NORSK REGNESENTRAL
NORWEGIAN COMPUTING CENTER

Large Language Models

Yves Scherrer & Pierre Lison

IN4080, autumn 2025, lecture 9

Oblig 3

- The third obligatory assignment is about building a RAG system that can answer Star Wars questions
- Submission deadline: **October 17**
- The best way to do this assignment is on **Fox**, the HPC cluster from UiO
  - Do register ASAP if you haven't yet done so!
  - If you have problems registering to Fox or running code on it, you can also use on your own computer



Outline

- Quick recap on the Transformer
- Training LLMs
- Model alignment
- Prompting LLMs
- Generation
- Retrieval-Augmented Generation (RAG)
- Open challenges

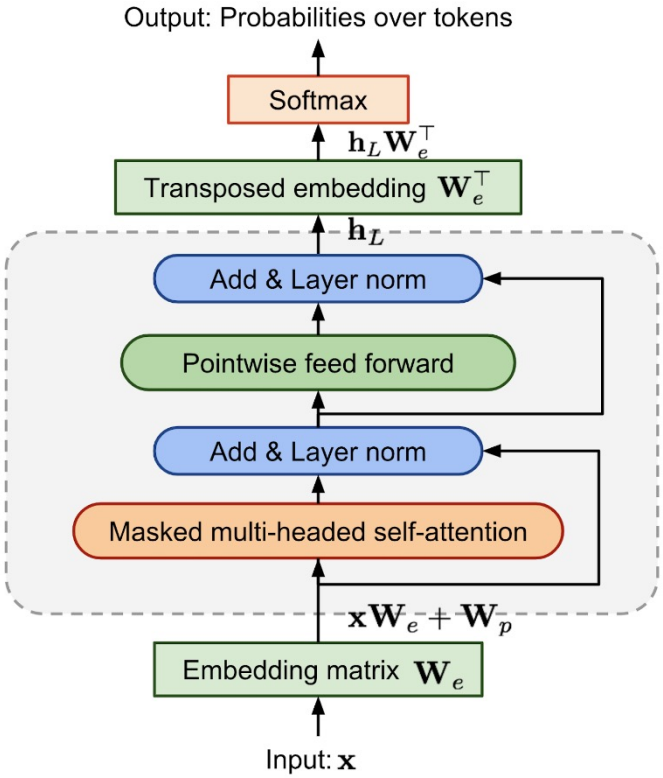
Outline

- **Quick recap on the Transformer**
- Training LLMs
- Model alignment
- Prompting LLMs
- Generation
- Retrieval-Augmented Generation (RAG)
- Open challenges

Large language models

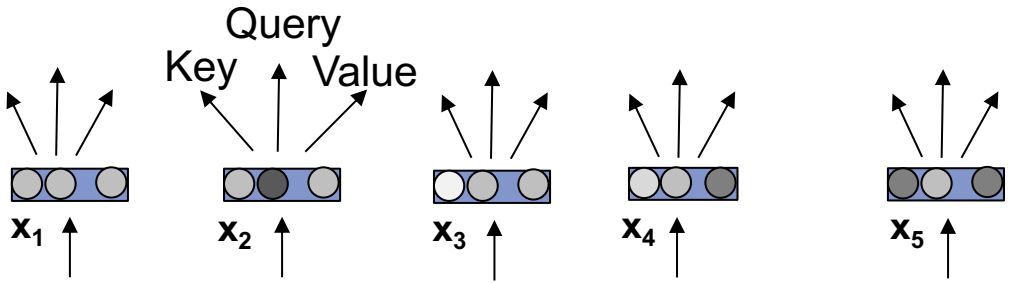
All variants of the same neural architecture: the **Transformer**

- Key mechanism: **self-attention** (takes into account the *context* of each word)
- Many layers stacked onto each other
- **Input**: sequence of tokens
- **Output**: probabilities for the next token



Self-attention

Each token vector $\mathbf{x}_i$ is mapped via *learned* linear projections to three distinct vectors: the query $\mathbf{q}_i$, key $\mathbf{k}_i$ and value $\mathbf{v}_i$



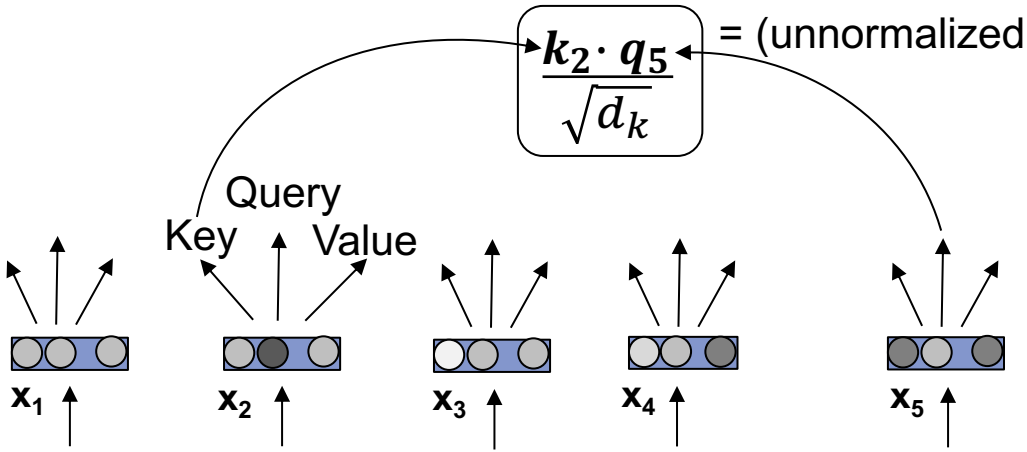
Token vectors at layer *l*

He sat on the bench

Initial tokens

Self-attention

After normalization (softmax), we obtain the final attention scores α_{ij} between all token pairs *(i,j)*



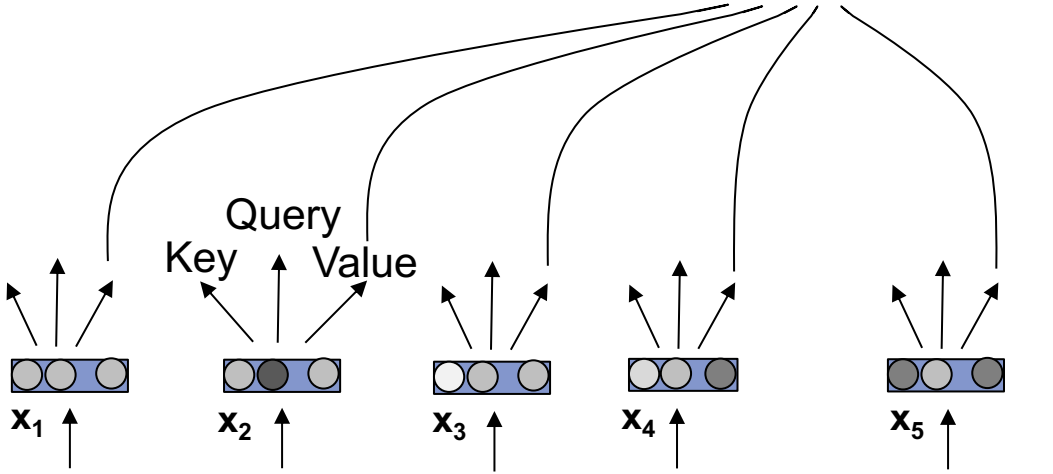
We then compute the *attention scores* between tokens

(by computing the dot products between *query* and *keys*)

