# Language Technology

## Chapter 17: Transformers: Encoder-Decoder and Decoder

https://link.springer.com/chapter/10.1007/978-3-031-57549-5_17

Pierre Nugues

Pierre.Nugues@cs.lth.se

October 16, 2025

# Machine Translation

- Process of translating automatically a text from a source language into a target language
- Started after the 2nd world war to translate documents from Russian to English
- Early working systems from French to English in Canada
- Renewed huge interest with the advent of the web
- Google claims it has more than 500m users daily worldwide, with 103 languages.
- Massive progress permitted by the encoder-decoder networks

# Corpora for Machine Translation

Initial ideas in machine translation: use bilingual dictionaries and formalize grammatical rules to transfer them from a source language to a target language.

Statistical machine translation:

1. Use very large bilingual corpora;
2. Align the sentences or phrases, and
3. Given a sentence in the source language, find the matching sentence in the target language.

Pioneered at IBM on French and English with Bayesian statistics.

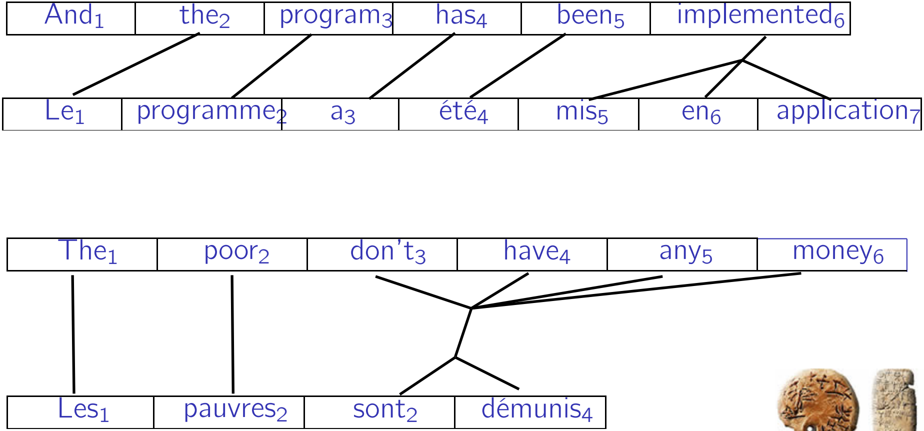
As of today, the encoder-decoder architecture is dominant

# Parallel Corpora (Swiss Federal Law)

| German | French | Italian |
|---|---|---|
| **Art. 35 Milchtransport** | **Art. 35 Transport du lait** | **Art. 35 Trasporto del latte** |
| 1 Die Milch ist schonend und hygienisch in den Verarbeitungsbetrieb zu transportieren. Das Transportfahrzeug ist stets sauber zu halten. Zusammen mit der Milch dürfen keine Tiere und milchfremde Gegenstände transportiert werden, welche die Qualität der Milch beeinträchtigen können. | 1 Le lait doit être transporté jusqu'à l'entreprise de transformation avec ménagement et conformément aux normes d'hygiène. Le véhicule de transport doit être toujours propre. Il ne doit transporter avec le lait aucun animal ou objet susceptible d'en altérer la qualité. | 1 Il latte va trasportato verso l'azienda di trasformazione in modo accurato e igienico. Il veicolo adibito al trasporto va mantenuto pulito. Con il latte non possono essere trasportati animali e oggetti estranei, che potrebbero pregiudicarne la qualità. |

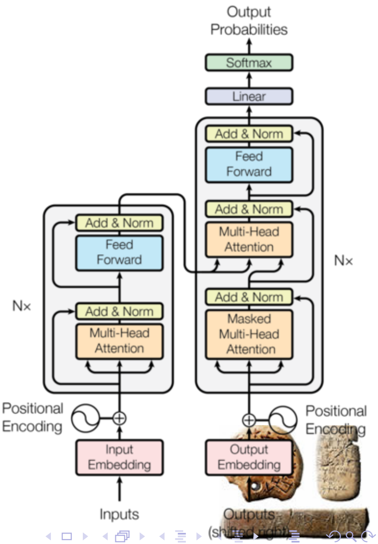
# Alignment (Brown et al. 1993)

## Canadian Hansard



# Transformers: The Encoder-Decoder

1. Transformers were originally developed for machine translation
2. Initial language pairs: English-French, English-German, and the reverse
3. The input corresponds to words in the source language (say English) and the output in the target language (French or German)



# Generation

The transformer consists of an encoder and a decoder.

