLM agents	20 min
Looking ahead	10 min

Recap of last week

Attention, self-attention and the transformer block

Where does attention come from?

- Attention was introduced for machine translation by Bahdanau, Cho & Bengio (2014)
  - Developed in Yoshua Bengio's lab, in Montréal
- Goal: remove the bottleneck of RNN encoder-decoders
- 2017: the transformer keeps the attention and drops the recurrence
  - “Attention Is All You Need” (Vaswani et al.)



NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

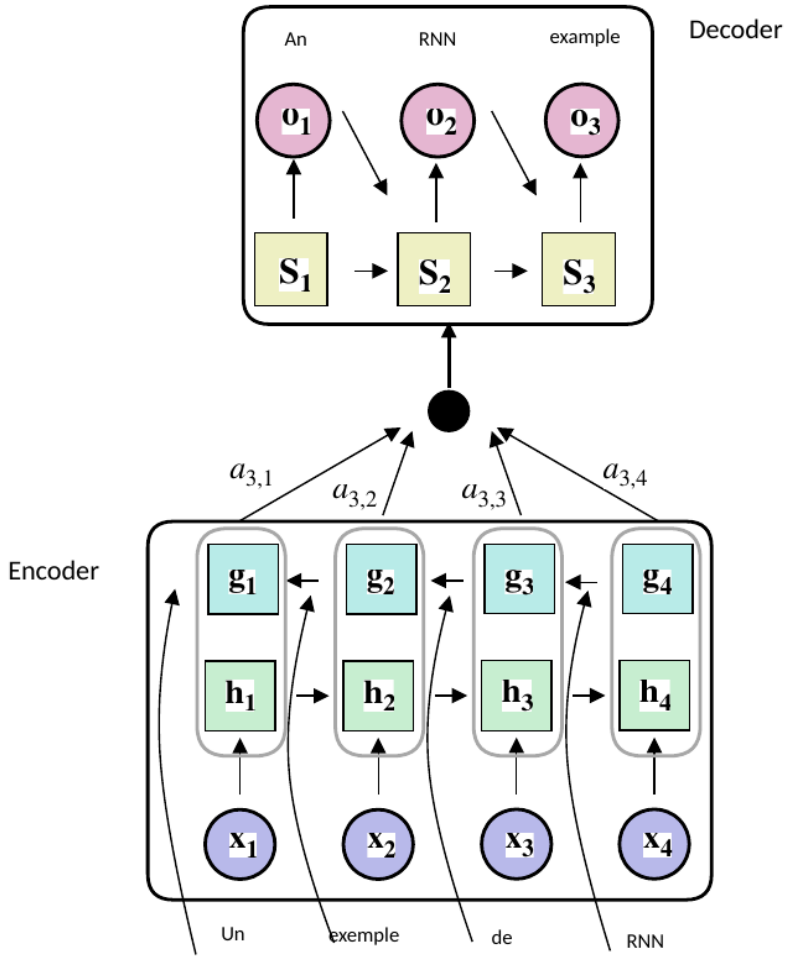
KyungHyun Cho **Yoshua Bengio***
Université de Montréal

ABSTRACT

Neural machine translation is a recently proposed approach to machine translation. Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neural network that can be jointly tuned to maximize the translation performance. The models proposed recently for neural machine translation often belong to a family of encoder–decoders and encode a source sentence into a fixed-length vector from which a decoder generates a translation. In this paper, we conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture, and propose to extend this by allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. With this new approach, we achieve a translation performance comparable to the existing state-of-the-art phrase-based system on the task of English-to-French translation. Furthermore, qualitative analysis reveals that the (soft-)alignments found by the model agree well with our intuition.

Bahdanau, Cho & Bengio, Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015 (arXiv 2014)

(Soft) attention

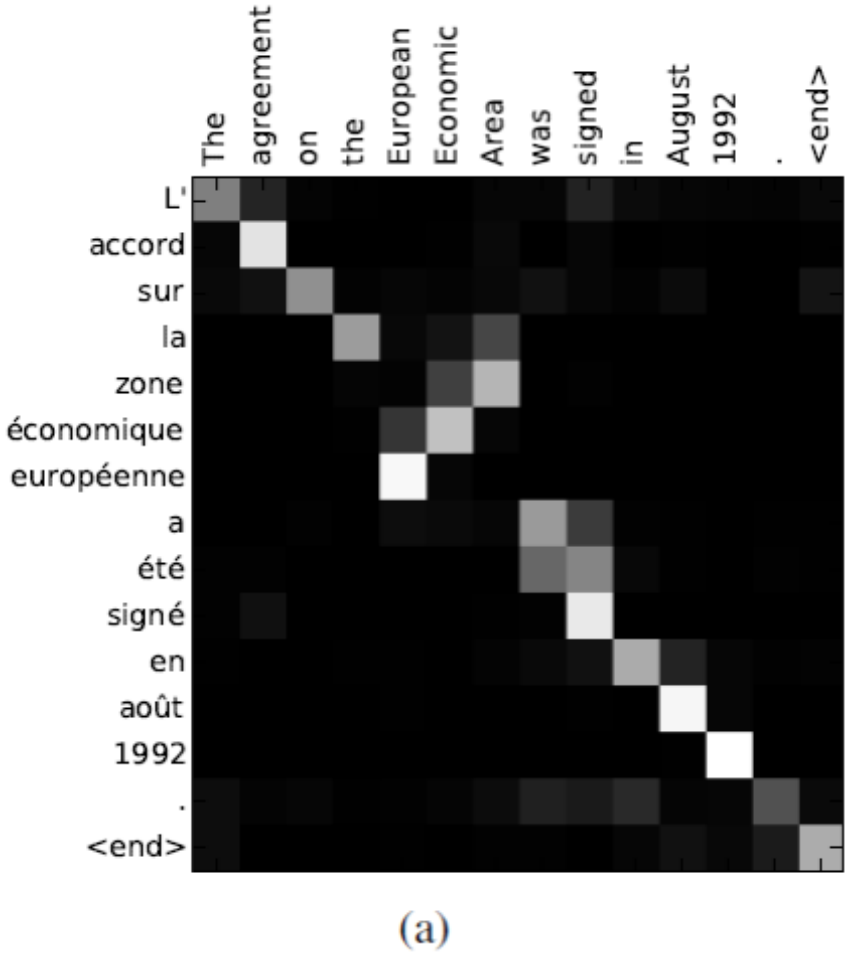


- At each output step t, the decoder looks at all the encoder states
- It combines them into a context vector:

$$c_t = \sum_{j=1}^{T_x} a_{tj} \left[g_j, h_j \right]$$

- a_tj: how much output word t attends to input word j, with Σ_j a_tj = 1
- No more bottleneck: the representation grows with the length of the input

Attention in action (EN → FR)



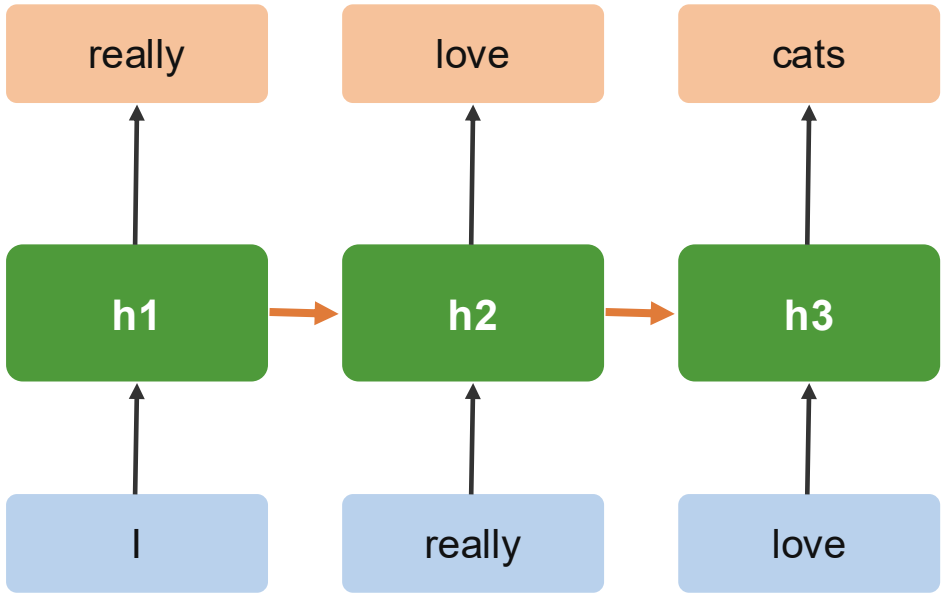
(a)

- Each row: one French output word
- Each column: one English input word
- Bright cell = large attention weight a_tj
- The weights are learned, never supervised
- Note the inversion: “European Economic Area” → « zone économique européenne »

Bahdanau, Cho & Bengio (2014), Figure 3(a)

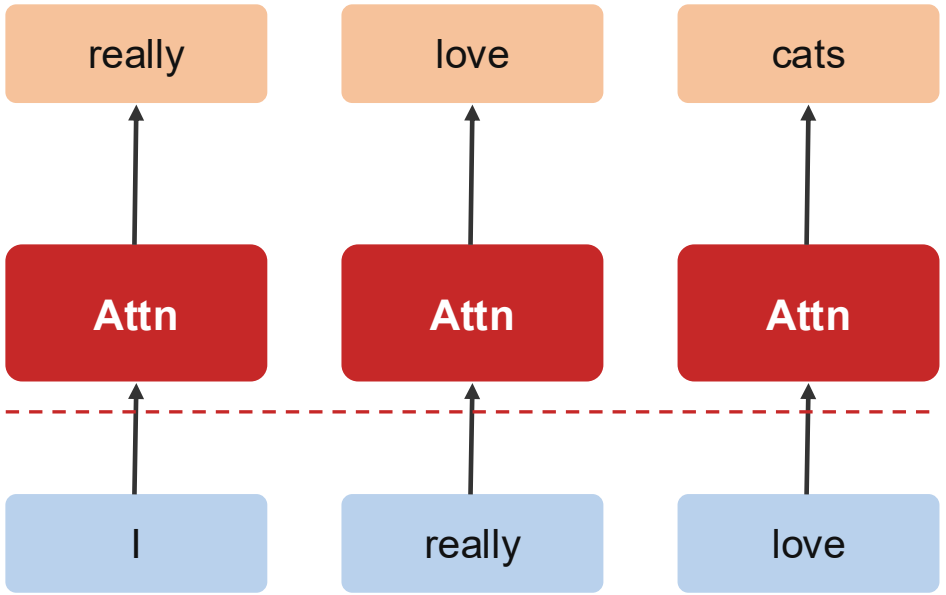
Why drop the recurrence?

RNN: one token after the other



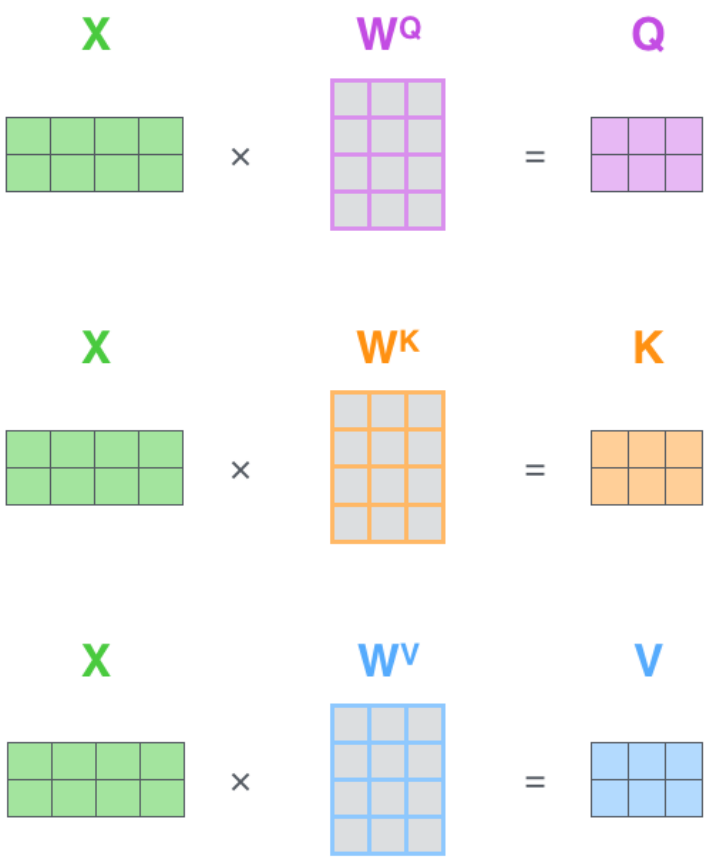
h_t needs h_t−1 (orange arrows): the computation cannot be split across positions. Slow to train on large datasets.

Transformer: all tokens at once



Every position is computed in parallel (each one attends to the others). Training parallelizes very well on GPUs.