Initial tokens

Self-attention

$$a_i = \sum_{j \le i} \alpha_{ij} v_j$$



He sat on the bench

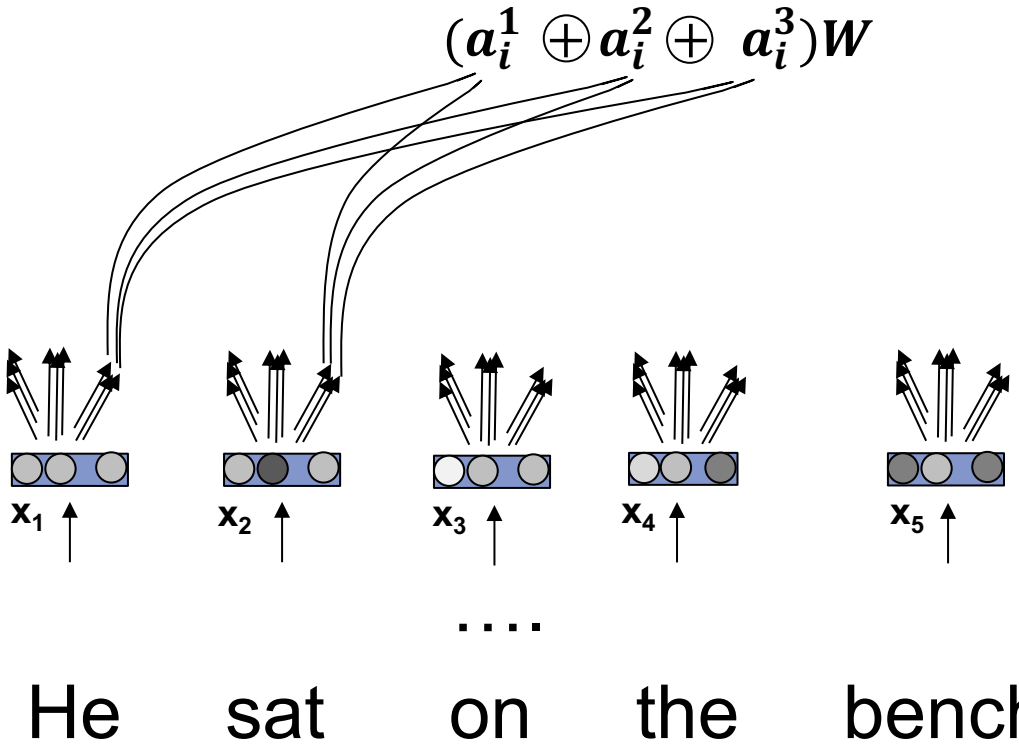
The output vector a_i is the **weighted sum** of the value vectors v_j for all previous tokens $j \le i$

… where the weights are the attention scores α_{ij}

Token vectors at layer *l*

Initial tokens

Self-attention



This process is repeated several times (one per “head”), and the final output vector is the concatenation of all heads, followed by a linear projection

Token vectors at layer *l*

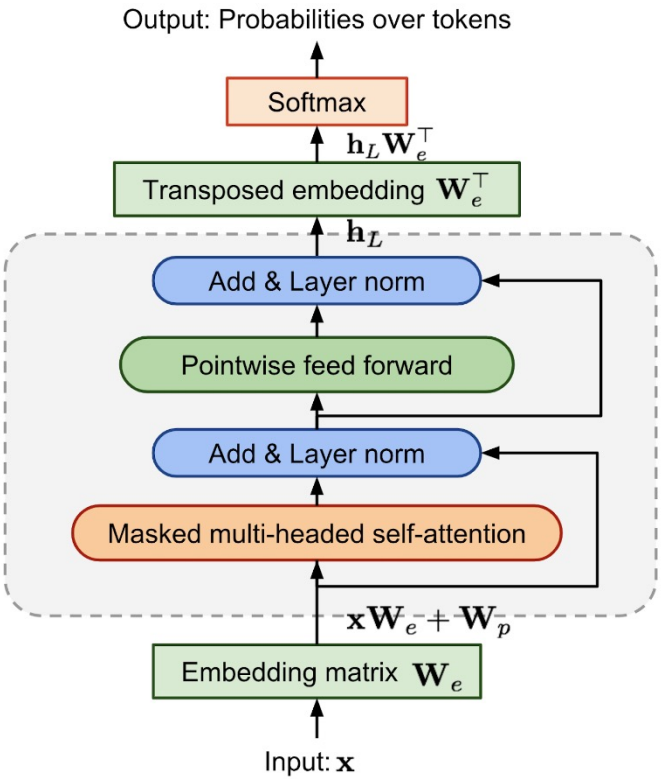
Initial tokens

Transformer blocks

After the self attention, the output vectors a_i are further processed:

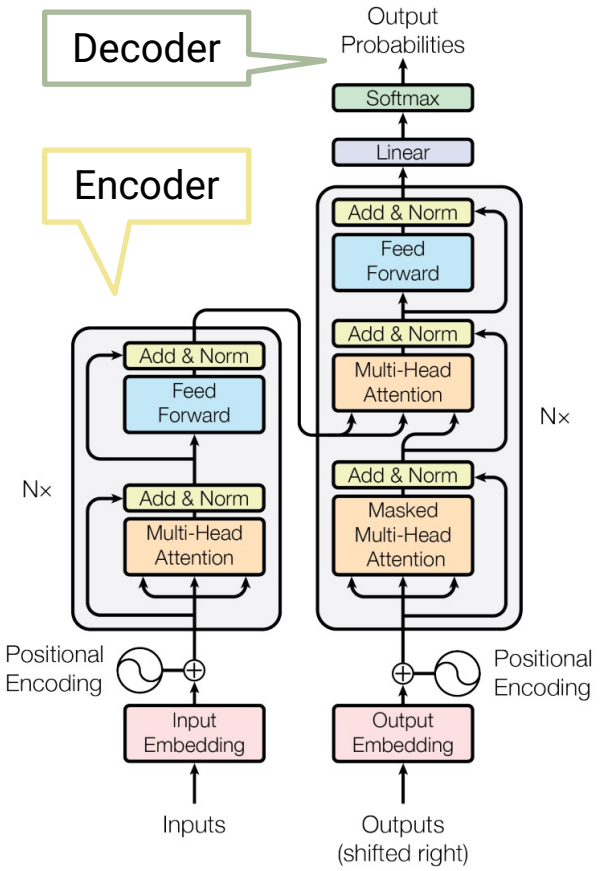
- Addition of residual connections
- Layer normalisation
- Feed-forward neural net