1. The encoder builds a representation of an input sequence, $(x_1, x_2, ..., x_n)$ called the memory, $M$,

$$\text{encoder}(x_1, x_2, ..., x_n) = M;$$

2. The decoder uses $M$ and a start symbol <s> as first input. The decoder generates a new output and concatenates it until it generates a stop symbol, </s>:

$$\begin{aligned}
\text{decoder}(M, \texttt{<s>}) &= y'_1, \\
\text{decoder}(M, \texttt{<s>}, y'_1) &= y'_2, \\
\text{decoder}(M, \texttt{<s>}, y'_1, y'_2) &= y'_3, \\
&\ldots \\
\text{decoder}(M, \texttt{<s>}, y'_1, y'_2, ..., y'_{p-1}) &= y'_p, \\
\text{decoder}(M, \texttt{<s>}, y'_1, y'_2, ..., y'_{p-1}, y'_p) &= \texttt{</s>}.
\end{aligned}$$

# Example

For two source and target sequences of characters in English and French:

```
Source: ('H', 'e', 'l', 'l, 'o')
Target: ('B', 'o', 'n', 'j', 'o', 'u', 'r')
```

We first encode *Hello*, the source sequence:

$$\text{encoder}(H, e, l, l, o) = M;$$

Using $M$ and <s>, we decode the target sequence:

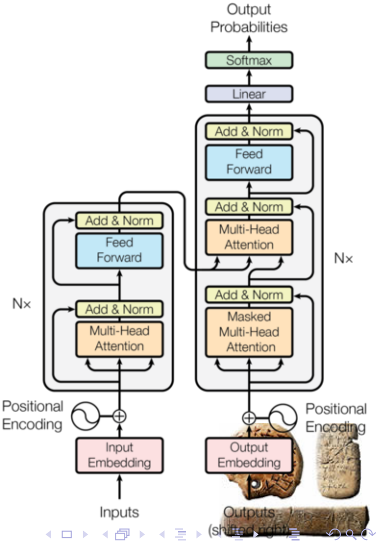
$$\begin{aligned}
\text{decoder}(M, \texttt{<s>}) &= B, \\
\text{decoder}(M, \texttt{<s>}, B) &= o, \\
&\ldots \\
\text{decoder}(M, \texttt{<s>}, B, o, n, j, o, u) &= r, \\
\text{decoder}(M, \texttt{<s>}, B, o, n, j, o, u, r) &= \texttt{</s>},
\end{aligned}$$

We would need a linear layer to map it to the character prediction.

# Decoder (I)

The right part of the initial transformer
Same components:

1. Attention
2. Add and norm
3. Feed-forward



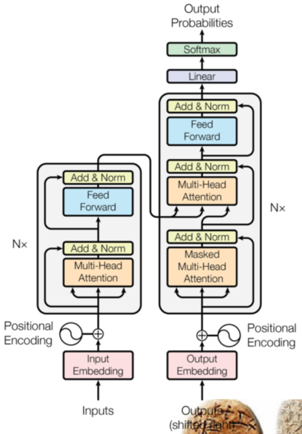
# Decoder (II)

But:

1. Two attentions:
    1. **Masked** self-attention on the inputs
    2. **Cross-attention** with the encoded source
2. Second attention module:
   $Q = X_{\text{dec\_sublayer}}, \; K = Y_{\text{enc}}, \; V = Y_{\text{enc}}$
   with