Self-attention: queries, keys and values



Each word i is projected into a query, a key and a value. Word i compares its query to the key of every word j:

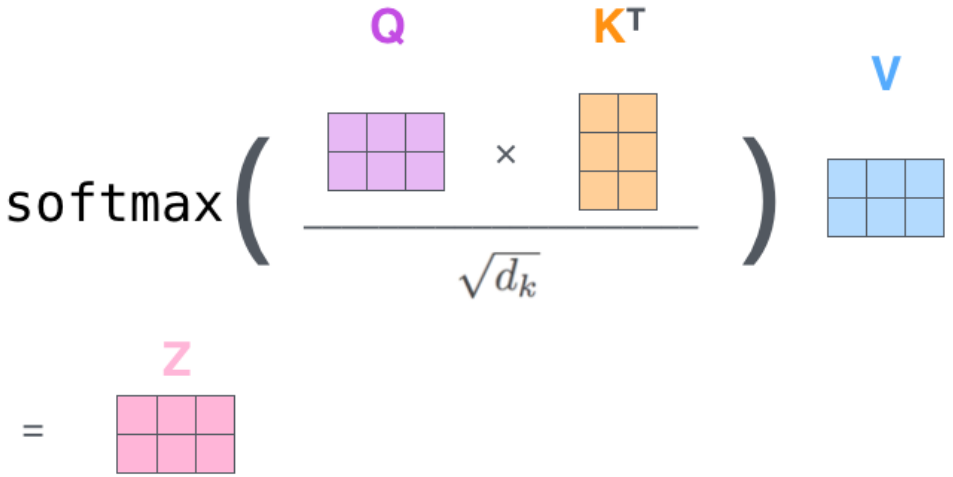
$$e_{ij} = \frac{(x_i W_q)(x_j W_k)^\top}{\sqrt{D}} \qquad a_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

Its output is the weighted sum of the values:

$$z_i = \sum_{j=1}^{N} a_{ij} \left(x_j W_v\right)$$

- Rows (i): the word that looks (query). Columns (j): the word looked at (key).
- For each i, the weights a_ij sum to 1 over j

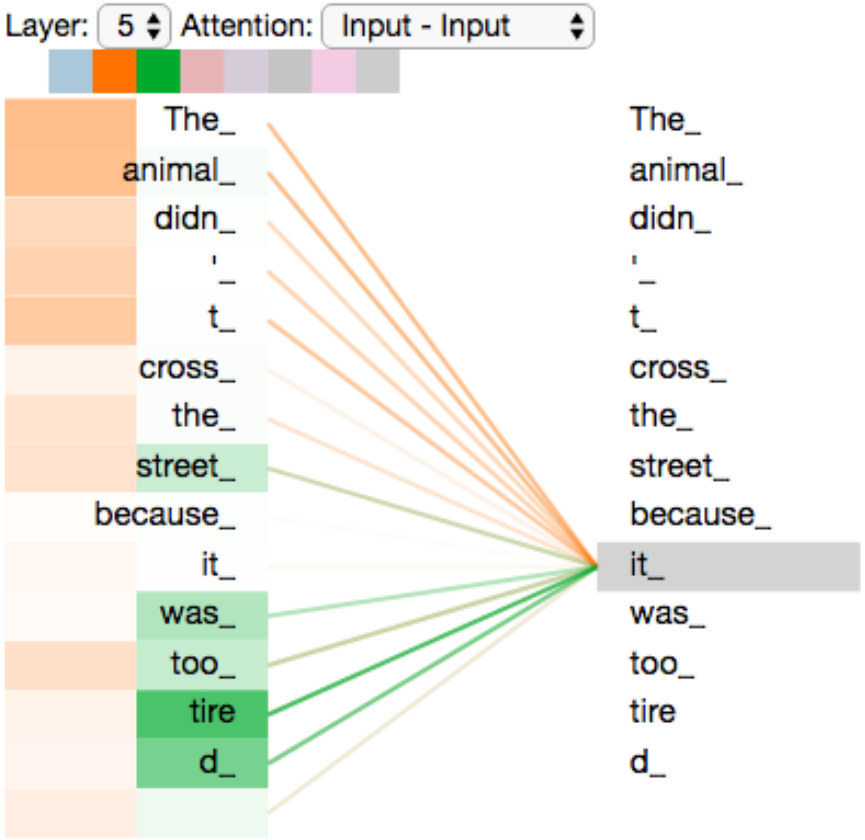
Self-attention in matrix form



$$Z = \text{softmax}\left(\frac{QK^\top}{\sqrt{D}}\right)V$$

- All N words are processed at once: one matrix product for the scores, one for the outputs
- Dividing by √D keeps the scores in a range where the softmax is not saturated (stable training)

Multi-head self-attention

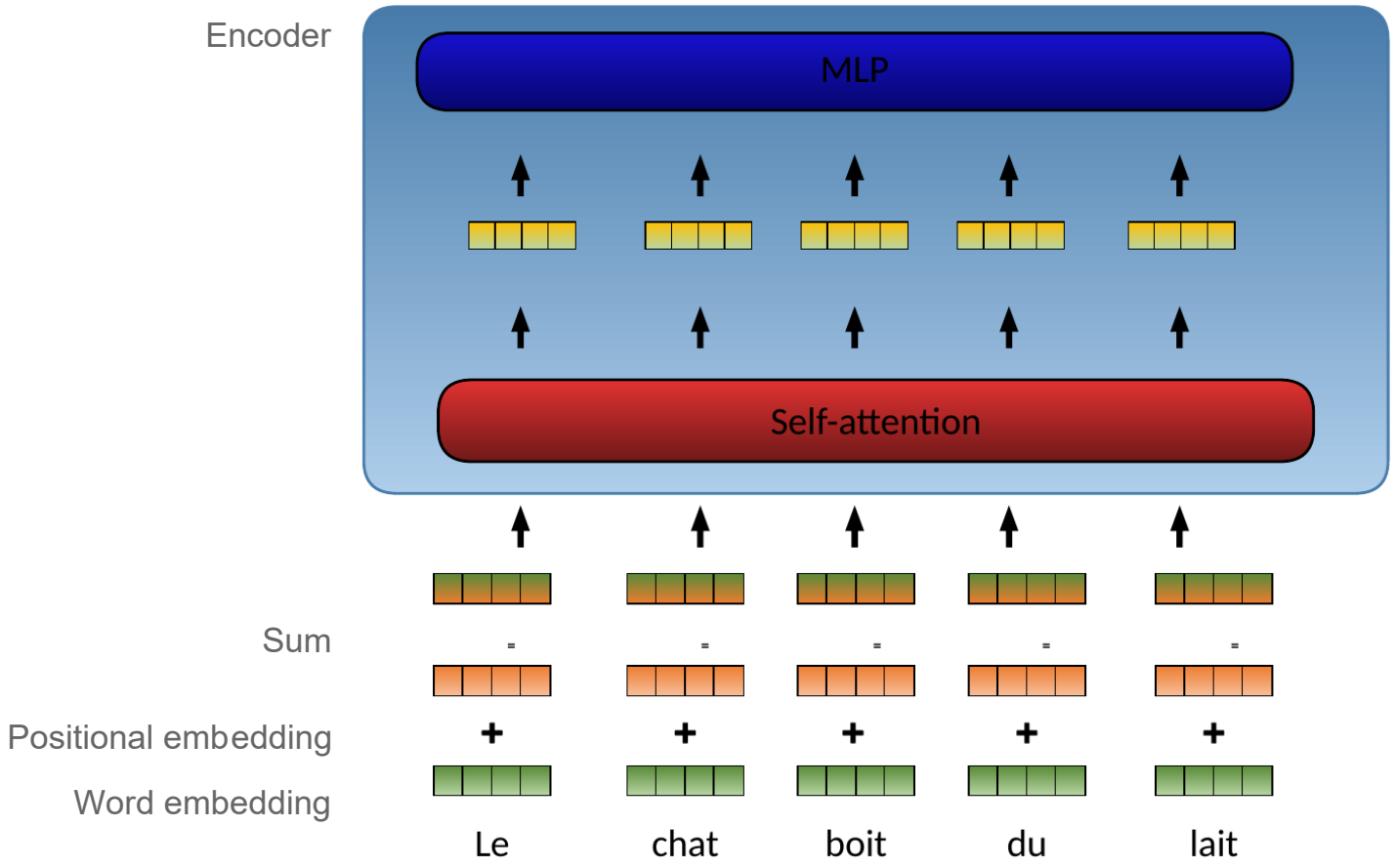


Left: two heads of the same layer, for the word "it"

- One head captures one notion of similarity
- Several heads: several notions at once (syntax, coreference, …)
- Each head h has its own W_q, W_k, W_v. The heads are concatenated and projected:

$$x'_j = [z_j^1 \; z_j^2 \; \cdots \; z_j^H] \, W_0$$

Word + positional embeddings



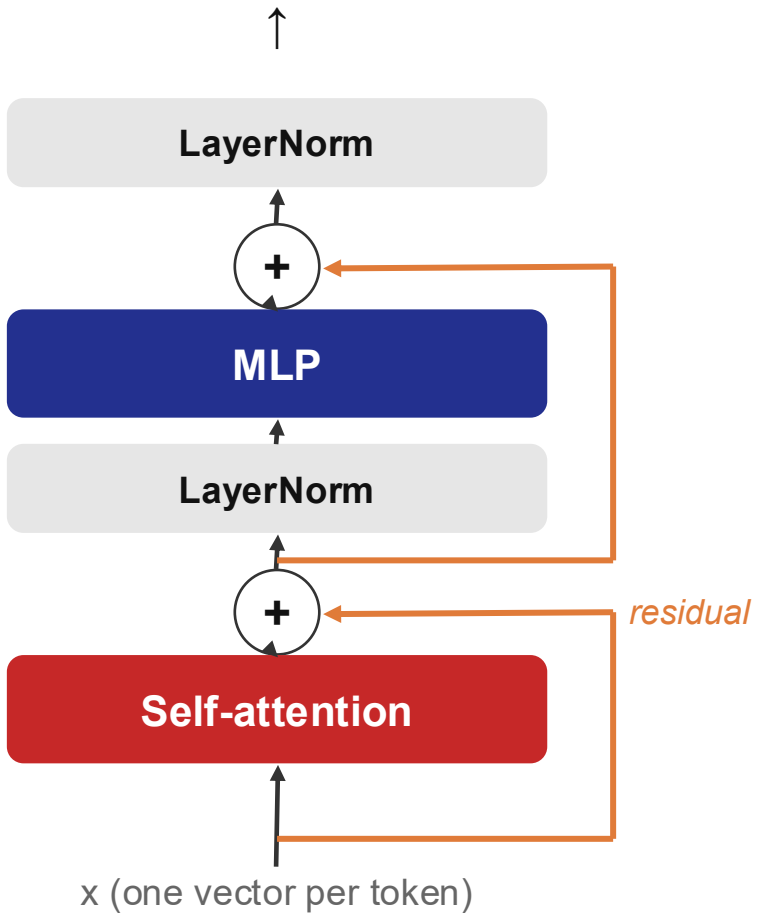
- Self-attention treats a sentence as a set: it ignores word order
- “cat eats plant” ≠ “plant eats cat”
- Remedy: add a vector that depends on the position (learned or sinusoidal)