And the resulting vectors then used as input for the next transformer block

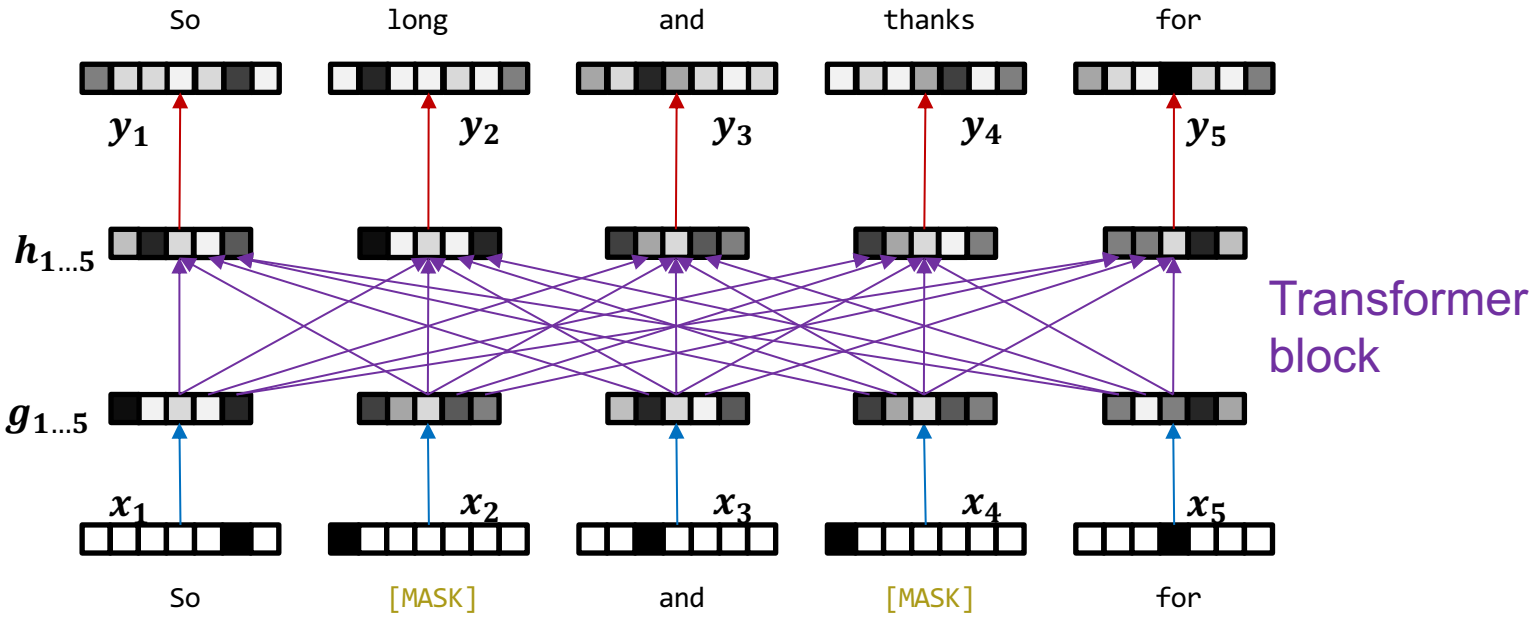


Types of Transformer models

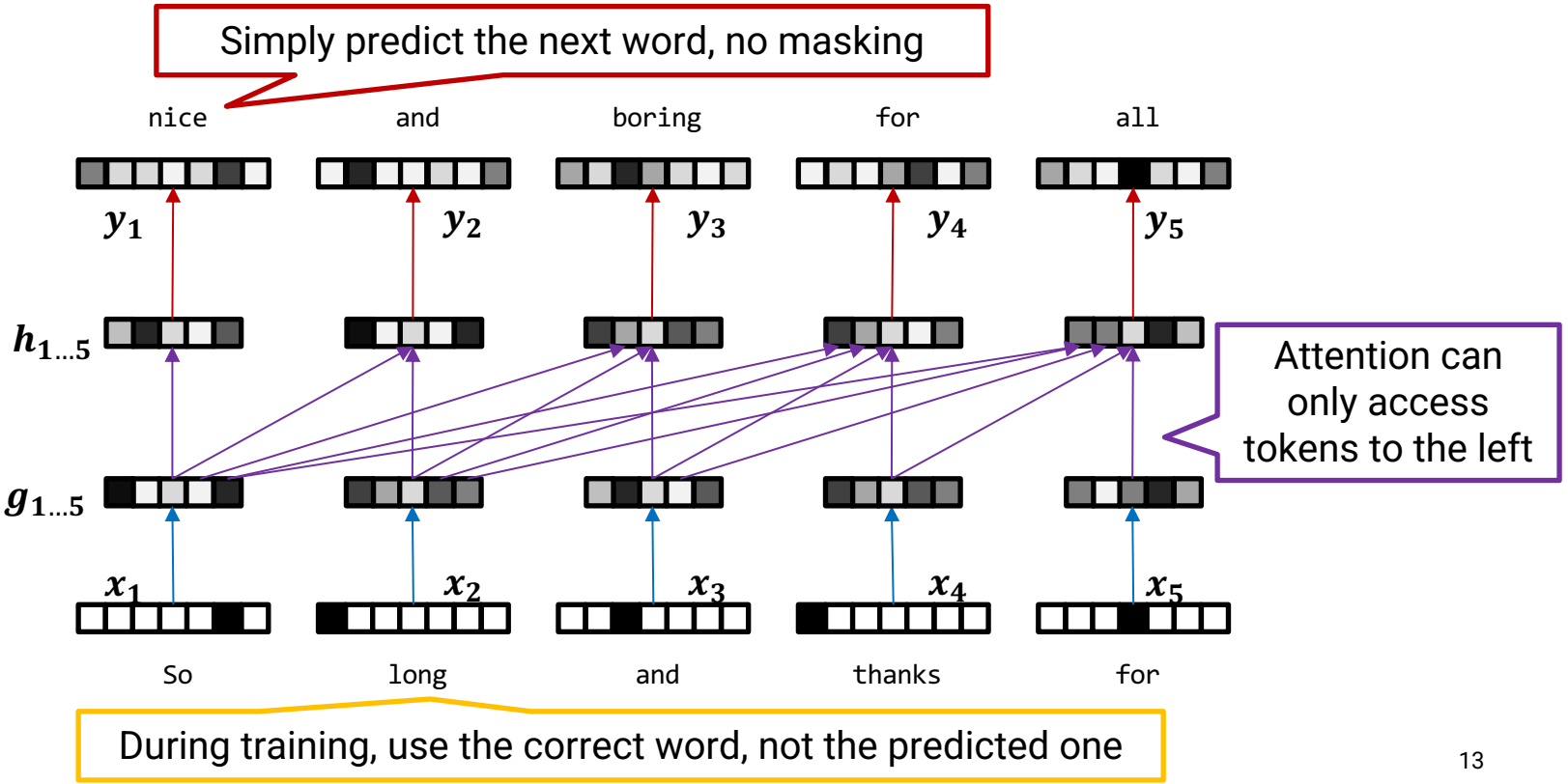
- Encoder models with self-attention
  - BERT
  - Pretraining: masked language modeling
- Decoder models with self-attention
  - GPT
  - Pretraining: next word prediction
- Encoder-decoder models with cross-attention
  - The original Transformer
  - Machine translation
  - Training: sentence pairs



Encoder models – Masked language modelling



Decoder models – Next word prediction



Language modelling with self-attention

Probabilistic n-gram language model:

$$P(w_1, \dots, w_n) = P(w_1) \cdot P(w_2 | \, w_1) \cdot P(w_3 | w_1, w_2) \cdot \dots \cdot P(w_n | w_1, \dots, w_{n-1})$$

- For n-gram models, we could not condition on all preceding tokens
- Thanks to attention, we can do that now
- But typically, far-away tokens get less attention

Language modelling with self-attention

For simplicity, let's assume a single head:

$$\alpha_{t,c} = \text{softmax}\left(\begin{cases} -\infty, & c > t \\ \dfrac{\boldsymbol{q_t} \cdot \boldsymbol{k_c}}{\sqrt{d_k}}, & c \leq t \end{cases}\right)$$

q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

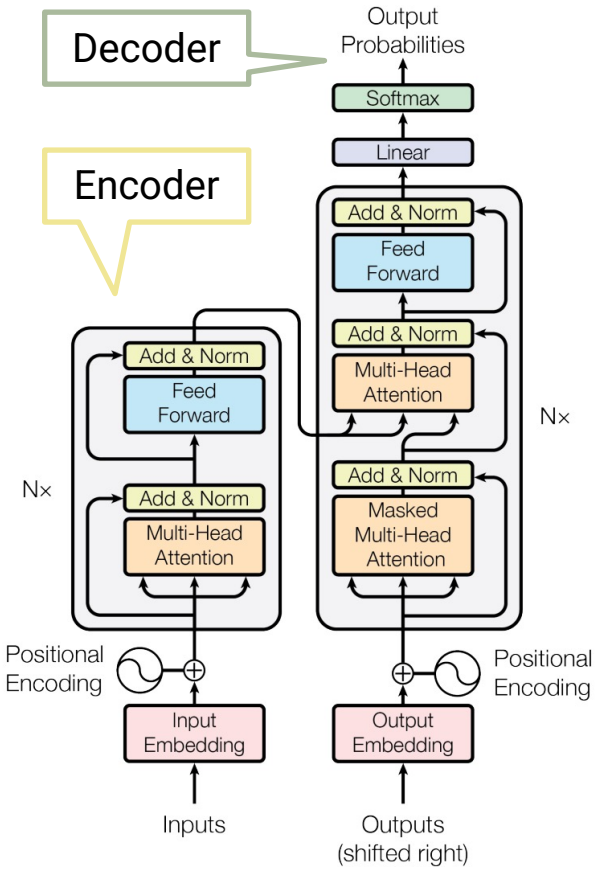
$$\boldsymbol{h_t} = \sigma\left(\sum_{i=1}^{n} \alpha_{t,i} \cdot \boldsymbol{v_t}\right) = \sigma\left(\sum_{j \leq i} \alpha_{j,i} \cdot \boldsymbol{v_j}\right)$$

Language modelling with self-attention

- In terms of model architecture, that’s about all you need to know in order to build your own GPT model ☺
- Of course, you’ll still need the right amount (and quality) of training data…

Types of Transformer models

- Encoder models with self-attention
  - BERT
  - Pretraining: masked language modeling
- Decoder models with self-attention
  - GPT
  - Pretraining: next word prediction
- **Encoder-decoder models with cross-attention**
  - **The original Transformer**
  - **Machine translation**
  - **Training: sentence pairs**



Machine translation

Some (fairly obvious) facts about MT:

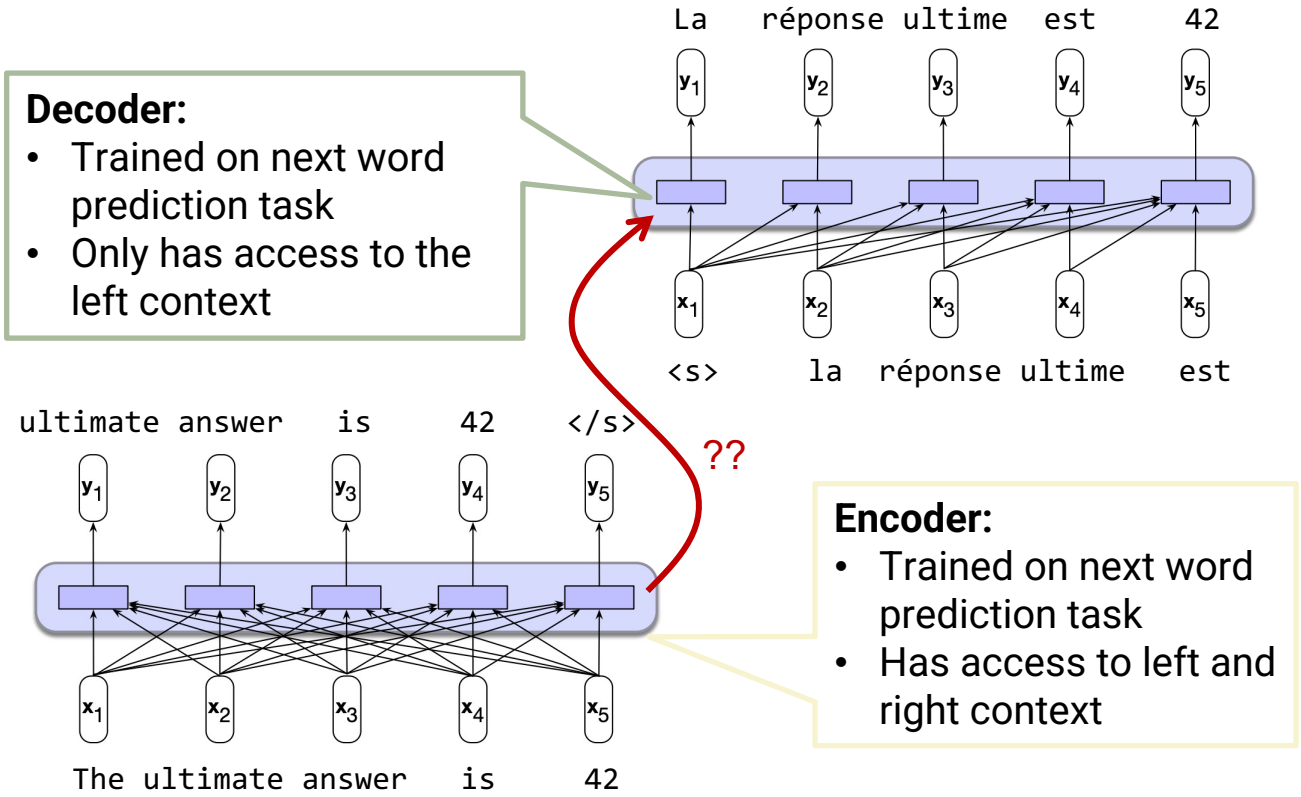
- Input: a sentence in the source language
- Output: a sentence in the target language
  - Hopefully grammatical
  - Hopefully with the same meaning
  - Source and target sentence may have different numbers of words, different word orders

Main idea:

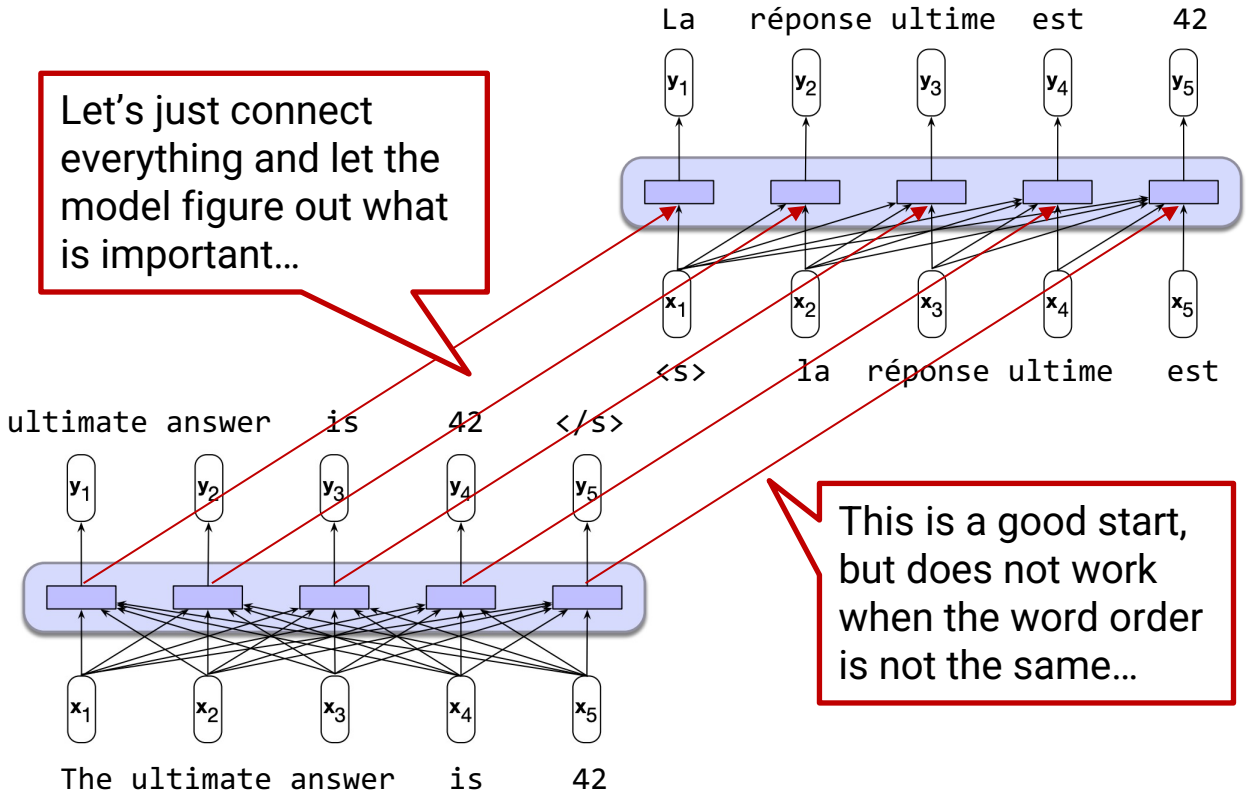
- Train an encoder for the source language
- Train a decoder for the target language
- Connect the two

Everything can be trained together thanks to backpropagation

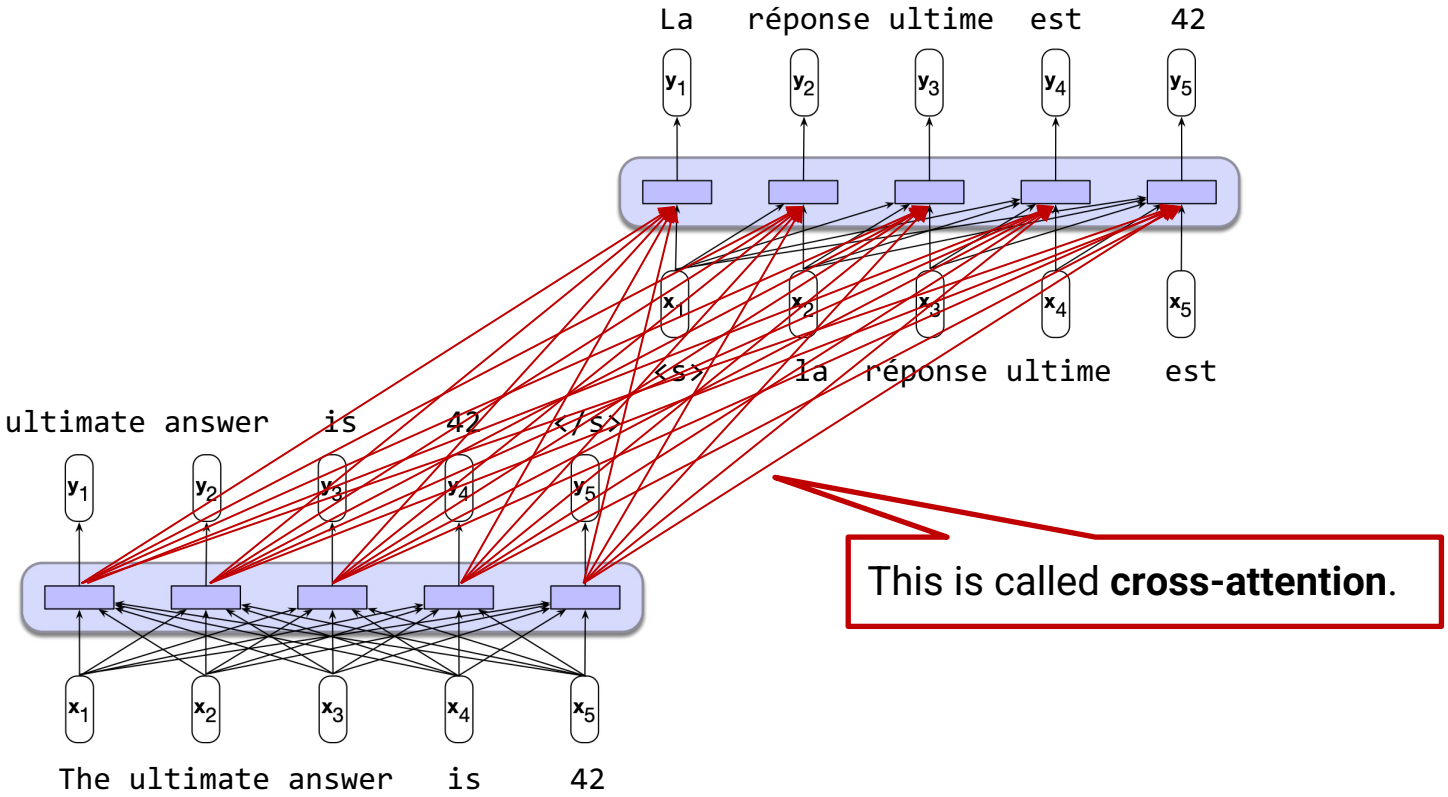
Connecting encoders and decoders



Connecting encoders and decoders



Connecting encoders and decoders



Cross-attention

Dot-product attention:

$$\alpha_{i,j} = \mathrm{softmax}\left(\boldsymbol{h_j^{enc}} \cdot \boldsymbol{h_{i-1}^{dec}}\right)$$

- Determines the importance of the jth word of the encoder for the ith word of the decoder.
- There are other attention types, and one can again use multiple heads.

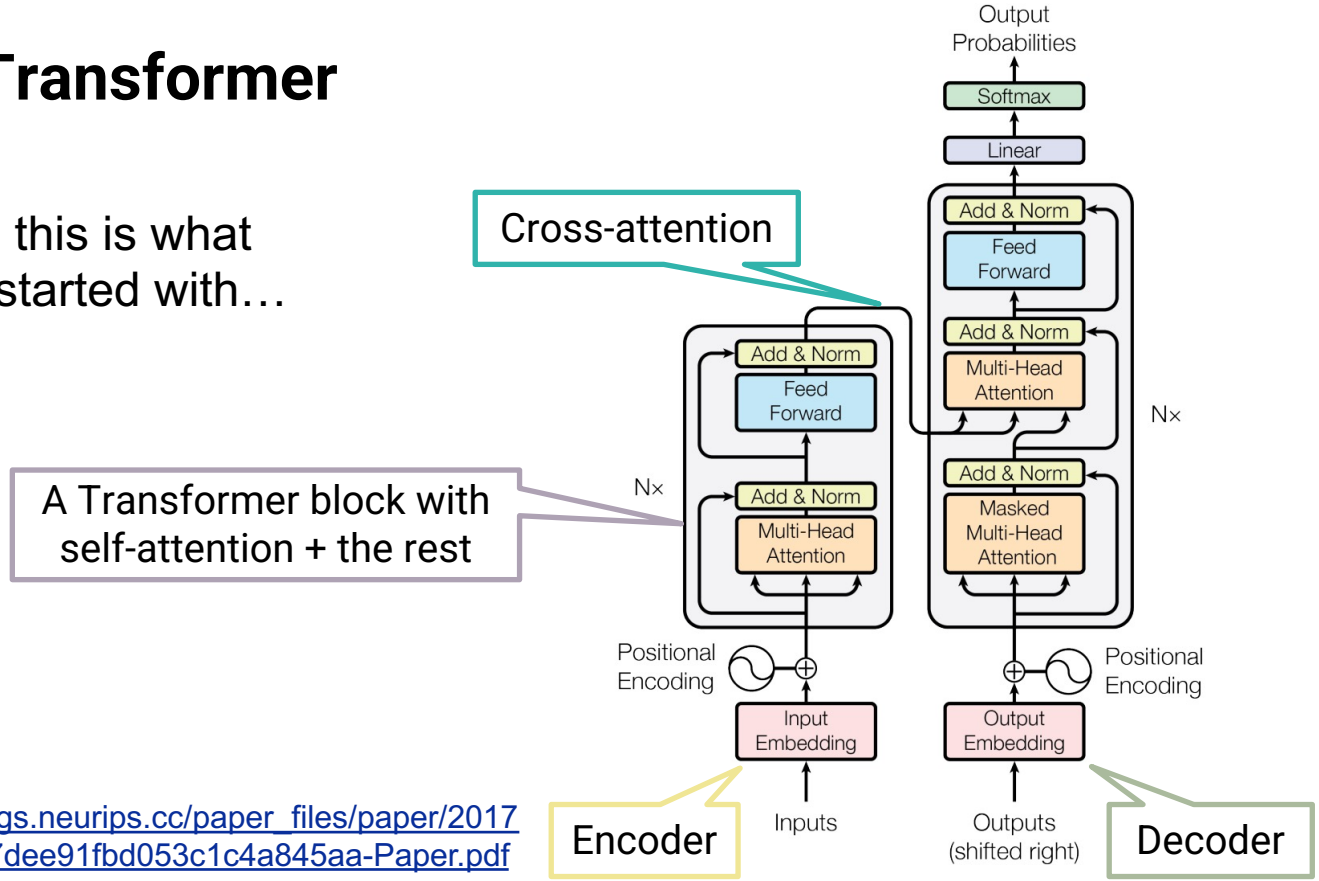
Context vector:

$$\boldsymbol{c_i} = \sum_j \alpha_{i,j} \cdot \boldsymbol{h_j^{enc}}$$

- This vector is then combined with the vector produced by the decoder self-attention.

The full Transformer

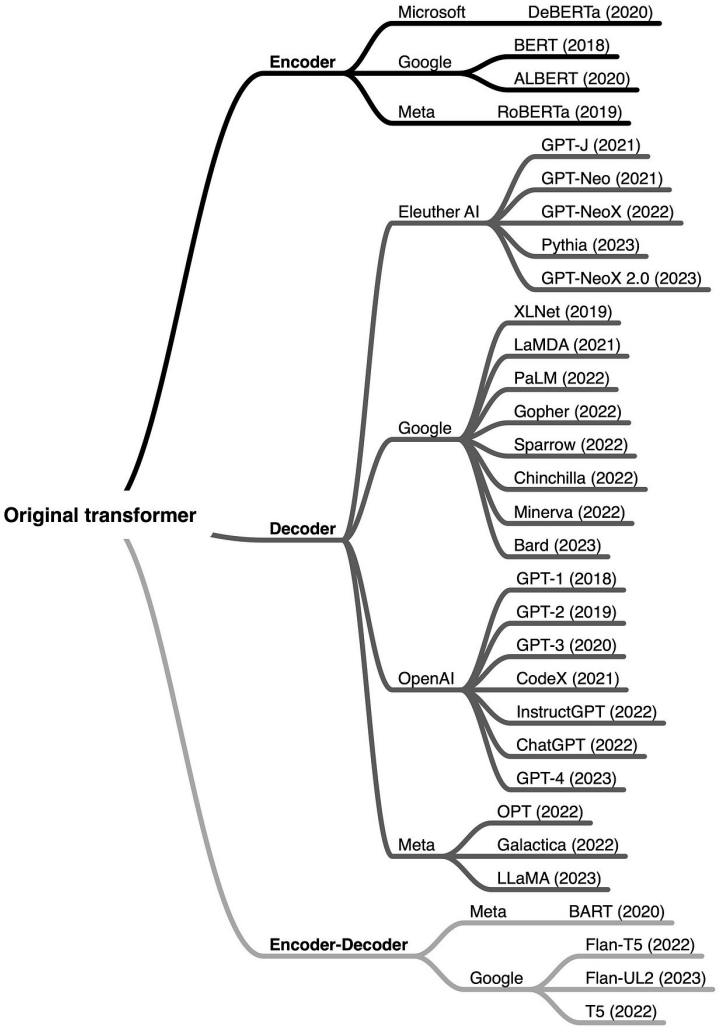
Historically, this is what everything started with…



https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Types of LLMs

- **Encoder-only:**
  - Masked language modelling
  - Smaller & older models, but still very good for IE tasks
- **Decoder-only**:
  - Now the dominant approach to LLMs
- **Encoder-decoder**:
  - Both self-attention and cross-attention
  - Employed for tasks such as MT



Outline

- Quick recap on the Transformer
- **Training LLMs**
- Model alignment
- Prompting LLMs
- Generation
- Retrieval-Augmented Generation (RAG)
- Open challenges

Large Language Models

= Machine learning models optimized to *predict the next word* *in text*

Deep neural net
(*Transformer* architecture)
with many billions of parameters

Trained on huge collections of
texts crawled from the web
(Wikipedia, books, web fora, etc.)