Finishing the transformer

Residual connections, decoding and the full architecture

Residual connections and layer normalization

- **Residual connections**
  - Each sub-layer adds its output to its input: x + f(x)
  - Gradients flow directly through the “+”: deep stacks train well
  - Biases the block toward small corrections (“simple” functions)
- **Layer normalization**
  - Normalizes each token's features to mean 0, standard deviation 1
  - Stabilizes training



$$z_i = \mathrm{LayerNorm}(x_i + \mathrm{SelfAttention}(X)_i)$$

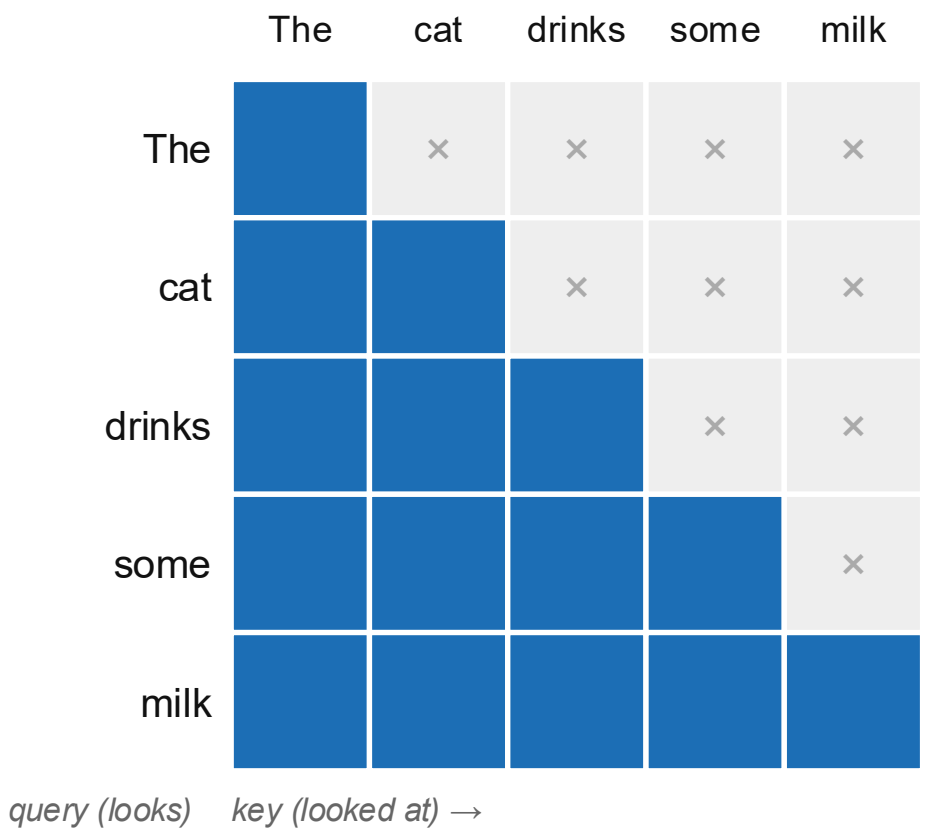
Decoding: predicting the next word

- The transformer transforms word representations
- To generate, add a softmax layer over the vocabulary on top of the last representation:

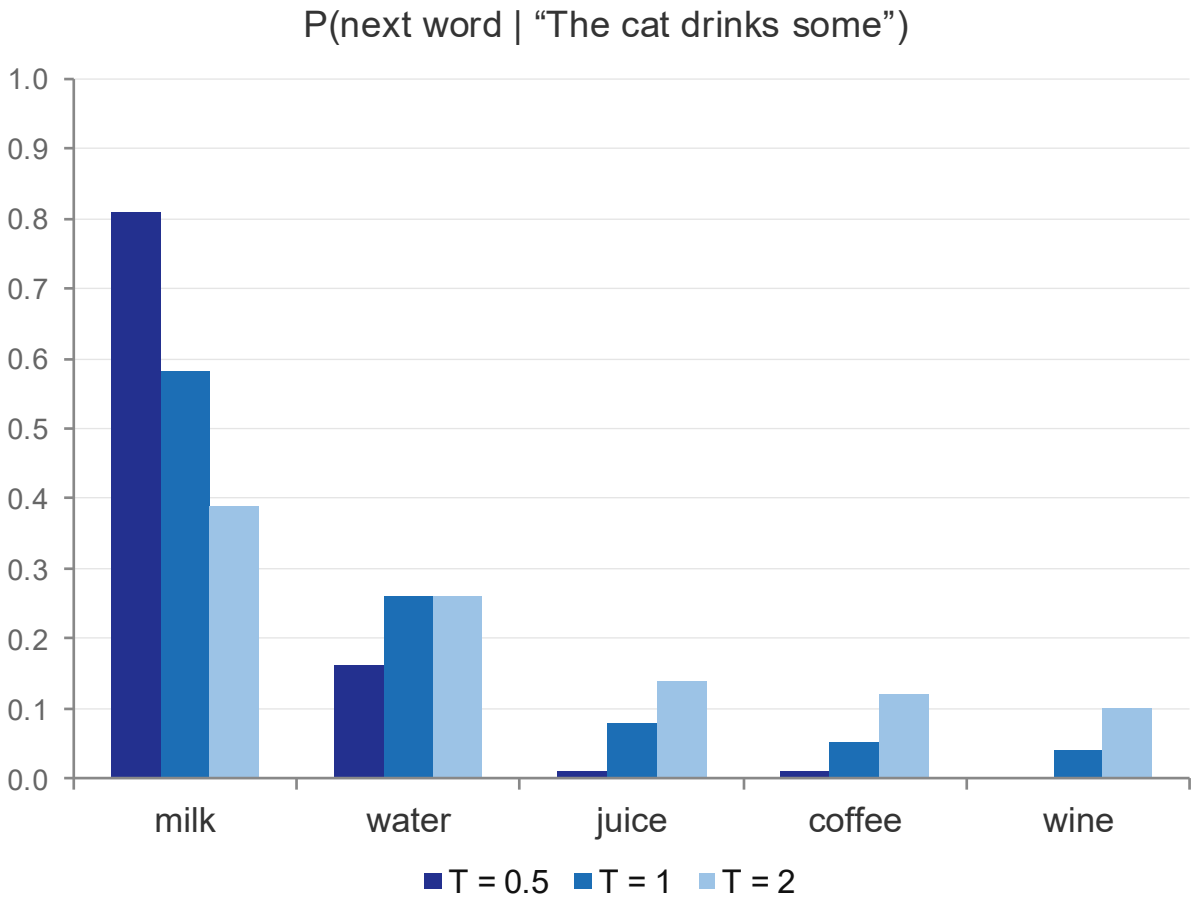
$$P(o_n \mid o_{1:n-1}) = \frac{\exp(g_w^\top x_{n-1})}{\sum_{w'=1}^{W} \exp(g_{w'}^\top x_{n-1})}$$

g_w: parameters of word w; W: vocabulary size

- A word may only attend to the previous words (causal mask)
- With rows = queries, the attention matrix is lower triangular



Choosing the next word: sampling

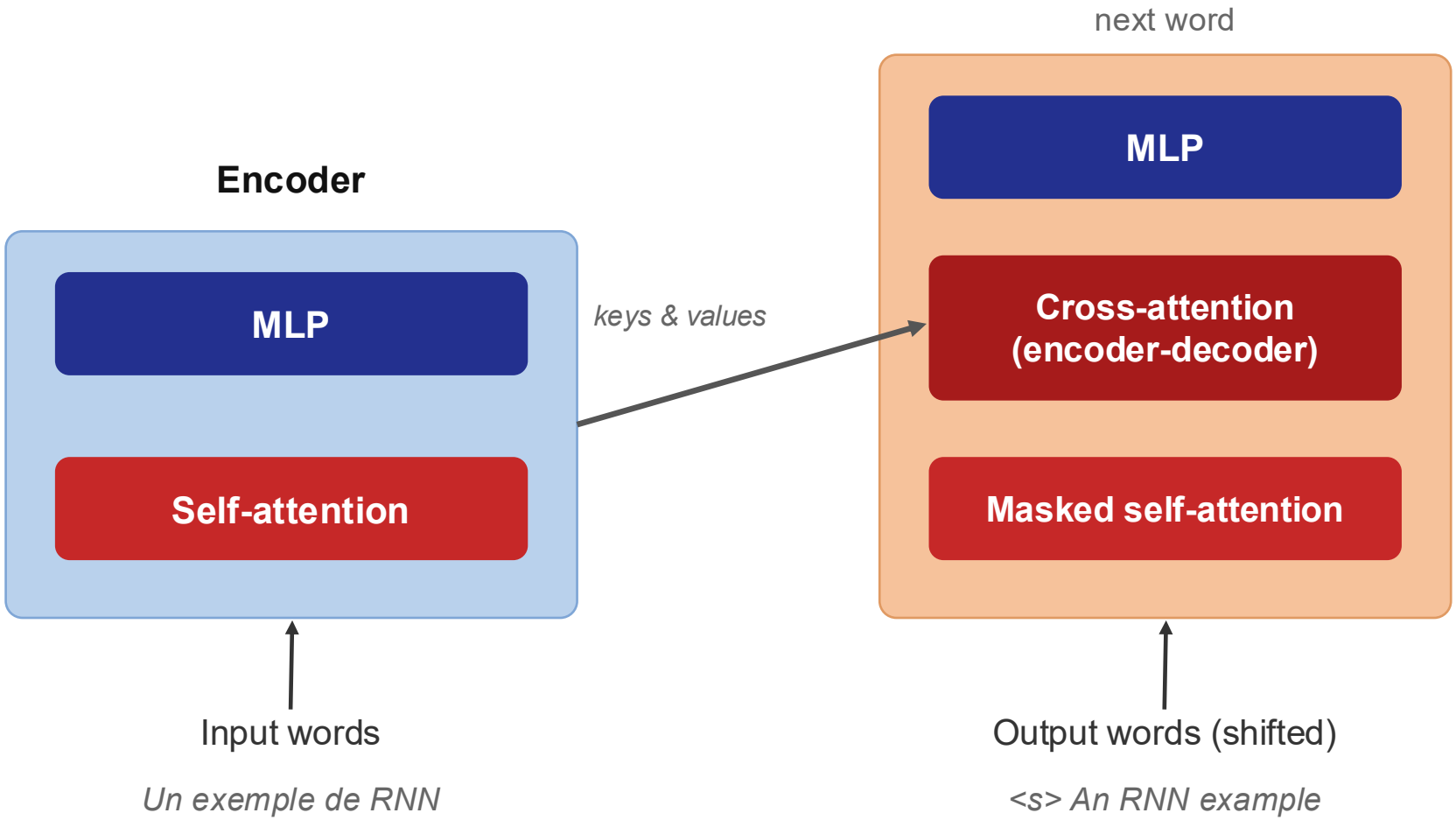


- Greedy: always take the most likely word
  - Deterministic, but often repetitive
- Sampling with a temperature T:

$$p_w = \frac{\exp(s_w/T)}{\sum_{w'} \exp(s_{w'}/T)}$$

  - T < 1: sharper (more predictable). T > 1: flatter (more varied)
- Top-k / top-p: sample only among the most likely words
- Chatbots sample: the same question can get different answers

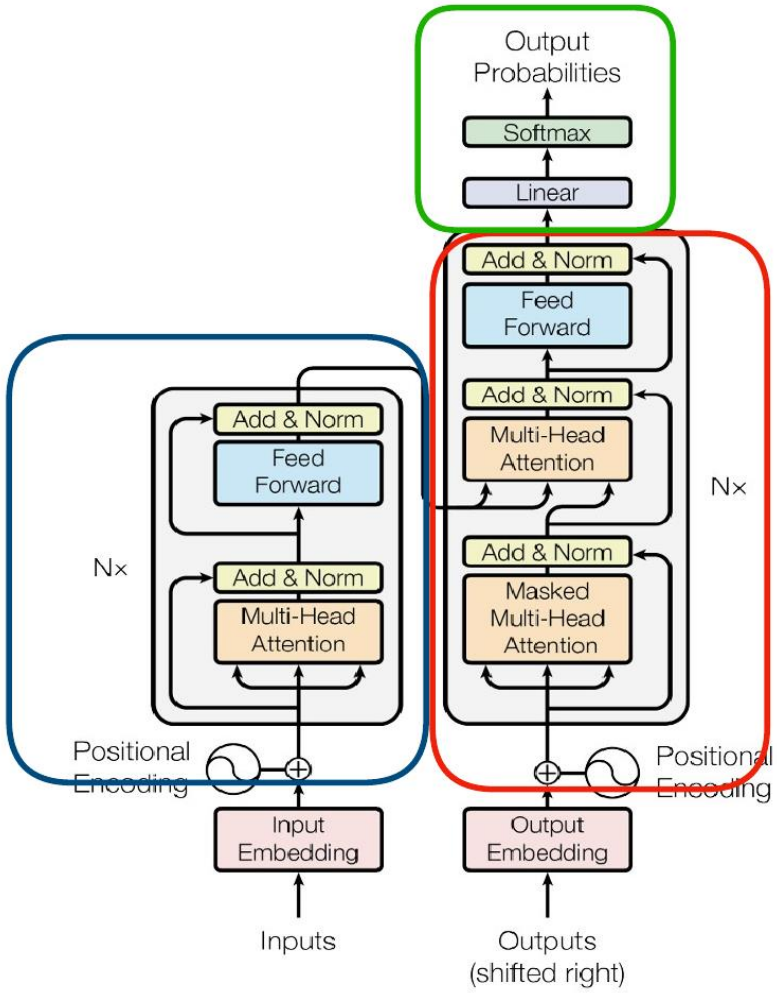
Encoder-decoder transformer



Decoder

- Encoder: reads the whole source sentence
- Decoder: masked self-attention over the words generated so far
- Cross-attention: queries from the decoder, keys and values from the encoder

Putting it all together



1. **Embed**
   Tokens → word embeddings + positional embeddings
2. **Encode**
   N blocks: self-attention + MLP, each with residual + LayerNorm
3. **Decode**
   N blocks: masked self-attention, cross-attention, MLP
4. **Predict**
   Linear + softmax → probability of each next word
5. **Repeat**
   Pick a word, append it to the output, start again

Figure: Vaswani et al. (2017). Animated walkthrough: jalammar.github.io/illustrated-transformer

Decoder-only models

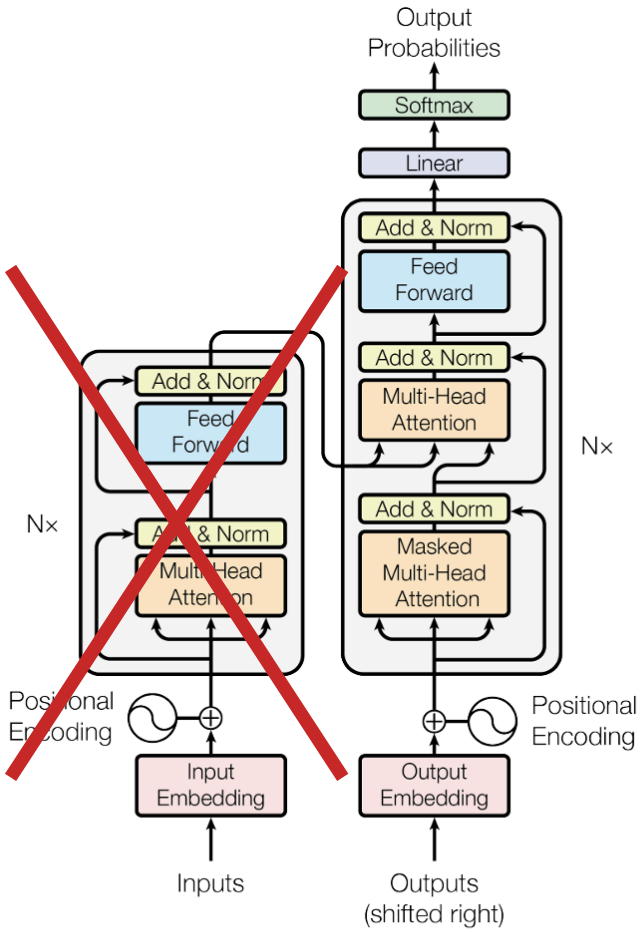


Figure 1: The Transformer - model architecture.

- Almost all of today's text LLMs are decoder-only (GPT, Claude, Gemini, Llama, Qwen, …)
- No encoder and no cross-attention: the prompt is simply the beginning of the sequence
- The masked decoder already reads all the previous words
- One architecture, one objective (predict the next token), which scales very well

Is attention free? Its cost

	RNN	Transformer
Training on a sequence	Sequential	Parallel ✓
Generating token n	O(1): only needs h_n−1 ✓	O(n): attends to all previous tokens
Generating n tokens	O(n) ✓	$O(n^2)$
Memory while generating	One hidden state ✓	Keys and values of every previous token

- Transformers trade cheap generation for fast, parallel training: the right trade for learning from huge datasets
- Long contexts are expensive in compute and memory. Much current research targets this cost.

One of many transformers

Encoder-only

Examples

BERT, RoBERTa

Typical uses

Classification, embeddings, search

Encoder-decoder

Examples

T5, BART

Typical uses

Translation, summarization

Decoder-only

Examples

GPT, Claude, Gemini, Llama, Qwen, DeepSeek, Mistral

Typical uses

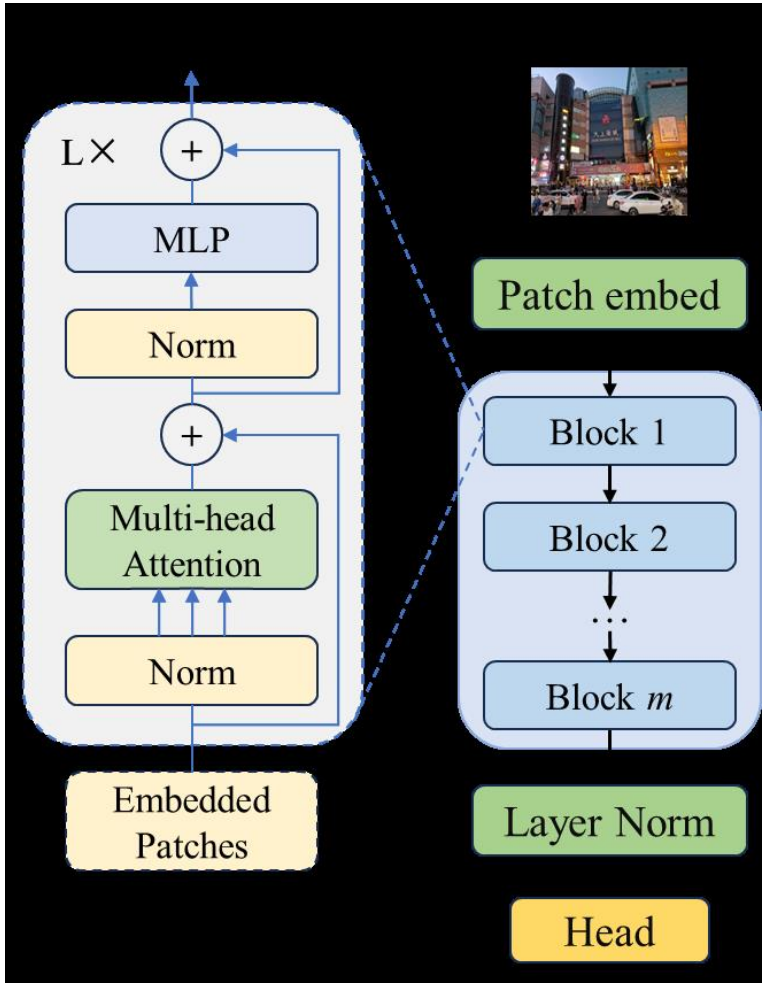
Generation, chat, code, agents

- These names refer to trained systems, not only to an architecture
- Size is counted in parameters: Llama-7B has 7 billion; frontier models are much larger

Partial list: huggingface.co/docs/transformers

Beyond text

- Images: cut the image into patches; each patch is a token (Vision Transformer)
- Same blocks: attention + MLP, with residuals and normalization
- Example: DINOv2 (Meta, 2023) learns visual features without labels
- Also audio, video, molecules and proteins, robotics, …
- Multimodal LLMs combine several of these token types

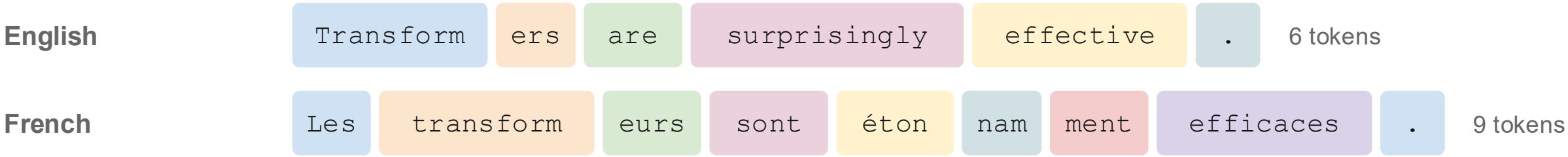


Dosovitskiy et al., An Image is Worth 16x16 Words (ICLR 2021); Oquab et al., DINOv2 (2023)

Training a language model

From now on: decoder-only transformers

From text to tokens



Actual split by the o200k tokenizer (GPT-4o). Most tokens include the space before the word. Other models split differently.

Byte-pair encoding (BPE)

- Start from characters (or bytes)
- Repeatedly merge the most frequent adjacent pair
- Stop at a vocabulary of ~50k–200k tokens
- Frequent words: one token. Rare words: several.

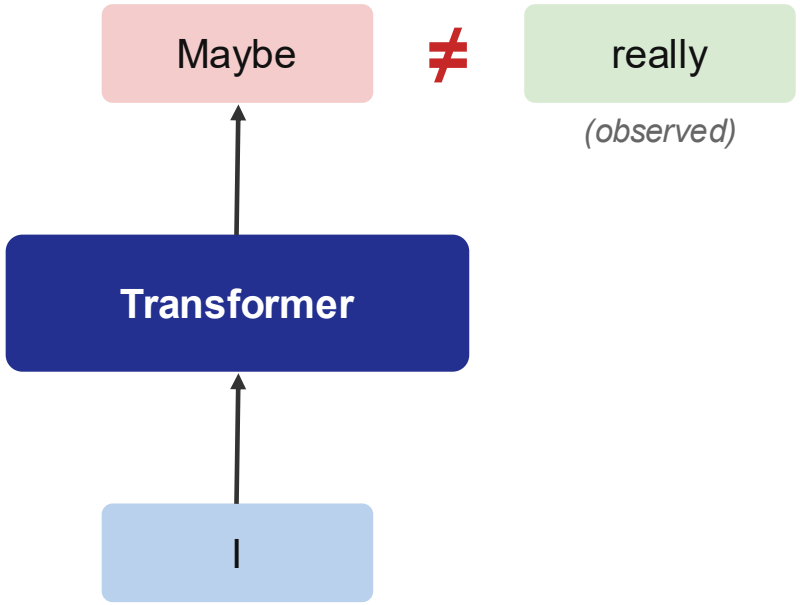
Why it matters in practice

- Context length and price are counted in tokens
- Other languages often need more tokens than English
- Explains some quirks: spelling, counting letters, arithmetic

Training: the naive way

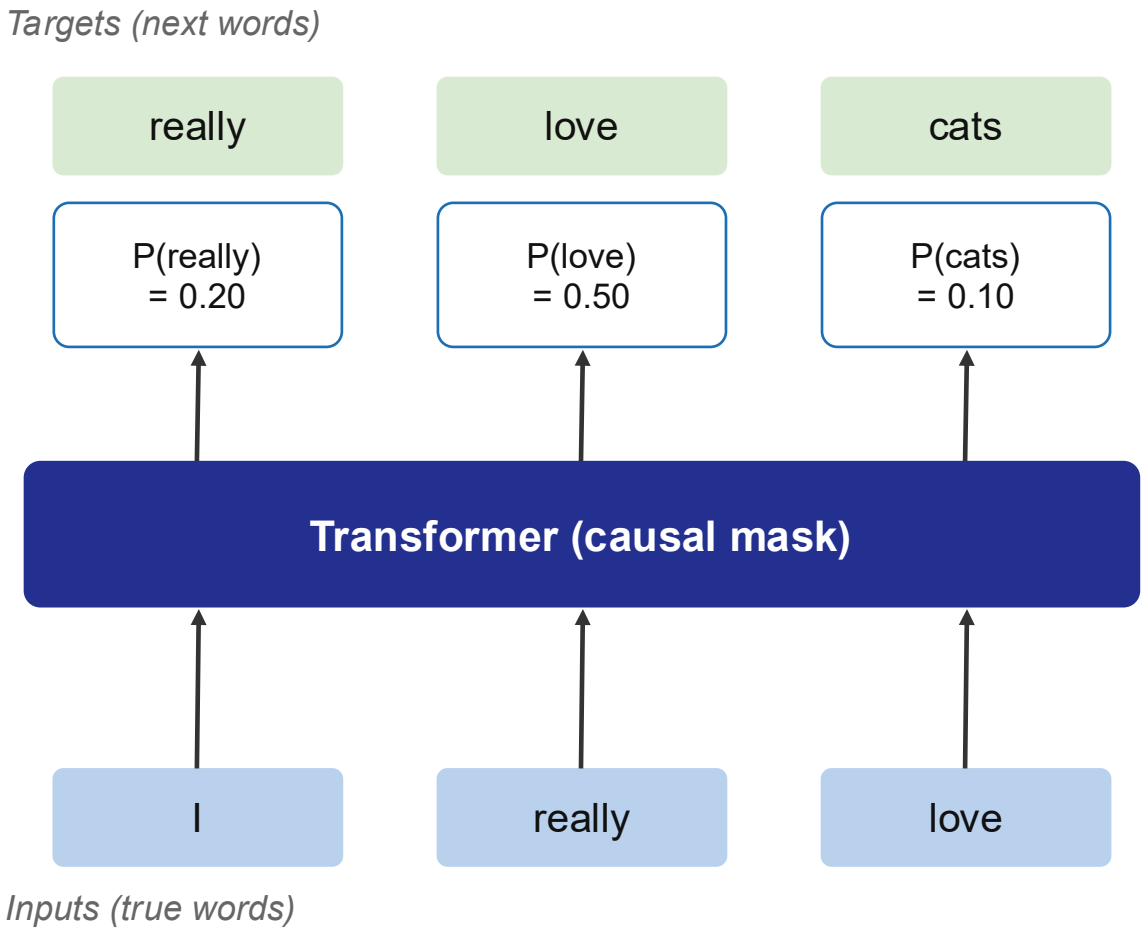
Training sentence: I | really | love | cats

(sampled)



- Generate a word with the model and compare it to the data?
  - Sampling is noisy and gives a 0/1 signal
  - One word at a time: slow
- But we already know the right answer…
  - Directly measure how likely the model was to produce it, and push that probability up