As a by-product of this “guessing game”, the neural model will gradually build up **representations** of both *linguistic* and *world* knowledge

This model can then be *adapted* for various text processing tasks
(generate answers in a dialogue, follow instructions, summarize texts, etc.)

Training process

The LLM is trained through *self-supervision*:

1. Given a text made of tokens $[t_1, t_2, \dots, t_n]$, we take partial sequences $[t_1, t_2, \dots, t_i]$ and seek to predict the next token t_{i+1}
2. The LLM will take as input $[t_1, t_2, \dots, t_i]$ and output a **probability distribution** $\hat{y}_i$ over possible next tokens
3. The loss for this token is then defined as the cross-entropy

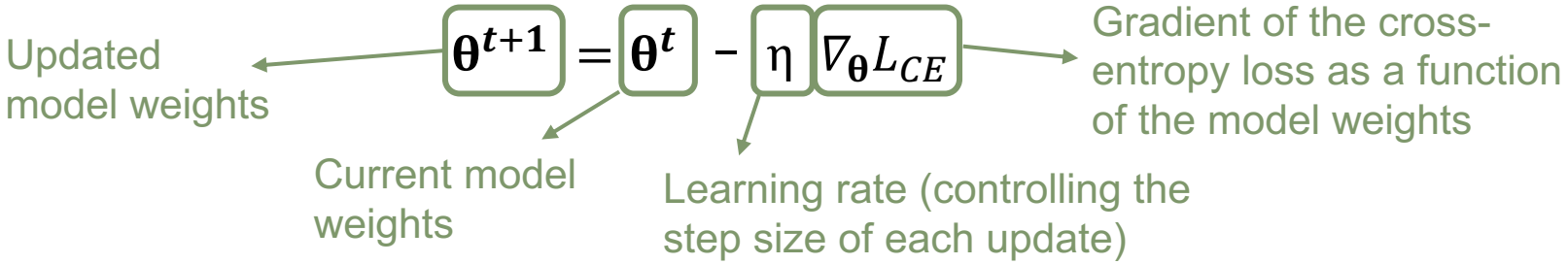
$$L_{CE} = -\log \hat{y}_i[t_{i+1}]$$

Probability of the actual token t_{i+1} in the output distribution $\hat{y}_i$

4. We then make (at batch level) a small update on the model weights $\boldsymbol{\theta}$ to reduce the loss

Training process

- Basic model update through stochastic gradient descent (SGD):



- LLM training relies on optimizers that are variants of SGD:
  - Such as Adam (Adaptive Moment Estimation)
  - + *Distributed* training setup using many GPUs

Training process



State-of-the-art Large Language Models:

- have *hundreds of billions* of parameters
- Are trained with *petabytes of data*
- Optimized *with dozens of* GPUs for many days
- May consist of several «specialized» LLMs (mixture of experts, MoE)

Outline

- Quick recap on the Transformer
- Training LLMs
- **Model alignment**
- Prompting LLMs
- Generation
- Retrieval-Augmented Generation (RAG)
- Open challenges

Fine-tuning

Once a model has been (pre)-trained, it can be *fine-tuned* for a particular task

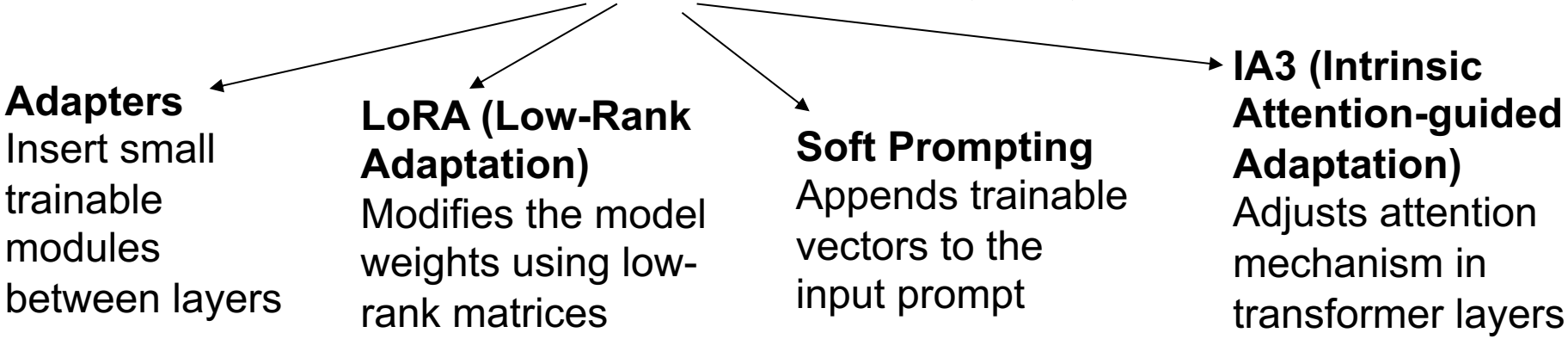
- Basically “further” training with smaller amounts of in-domain data, taking the pretrained model as starting point
- Much more data-efficient than learning a model from scratch!

  (*why*? because the initial model has internalized “semantic representations” that can be transferred to the task to solve)

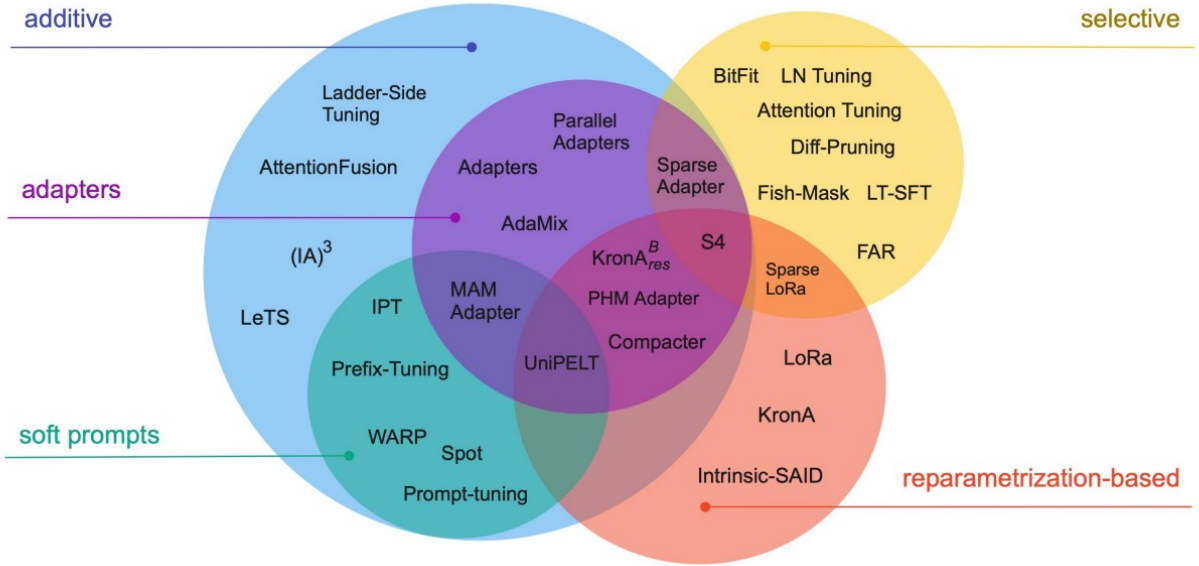


Fine-tuning

- For smaller models like BERT, we can easily do “full” fine-tuning
- For larger LLMs (like Llama, Phi, Mistral, Qwen, …), this is more complicated
- Solution: only optimize a small fraction of parameters
  - So-called Parameter-Efficient Fine Tuning (PEFT)



Parameter-efficient Fine-tuning



And many more…

(new fine-tuning techniques pop up every month)

To get started with PEFT: https://huggingface.co/docs/peft

Model alignment

- A “standard” LLM is optimized to find likely continuations of an input string

  Input: Translate to French: The small dog

  Output: crossed the road → A perfectly plausible continuation, but not what we were looking for

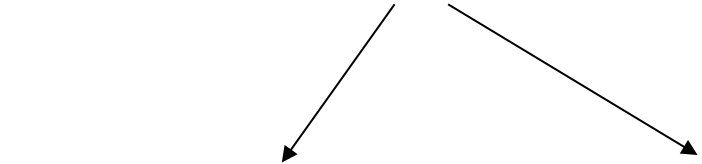
- We often want our LLMs to **follow instructions** and generate outputs that satisfy certain **quality criteria**

  ↓

  Be relevant and informative, do not spread disinformation, avoid offensive or abusive language, refrain from producing dangerous/illegal content, etc.