Teacher forcing



- Feed the true previous words, not the model's own predictions
- Thanks to the causal mask, every position is trained in parallel, in a single pass
- At each position: increase the probability of the observed next word
- One sentence = many training examples

The loss: negative log-likelihood

Maximize the probability of the training text, i.e. minimize:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{T} \log P_\theta(w_t \mid w_{<t})$$

Position	P(w_t | w_<t)	−log P
`really	I`	0.20	1.61
`love	I really`	0.50	0.69
`cats	I really love`	0.10	2.30
Mean		**1.54**

- Same as the cross-entropy between the predicted distribution and the observed word
- Small probability on the right word → large loss
- Averaged over trillions of tokens in practice

Evaluating a language model: perplexity

$$\mathrm{PPL} = \exp\left(-\frac{1}{T}\sum_{t=1}^{T}\log P_\theta(w_t \mid w_{<t})\right)$$

1

perfect model: always certain and right

≈ 4.6

our example (mean loss 1.54): like choosing among ~4.6 words

50 000

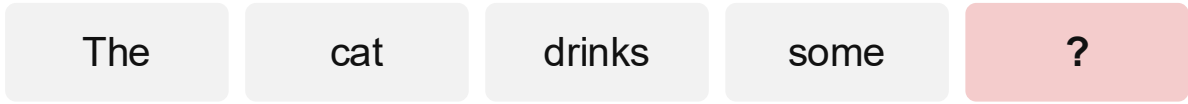
uniform guess over a 50k-token vocabulary

- Exponential of the average loss, computed on held-out text. Lower is better.
- Intuition: the number of words the model is effectively hesitating between
- Only comparable between models that use the same tokenizer

Two classic pre-training objectives

Next-token prediction

Decoder-only models: GPT, Llama, Claude, …



Masked-token prediction

Encoder-only models: BERT, RoBERTa



- Both need no labels: the text is its own supervision (self-supervised learning)
- Next-token prediction can generate text directly, which is why it dominates for LLMs

From a language model to an assistant

Pre-training, mid-training, post-training

Three stages of training



- Web-scale text and code
- Next-token prediction
- General knowledge and language
- By far the most compute

- Curated, higher-quality data
- More code, math, reasoning
- Longer documents → longer context

- Instructions and conversations
- Human preferences (RLHF)
- Verifiable rewards (RLVR)
- Turns it into an assistant

The boundaries between stages are fuzzy, and companies rarely publish all the details.

Pre-training

- Next-token prediction with teacher forcing, as we just saw
- Data: a large share of the public web, books, code
- Goal: bake general knowledge and language into the weights
- Frontier runs take weeks to months on thousands of accelerators
- **Important: pre-training alone does not produce a useful chatbot**

~15 trillion

tokens used to pre-train Llama 3 (Meta, 2024)

Roughly 15,000 lifetimes of reading (about 1 billion words each)

Meta, The Llama 3 Herd of Models (2024); Epoch AI, longest training runs (epoch.ai/data-insights/longest-training-run)

Mid-training

Keep training with the same next-token objective, but on a different mixture of data, toward the end of pre-training.

1 Better data

Filtered, higher-quality text. More code, math and scientific content.

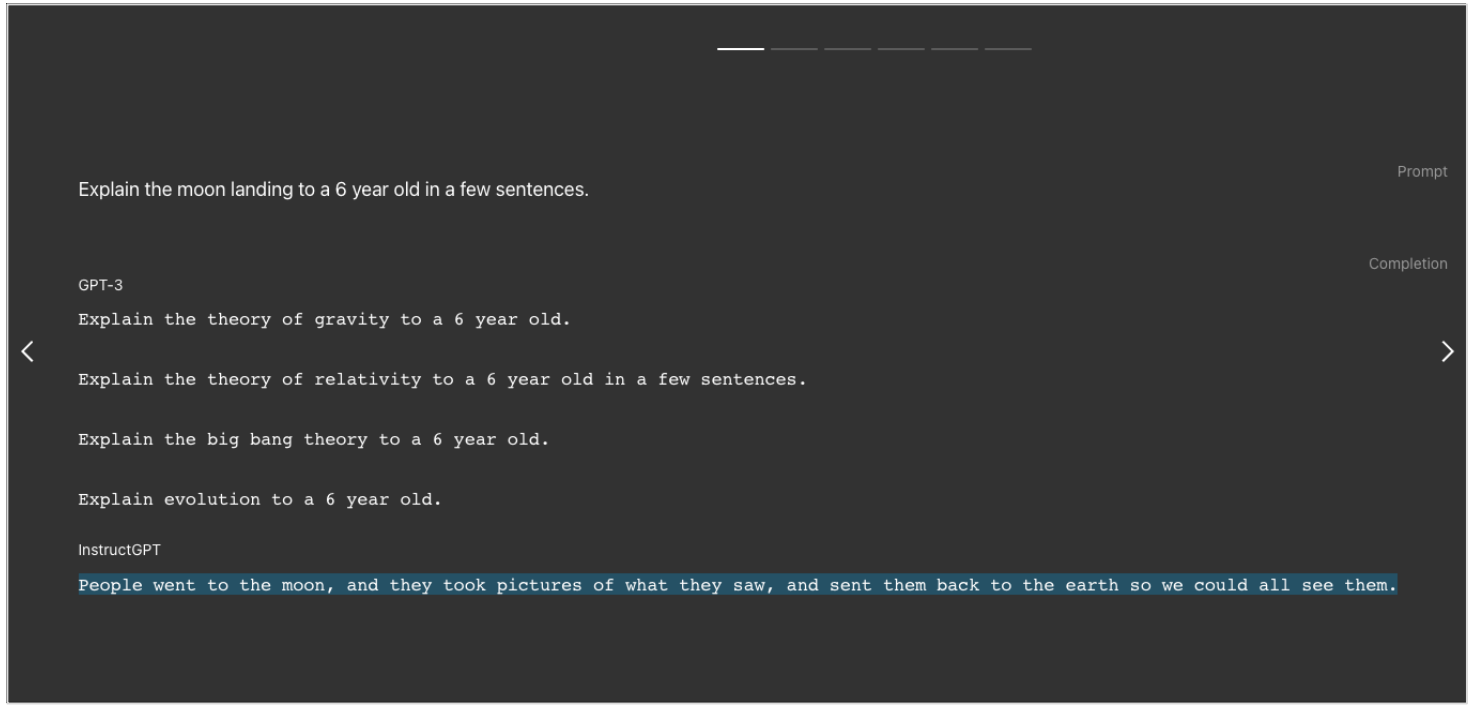
2 Longer context

Long documents teach the model to use contexts of hundreds of thousands of tokens.

3 Preparing post-training

Examples of reasoning, conversations and instructions, so the next stage starts from a better place.

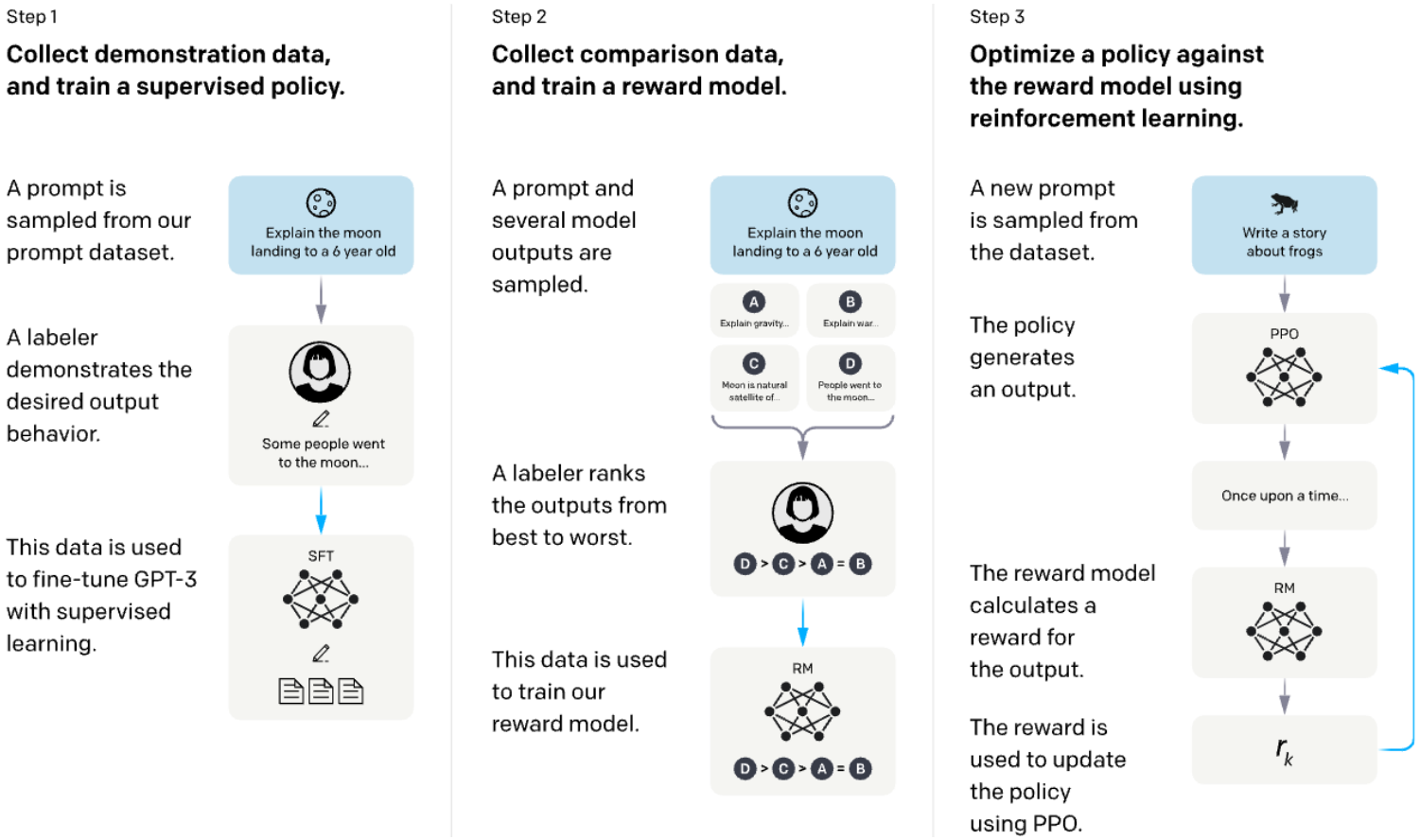
What comes out of pre-training?



- A very good autocomplete
- Asked to explain the moon landing to a 6-year-old, GPT-3 continues the pattern: it writes more similar requests
- The post-trained model (InstructGPT) answers the question

OpenAI, Aligning language models to follow instructions (2022)

How do we post-train? The InstructGPT recipe



- **1. Instruction tuning**
  - Supervised: learn from demonstrations
- **2–3. RL from human feedback**
  - Learn a reward from human preferences, then optimize it

Fun fact: the last author, Ryan Lowe, did his PhD at McGill with Joelle Pineau.

Ouyang et al., Training language models to follow instructions with human feedback (NeurIPS 2022)

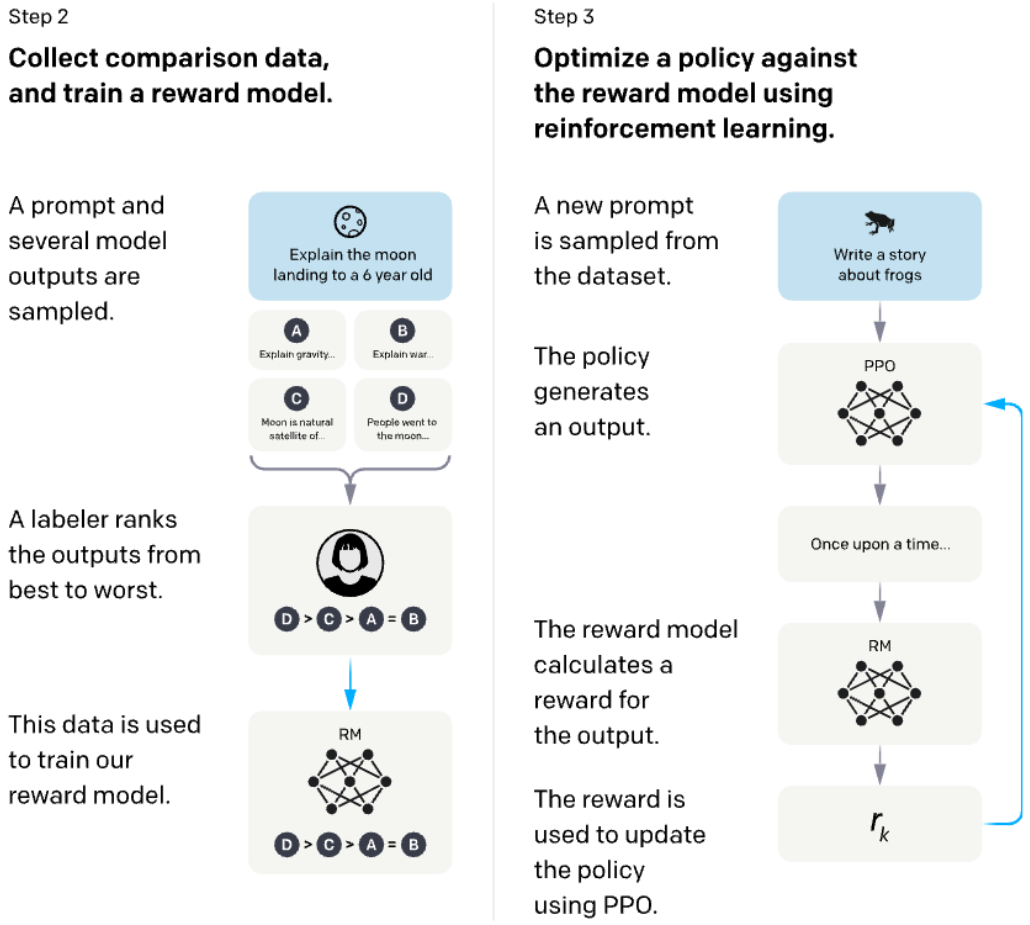
Step 1: instruction tuning

```
<|system|> You are a helpful assistant.
<|user|> What is the capital of Québec?
<|assistant|> Québec City.
<|user|> And its population?
<|assistant|> About 550,000 people.
```

Special tokens mark who is speaking. The loss is computed only on the assistant's tokens (orange).

- Same objective as pre-training: next-token prediction with teacher forcing
- Different data: curated conversations and demonstrations of good answers
- Much smaller dataset than pre-training
- Teaches format and behavior more than knowledge

Steps 2–3: RL from human feedback (RLHF)



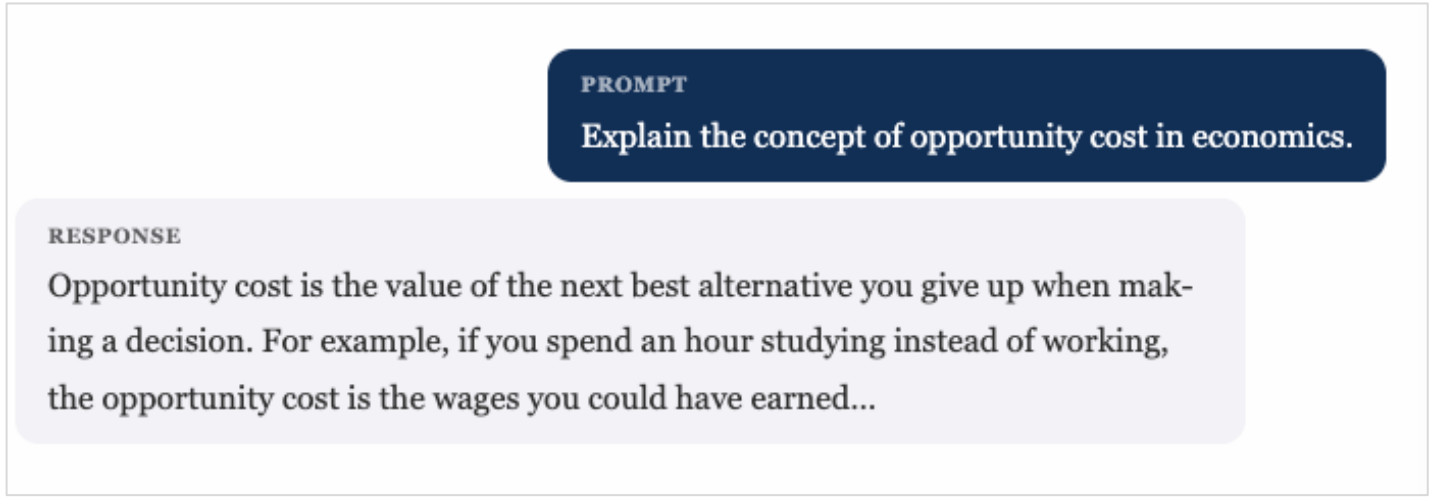
- Humans compare answers: which one is better?
- A reward model r(x, y) learns to predict these preferences
- Then optimize the LLM to get high rewards, without drifting too far from the instruction-tuned model:

$$\max_{\theta} \; \mathbf{E}_{y \sim \pi_\theta(\cdot \mid x)}[r(x, y)] - \beta \, \mathrm{KL}(\pi_\theta \,\|\, \pi_{\mathrm{SFT}})$$

The KL term keeps the model close to π_SFT and limits "gaming" the reward model.

Ouyang et al. (2022)

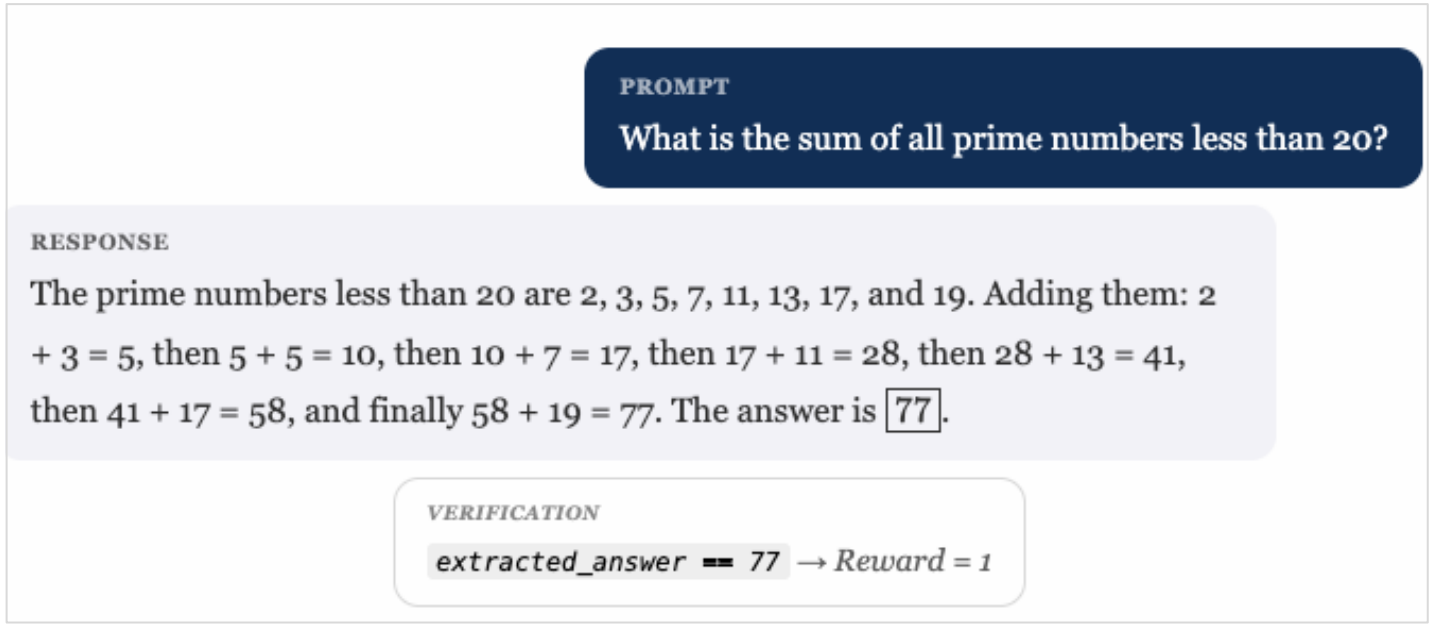
RLHF: when quality is a judgment call



- No single correct answer to “Explain opportunity cost”
- Quality = clarity, accuracy, completeness, helpfulness
- Easy for people to compare two answers, hard to write a rule
- RLHF learns the rule from comparisons

N. Lambert, RLHF Book (rlhfbook.com)

RL from verifiable rewards (RLVR)



- Same idea as RLHF, but the reward comes from a verifier instead of a learned model
  - Math: is the final answer correct?
  - Code: do the tests pass?
  - Puzzles, games, …
- Harder to fool than a learned reward, though not impossible (“reward hacking”)

N. Lambert, RLHF Book (rlhfbook.com)

Reasoning models

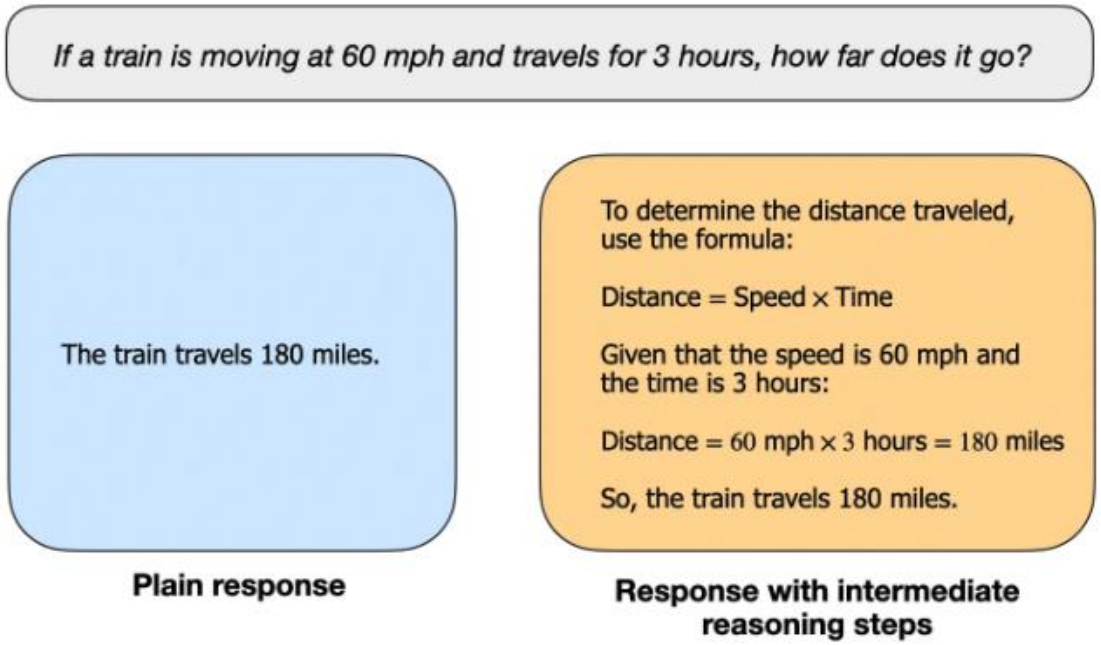
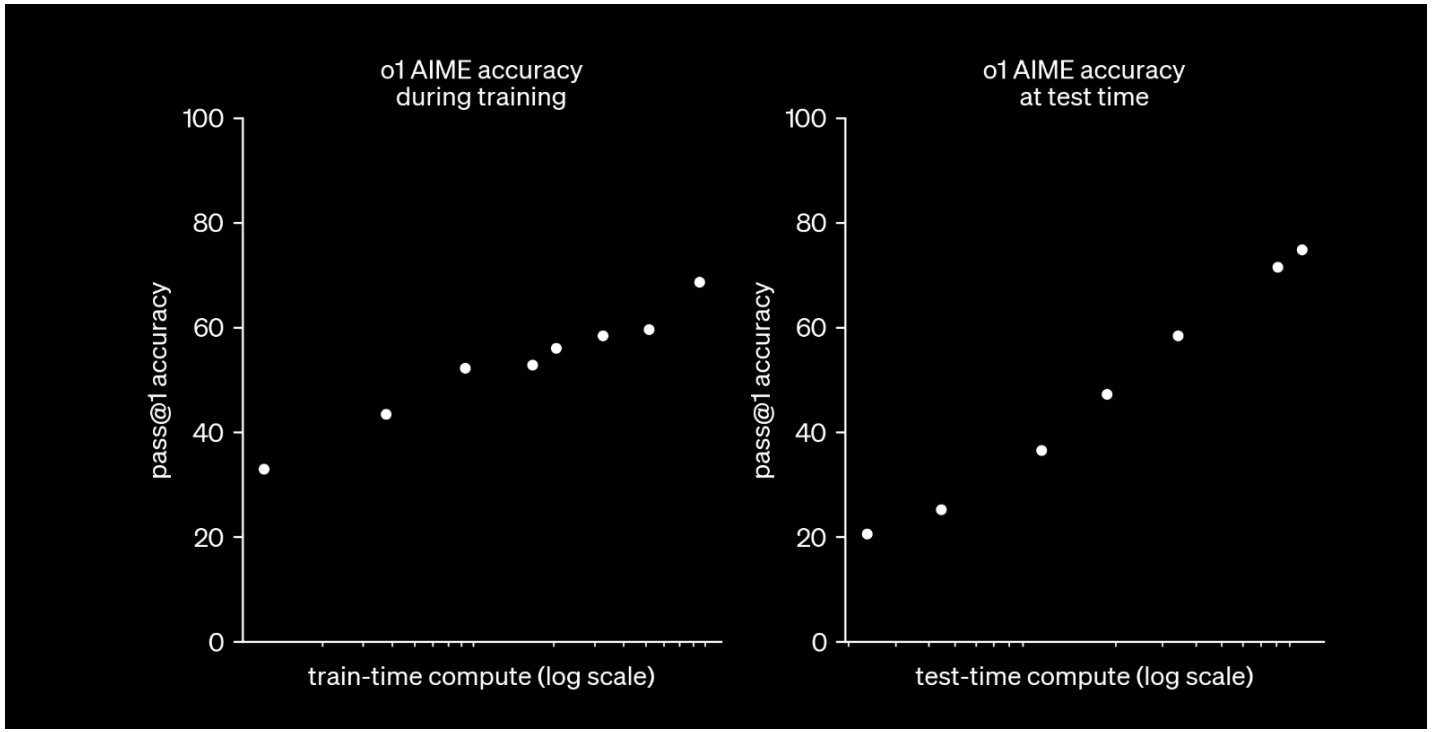


Figure 2: A regular LLM may only provide a short answer (as shown on the left), whereas reasoning models typically include intermediate steps that reveal part of the thought process. (Note that many LLMs who have not been specifically developed for reasoning tasks can also provide intermediate reasoning steps in their answers.