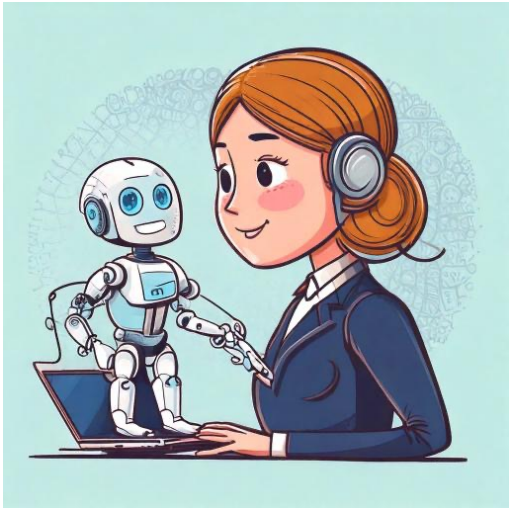
Model alignment

- Getting the model to output “what we want” is often called alignment
  - We seek to “align” the model with certain values & rules of conduct
- This is not a pure technological question!
  - Deep ethical/political questions about what the LLM should “align” towards
- Two alignment strategies:



Instruction tuning

Preference alignment



Instruction tuning

- Classical fine-tuning on a language modelling objective (= next token prediction), but on a dataset of <*instruction, correct response*> pairs
- How to collect such dataset?
  - Manually, by asking humans to write a question/problem and a correct answer to that problem
  - By converting existing datasets for supervised NLP tasks (for text classification, sentiment analysis, NER, machine translation, etc.) to prompts > answers
  - By using LLMs to generate instructions-> responses, based on external databases, general ethical guidelines, etc.

Preference alignment



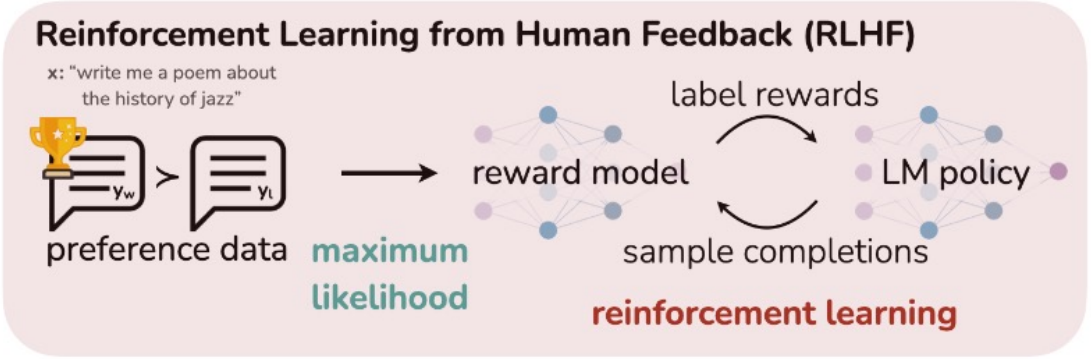


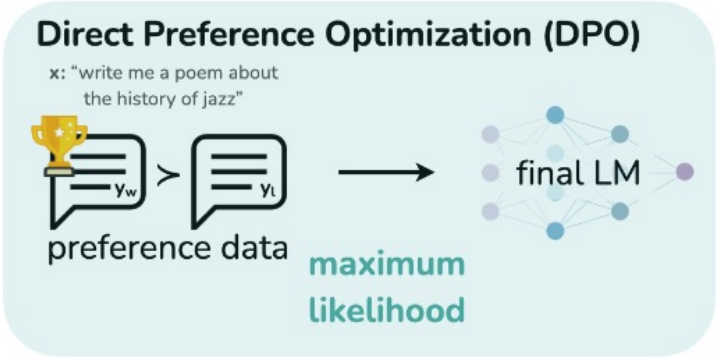
One can also improve the model alignment using **preference data**, such as *ratings* of answers to prompts

- Several formats are possible, but a common one is to use triples <*prompt, chosen response, reject response*>
- The ratings can be collected by paying crowdworkers (based on guidelines that define what constitutes a “good” response), or by asking users to rate the responses they get

Preference alignment

- Given a dataset of human preferences, the goal is to optimize the LLM to maximise the probability of getting a positive rating, and minimize the probability of a negative rating
- Several optimization methods are possible, most prominently RLHR and DPO



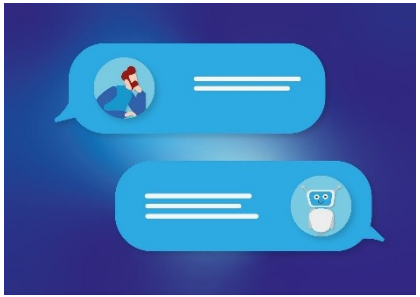


[from Rafailov et al, Direct preference optimization: Your language model is secretly a reward model. NeurIPS, 2023]

Outline

- Quick recap on the Transformer
- Training LLMs
- Model alignment
- **Prompting LLMs**
- Generation
- Retrieval-Augmented Generation (RAG)
- Open challenges

Prompting



- A prompt is simply the **input to the LLM**, which can be a question or instructions on the task to perform
- Many LLMs distinguish between the *system* and *user prompt*:
  - **System prompt**: Generic instructions, defined by the system provider, on how the system should respond (be helpful, avoid offensive language, etc.) and how it should present itself to the user (“You are ChatGPT, a large language model trained by OpenAI…”)
  - **User prompt**: Specific instructions provided by the system

Prompting

How a prompt is formulated can have a large influence on the response quality

If the prompt has both an instruction and a long input text, put the instructions at the end:

X "Translate this into French: Here is the text…."

V "Here is the text... Translate the above text into French"

If the LLM should classify text, or select predefined responses, do specify them clearly in the prompt

"Brussels is the capital of
(A) Belgium
(B) Netherlands
Answer with a single letter corresponding to the correct answer (A or B)."

Specify the *format* and *length* of the response.

For structured outputs, prefer human-readable formats such as JSON or YAML

Add DO and DO NOT whenever necessary

For some prompts, indicating the "intended audience" may also help:

"explain how logistic regression works to a high-school student".



May depend on LLM!

Prompting

- Lots of hype 2-3 years ago on the new job of “prompt engineer”
  - Increasingly less relevant today, as instruction-tuned LLMs get better & better



- But automated prompt tuning *can* improve performance
  - Often through some form of *iterative improvement search*

Initial prompt

Method for generating *variants* of a given prompt

Scoring metric (based on e.g labelled data)

In-context learning

For complex tasks, inserting in the prompt a few examples / demonstrations can help the model understand what kind of output is expected

➔ Called “in-context learning” or “few-shot prompting”



Not really “learning” (the model stays unchanged), but biasing the LLM towards certain types of outputs

Given a text, find all mentions of (direct or indirect) personal identifiers and return them in a list. Do not output anything else.

Examples:

- Text: “My name is Pierre Lison, I am 42 years old and work as a researcher in NLP”
  Output: [Pierre Lison, 42, researcher]
- …

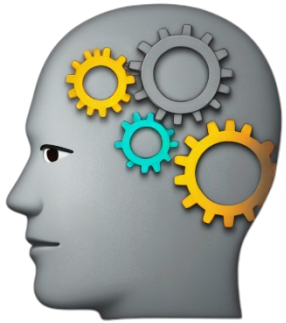
Now find the personal identifiers in this text:
“Yves Scherrer is an Associate Professor at the University of Oslo.”

In-context learning

- No need to include hundreds of examples! Often 1-5 shots are sufficient
- Including examples / demonstrations will increase the length of the prompt, and therefore the cost of each LLM inference
- You should include demonstrations that are as relevant as possible for the actual input to process
  - Either by searching for “similar” demonstrations in a dataset
  - Or programmatically, using prompt optimization tools such as DSPy



Chain-of-Thought



- LLMs remain somewhat limited in their abilities to perform complex, “multi-hop” reasoning
  - Math, science, advanced coding, planning, etc.
- But those “reasoning” capabilities can be improved by forcing the LLM to **generate longer, step-by-step responses**
  - “Let’s think step by step”
  - In-context demonstrations of reasoning steps involved in solving similar problems
  - Can also ask the model to criticize or debate its own responses

Reasoning models

- Reasoning-augmented LLMs like *DeepSeek-R1* or *OpenAI o3* take this idea of reasoning step by step a step further
- Key idea:
  - automatically generate “hard” reasoning problems (each paired with a correct answer), often related to math or programming
  - Use reinforcement learning to train the LLM to generate reasonings that maximise the likelihood of correct answers
- Those models have been shown to yield better performance on complex tasks (but for a higher computational cost)

Outline

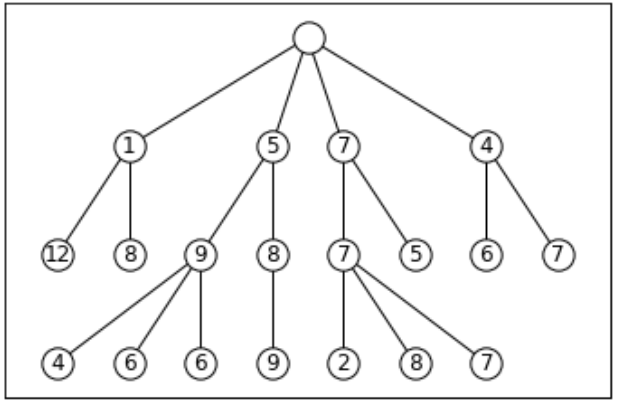
- Quick recap on the Transformer
- Training LLMs
- Model alignment
- Prompting LLMs
- **Generation**
- Retrieval-Augmented Generation (RAG)
- Open challenges

Generation

- Assume we have a trained a generative LLM that takes as input a list of $[t_1, t_2, \ldots, t_n]$ and return a probability distribution over the next token t_{i+1}
- How do we use the LLM to generate the full output ?
- Simplest approach: **greedy decoding**
  - At each step, just select the token *with highest probability*
  - And continue until a special <eos> token, or the maximum output length
- But greedy decoding can be *short-sighted*
  - Can get trapped in local minima (good “local” choice of token, but poor text at the end, with no way to backtrack)

Beam search decoding

- Alternative strategy: **beam search**
  - we keep at each step a «*beam*» of **K** hypotheses representing *the best partial outputs so far*
  - Those K hypotheses are gradually expanded, token by token
  - Until we reach <eos> or the maximum output length
  - Finally, we take the hypothesis with *max probability* among the K alternatives



Beam search with K=3

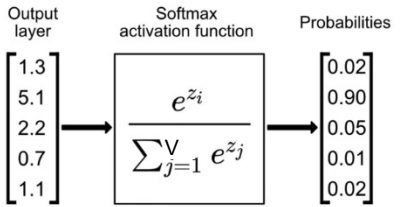
Less risk for local minima, but computational cost increases with K

Sampling strategies

- We can also **sample** the tokens one by one
- **Top-K sampling**
  - At each step, we select the **K** tokens with highest probability
  - We sample the next token only among those K possibilities (after renormalising the distribution)
  - And continue until <eos> or max sentence length
- **Top-p sampling**
  - Same principle, but instead of a fixed K, we select tokens among those making up the top **p** % of the distribution mass

Temperature

- The «creativity» of the generation can be controlled by the **temperature**
- Recall the softmax function that turns logits into a proper distribution



- We can divide the logits by a value T, which gives: $P(w_i) = \frac{e^{\frac{z_i}{T}}}{\sum_{j=1}^{V} e^{\frac{z_j}{T}}}$

Temperatures < 1 will make the decoder more “conservative”, increasing its focus on high-probability tokens

Temperatures > 1 will make the decoder more “creative”

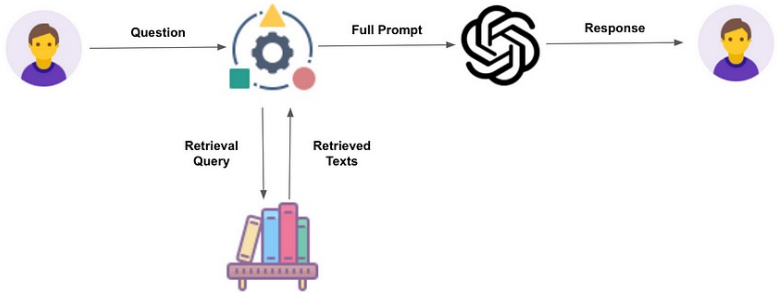
Outline

- Quick recap on the Transformer
- Training LLMs
- Model alignment
- Prompting LLMs
- Generation
- **Retrieval-Augmented Generation (RAG)**
- Open challenges

Retrieval-augmented generation

- A standard LLM has all of its knowledge stored in its weights
- Many shortcomings:
  - Cannot *add*, *remove* or *update* factual knowledge without retraining the model
  - Lack of *transparency*: difficult to inspect the model knowledge, or understand where it comes from
  - How can we connect LLM to *internal databases* or to *specialized knowledge sources* ?
- **Retrieval-augmented generation** (RAG) provides an elegant solution

Retrieval-augmented generation



Basic idea:

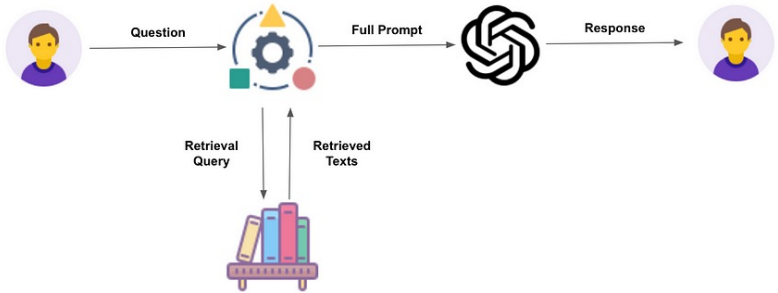
1. Given a user prompt, we first *search* for relevant documents in a text database
2. The most relevant texts are then retrieved and added to the prompt to generate the response:

Retrieval stage

Reading stage

«Given the following information: [RETRIEVED TEXTS], answer the following question: [USER QUESTION]»

Retrieval-augmented generation



- The retrieval stage is essentially a Information Retrieval problem
  - Can use «sparse» retrieval algorithms like TF-IDF or BM25
  - Or «dense» retrievers that operate on document embeddings
  - Can also retrieve information from non-text sources
- The reading stage is done with a standard LLM, ideally fine-tuned to take advantage of the retrieved documents to produce a good response

Retrieval-augmented generation

Main advantages of RAG systems:

1. Easier to **edit** the LLM's factual knowledge: simply update the underlying database, without needing to retrain anything!
2. Cleaner distinction between «linguistic» and «factual» knowledge
3. Better **transparency**: if a model produces a wrong answer, we can check whether the error stems from the retrieved documents
4. Better **attribution**: the LLM can indicate the sources that have been used to generate its response

Outline

- Quick recap on the Transformer
- Training LLMs
- Model alignment
- Prompting LLMs
- Generation
- Retrieval-Augmented Generation (RAG)
- **Open challenges**



Write a short paragraph about the Norwegian Computing Center, and highlight one specific contribution in Norway.

The Norwegian Computing Center (Norsk Regnesentral or NR) is a renowned research institute and hub for computational science and technology in Norway. Established in 1952, NR has played a pivotal role in advancing the field of computer science and mathematics within the country. One of its notable contributions is the development of the NORHEIMSUND model, which has been instrumental in improving the efficiency and accuracy of weather forecasting in Norway. This model utilizes advanced computational algorithms to simulate and predict weather patterns, aiding meteorologists in providing more precise and timely weather forecasts for the region. The Norwegian Computing Center's dedication to innovation and cutting-edge research has had a lasting impact on various sectors in Norway, including meteorology and environmental sciences.

?

?

Factuality

- Language Models are optimized to produce *plausible* texts, not necessarily *correct* ones!
- Incorrect answers may come from the training data, which may contain various types of errors or disinformation…
- … But language models may still hallucinate with a “perfect” training set!
- And often do so in an overly *confident tone*

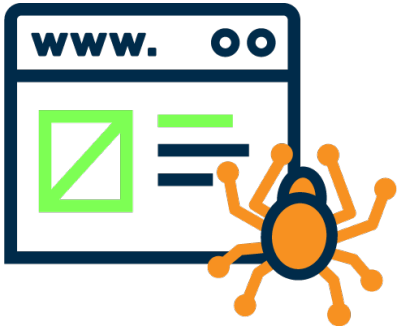


What we haven’t touched

- **Multimodal foundation models** combining text with other modalities, in particular images, videos, but also physical environments (for robotics)
- **Watermarking** of LLM-generated content
- **Broader societal questions**, such as questions related to web crawling and copyright







Next weeks



- Next week, we will talk about **dialogue systems**
  - Text-based (chatbots) or speech-based
  - From simple “intent classification” to LLM-based agents
- In two weeks (October 20): no physical lecture, but a set of shorter videos on advanced topics related to dialogue systems