- RLVR teaches models to write long reasoning traces before answering
- Called “reasoning” or “thinking” models: OpenAI o1 (2024), then DeepSeek-R1 and most current frontier models
- Nobody writes these traces by hand: the model discovers them because they lead to correct answers

S. Raschka, Understanding Reasoning LLMs (2025)

A new axis: test-time compute

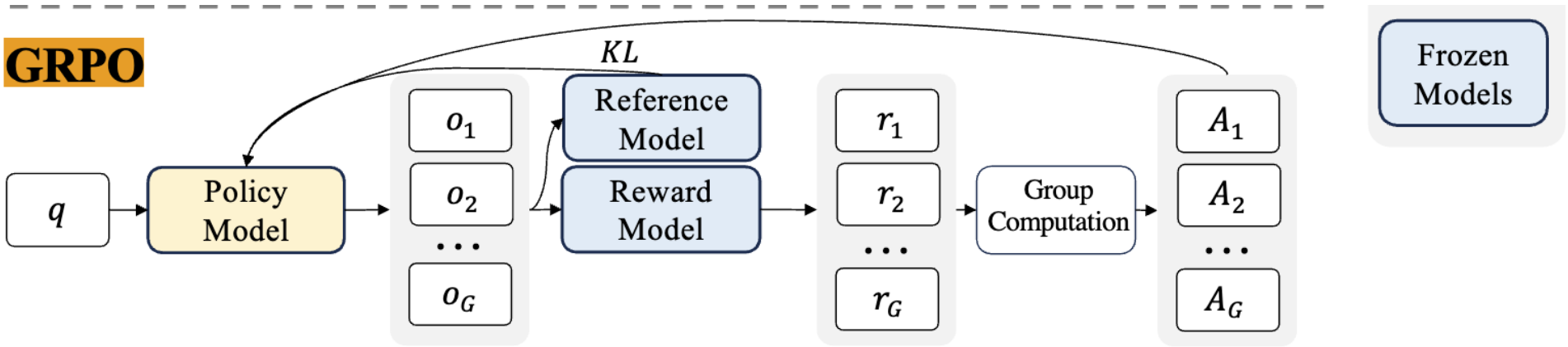


- Left: more RL training compute → better accuracy
- Right: more “thinking” at test time → better accuracy
- A second way to scale: spend more compute per question
- Trade-off: slower and more expensive answers

OpenAI, Learning to reason with LLMs (o1, 2024). Accuracy on the AIME math competition; x-axes on log scales.

DeepSeek-R1 and GRPO

- January 2025: an open-weight model that matches o1 on many reasoning benchmarks
- Recipe: RLVR with a simple algorithm, GRPO



For each question, sample a group of G answers and score them. Each answer is rewarded relative to its group:

$$A_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G)}$$

No separate value network: simpler and cheaper than PPO.

DeepSeek-AI, DeepSeek-R1 (2025); Shao et al., DeepSeekMath (2024)

Post-training: summary

1 Instruction tuning

Format and behavior: from autocomplete to assistant

2 RLHF

Human preferences, for open-ended answers

3 RLVR

Verifiable tasks (math, code), and long reasoning

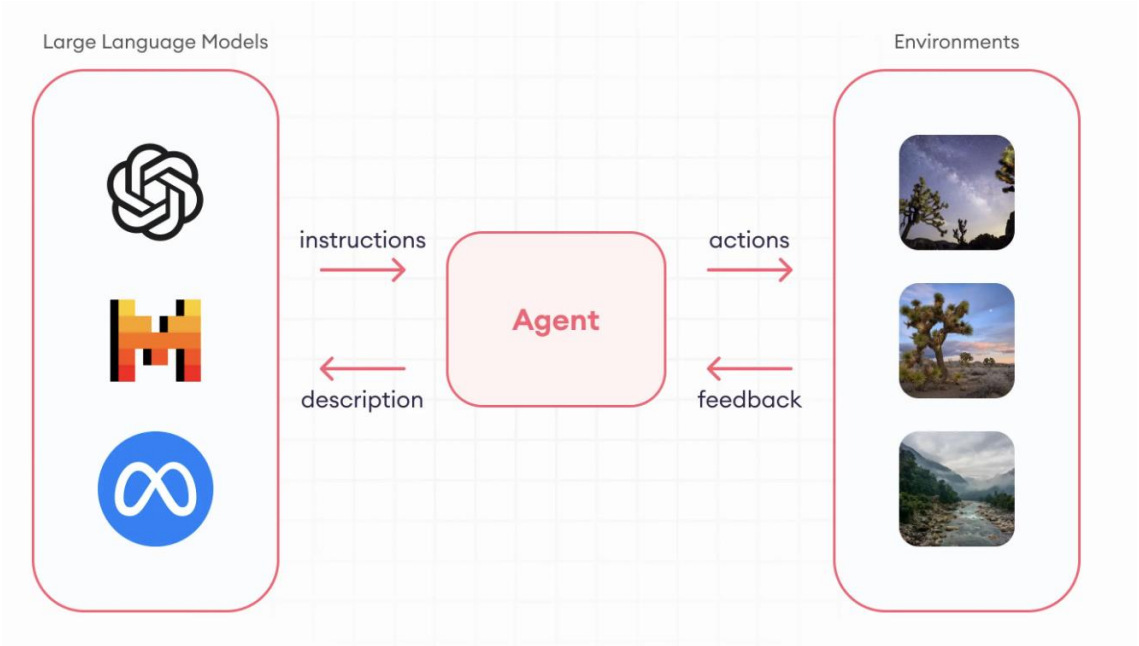
- Over the past few years, post-training has become a central part of building an LLM
- It opened new axes of scaling: RL compute and test-time compute

LLM agents

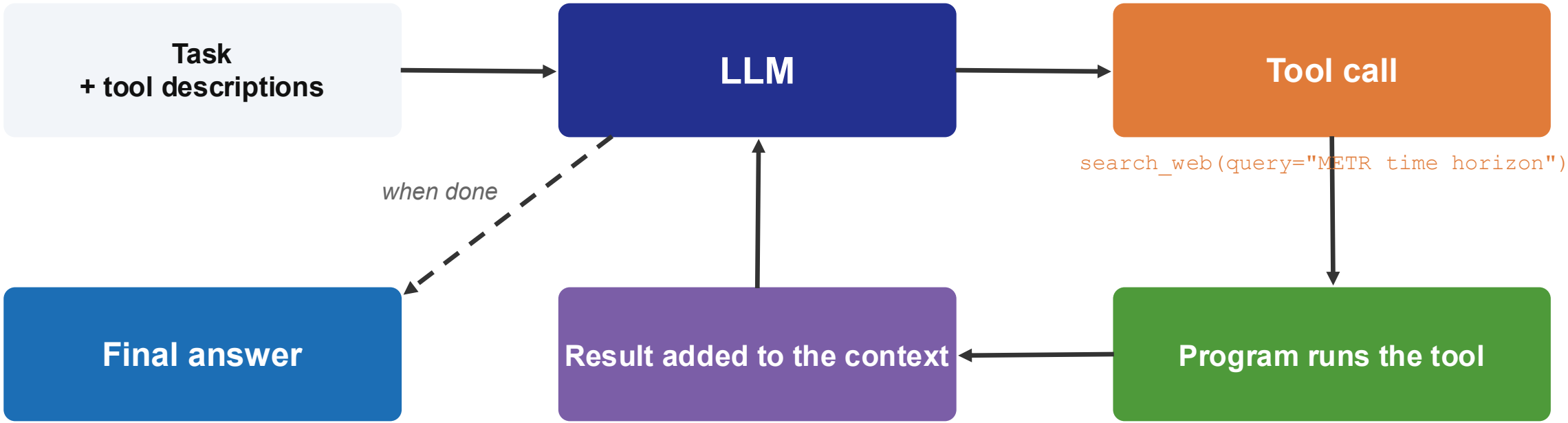
How do ChatGPT, Claude and Gemini actually work?

What is an LLM agent?

- No single standard definition
- Generally: an LLM that acts in an environment through tools, over several steps
- Tools: web search, code execution, files, databases, memory, …
- Adoption is growing quickly, especially in software development



How does it work? The agent loop



- The LLM only produces text; a tool call is text in an agreed format (e.g., JSON)
- A regular program (the “harness”) executes it and feeds the result back
- Repeat until the task is done: this is what happens when a chatbot searches the web or runs code

Examples of environments and tools

Software development

Environment: a code repository and a terminal. Tools: read and edit files, run code and tests.

Spreadsheets and data

Environment: your files or database. Tools: spreadsheet functions, SQL queries.

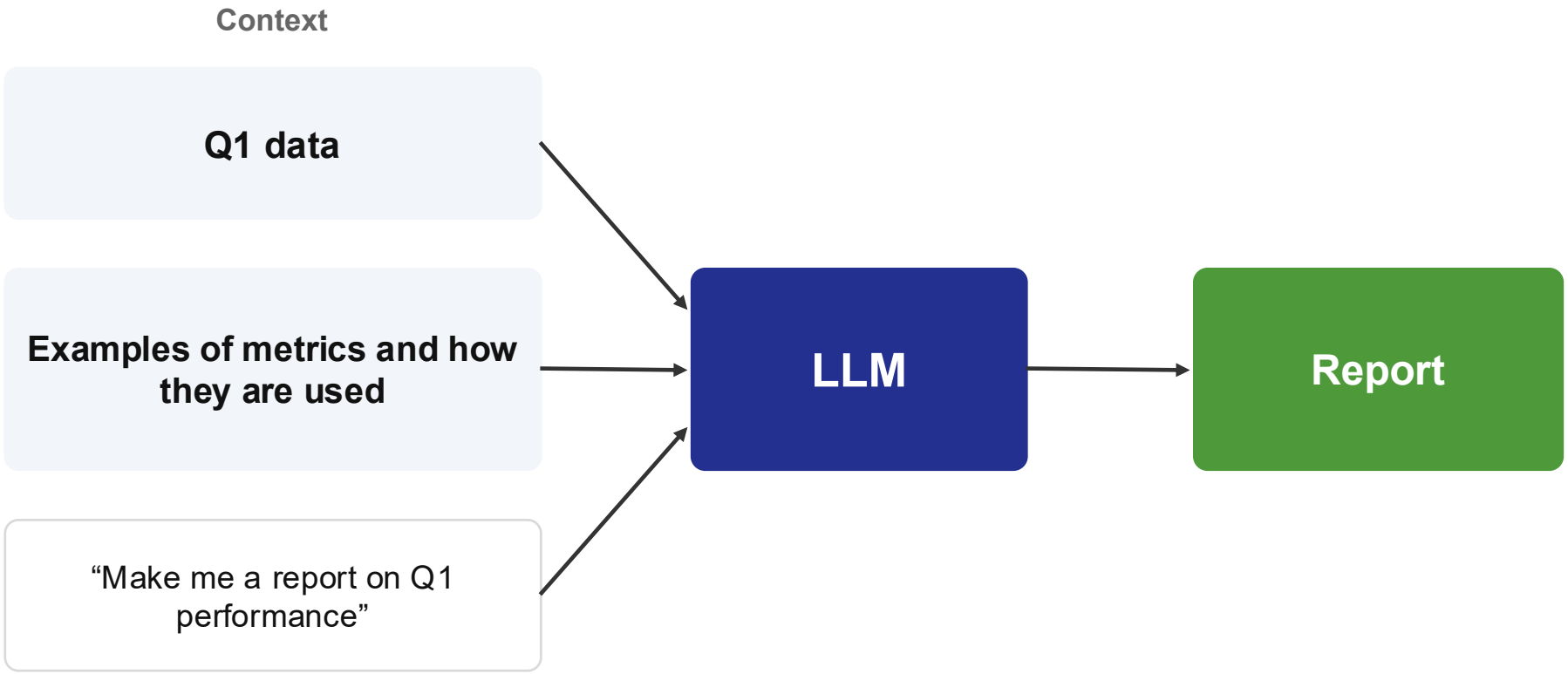
Information retrieval

Environment: the web. Tools: search engine, fetch a page.

Customer service

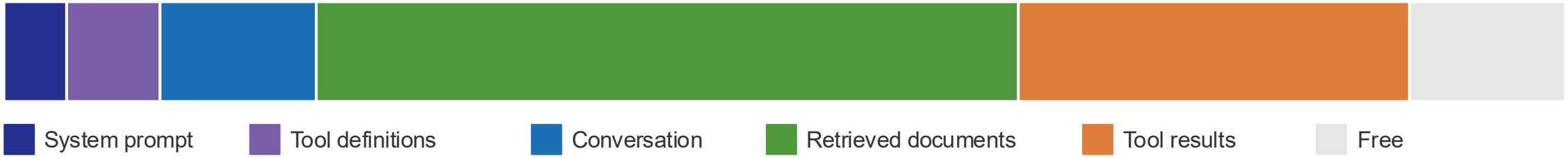
Environment: the company's systems. Tools: the functions human agents use, e.g. give_discount().

Agents do better with good context



- Putting the right information in the context is one of the biggest levers
- In-context learning: the model adapts to examples and instructions without any weight update

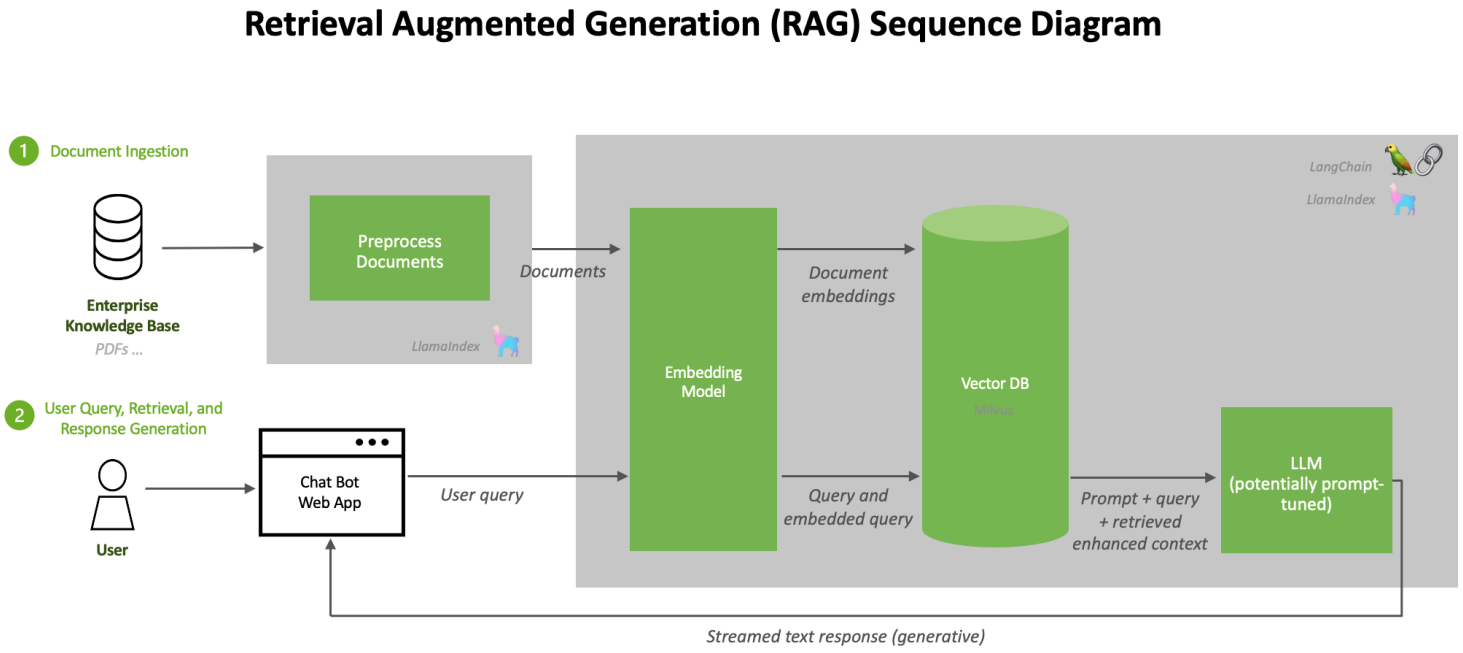
The context window



Illustrative composition of an agent's context

- Everything the model "sees" at once, counted in tokens
- Frontier models: from hundreds of thousands up to around a million tokens
- You pay per token, for input and for output
- A longer context is slower and more expensive, and quality can degrade
- Choosing what goes in is called context engineering

Retrieval-augmented generation (RAG)



- Automates the choice of context
- A retriever finds the documents most relevant to the question (often with embeddings)
- They are added to the prompt before the LLM answers
- Answers can cite their sources

NVIDIA Technical Blog, RAG 101

Limitations

1 Transient memory

- Knowledge is in the weights (fixed) or in the context (temporary)
- Without an explicit memory tool, nothing carries over between conversations
- Learning continually into the weights is an open problem

2 Long contexts

- Quality usually degrades as the context grows
- Cost of attention grows quadratically with length
- Active area of research

3 Verification

- LLMs still make mistakes, stated with confidence
- Outputs (facts, citations, code) must be checked
- The more autonomous the agent, the more this matters

Verification: an example

GPTZero finds 100 new hallucinations in NeurIPS 2025 accepted papers

- Fabricated references made it into accepted NeurIPS 2025 papers
- Missed by authors and by reviewers
- LLM output is a draft: verify facts, citations and code before relying on them

GPTZero

Agents: summary

1 Powerful tools

An LLM in a loop with tools can carry out multi-step tasks

2 Context is key

Careful curation or RAG brings large gains

3 Verify

Checking agent outputs remains essential

Looking ahead

Open models, security, and where things are heading

Open vs. closed models

Closed models

- Claude (Anthropic), GPT (OpenAI), Gemini (Google)
- Used through an API: pay per token
- Your data is sent to the provider
- Usually the most capable

Open-weight models

- Llama (Meta), Qwen (Alibaba), DeepSeek, Mistral, …
- Can be run on your own hardware
- You keep control of your data
- Have generally trailed the best closed models, though the gap has narrowed

The choice balances performance, cost and data confidentiality.

Security

! Jailbreaks

Carefully crafted inputs can get a model to ignore its safeguards.

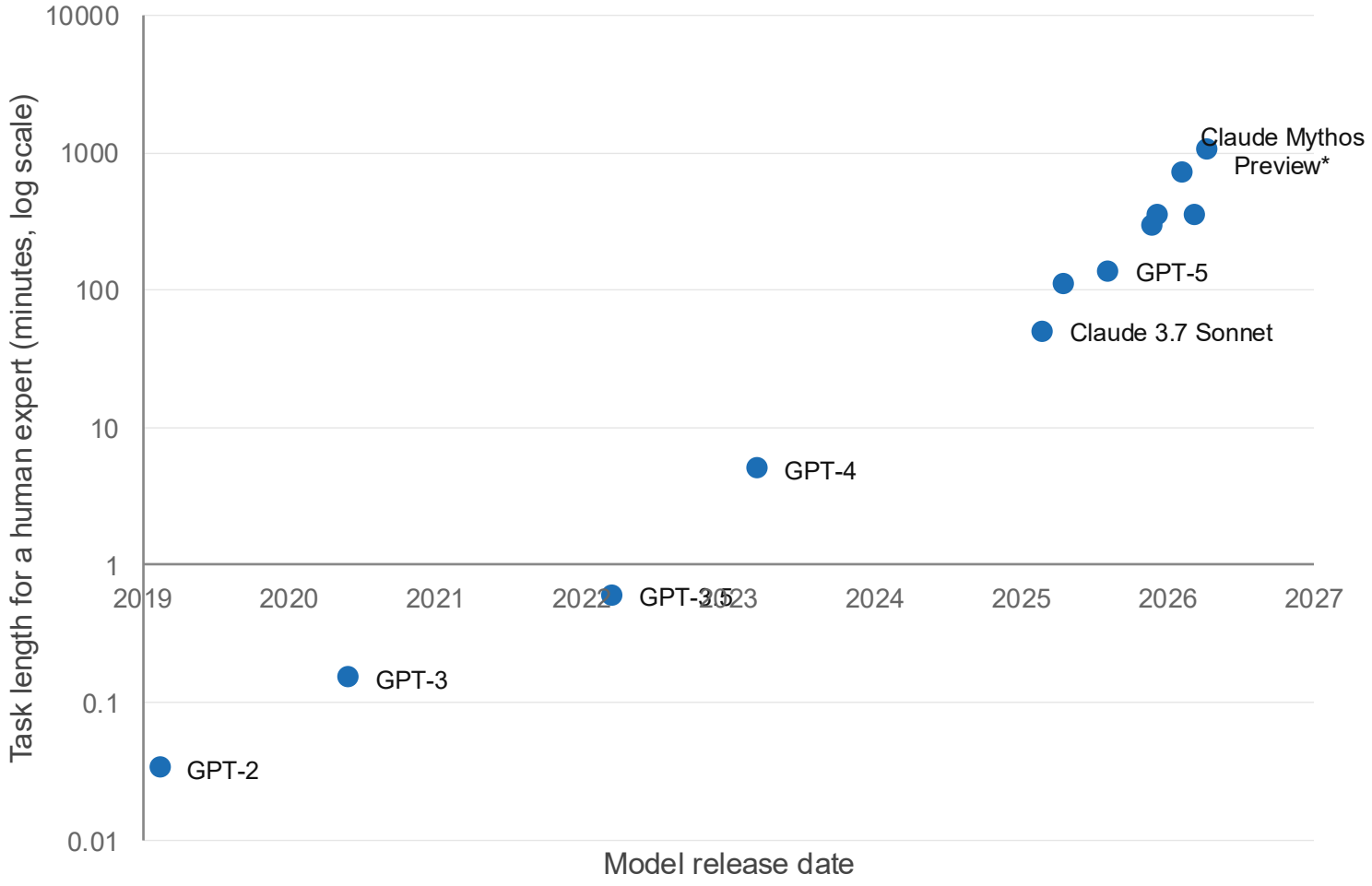
⚠ Untrusted content

Agents read web pages, e-mails and files written by others, and then take actions.

✓ Mitigations

Limit permissions, ask for confirmation, and monitor outputs, actions and reasoning.

Looking ahead: longer and longer tasks



- METR measures the length of software tasks (in human time) that AI agents complete with 50% success
- Doubled about every 7 months from 2019 to 2024, faster since
- Caveats: well-specified software tasks; estimates above 16 h are unreliable
- Implications: automation of tasks and jobs, security, governance. Will the trend continue?

*Source: METR, Task-Completion Time Horizons of Frontier AI Models (TH 1.0 and 1.1; page no longer updated as of Sept. 2026). *Early version, above the reliable range.*

Summary

Architecture	Self-attention + MLP blocks with residuals and normalization; today mostly decoder-only
Training	Next-token prediction with teacher forcing; evaluated with perplexity
From LM to assistant	Pre-, mid- and post-training (instructions, RLHF, RLVR, reasoning)
Agents	An LLM in a loop with tools; context and verification are key

References

- Bahdanau, Cho & Bengio, Neural Machine Translation by Jointly Learning to Align and Translate, 2014. arxiv.org/abs/1409.0473
- Vaswani et al., Attention Is All You Need, 2017. arxiv.org/abs/1706.03762
- J. Alammar, The Illustrated Transformer. jalammar.github.io/illustrated-transformer
- R. E. Turner, An Introduction to Transformers, 2024. arxiv.org/abs/2304.10557
- Ouyang et al., Training language models to follow instructions with human feedback, 2022. arxiv.org/abs/2203.02155
- N. Lambert, Reinforcement Learning from Human Feedback (book). rlhfbook.com
- DeepSeek-AI, DeepSeek-R1, 2025. arxiv.org/abs/2501.12948
- S. Raschka, Understanding Reasoning LLMs, 2025. magazine.sebastianraschka.com
- Kwa et al. (METR), Measuring AI Ability to Complete Long Tasks, 2025. arxiv.org/abs/2503.14499