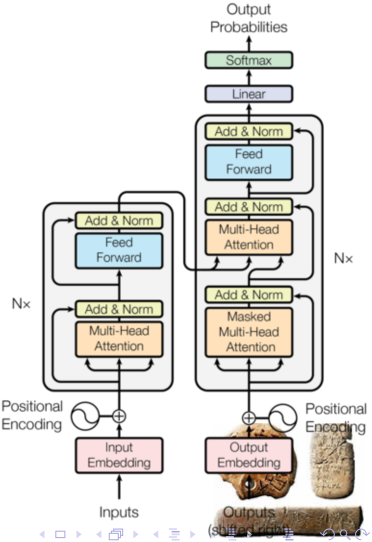
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\intercal}}{\sqrt{d_k}}\right)V,$$

3. The decoder predicts the input shifted by one (linear and softmax)



# Transformer Embeddings

- In Vaswani's paper, the input and output embeddings and the last linear module share the same matrix (weights)
- Same with BERT in the pretraining step for the input embeddings and last linear layer.
- See code: `https://github.com/google-research/bert/blob/master/run_pretraining.py#L240`)

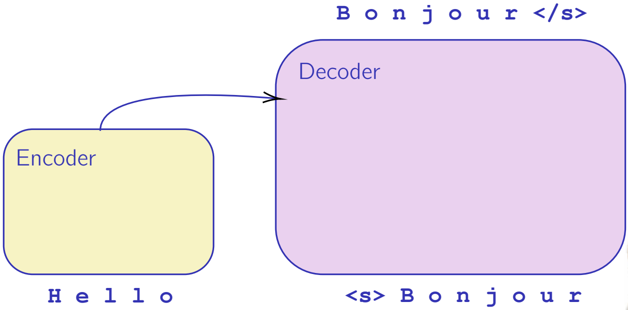


# Prediction with Characters

Translation pairs: (Source sentence, Target sentence)

For instance: (Hello, Bonjour), here with characters

1. Encoder input: *H e l l o*
2. Decoder input: *B o n j o u r* and the encoded *H e l l o*
3. Decoder output: *B o n j o u r* shifted by one to the left
4. We align the decoder strings with `<s>` and `</s>`



# Training and Inference Steps

1. Training step

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Target output: | B | o | n | j | o | u | r | </s> |
| | ↱ | Encoded source: | ⟶ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| Source input: | H e l l o | | Target input: | <s> | B | o | n | j | o | u | r |

2. Inference:

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | Target output: | B | ... | |
| | ↱ | Encoded source: | ⟶ | ↑ | | |
| Source input: | H e l l o | | Target input: | <s> | B | ... |

# Vaswani's Attention Scores

The attention scores are scaled and normalized by the softmax function.

$$\text{softmax}\left(\frac{QK^\intercal}{\sqrt{d_k}}\right),$$

| | i | must | go | back | to | my | ship | and | to | my | crew |
|---|---|---|---|---|---|---|---|---|---|---|---|
| i | 0.36 | 0.05 | 0.07 | 0.05 | 0.04 | 0.19 | 0.01 | 0.02 | 0.04 | 0.19 | 0.01 |
| must | 0.14 | 0.20 | 0.10 | 0.06 | 0.11 | 0.10 | 0.03 | 0.05 | 0.11 | 0.10 | 0.02 |
| go | 0.18 | 0.09 | 0.14 | 0.09 | 0.08 | 0.13 | 0.02 | 0.04 | 0.08 | 0.13 | 0.02 |
| back | 0.14 | 0.05 | 0.09 | 0.19 | 0.08 | 0.12 | 0.03 | 0.06 | 0.08 | 0.12 | 0.03 |
| to | 0.11 | 0.11 | 0.09 | 0.09 | 0.15 | 0.08 | 0.04 | 0.07 | 0.15 | 0.08 | 0.03 |
| my | 0.19 | 0.03 | 0.05 | 0.04 | 0.03 | 0.29 | 0.01 | 0.02 | 0.03 | 0.29 | 0.01 |
| ship | 0.03 | 0.03 | 0.03 | 0.04 | 0.05 | 0.03 | 0.55 | 0.03 | 0.05 | 0.03 | 0.13 |
| and | 0.10 | 0.08 | 0.07 | 0.10 | 0.12 | 0.09 | 0.04 | 0.15 | 0.12 | 0.09 | 0.04 |
| to | 0.11 | 0.11 | 0.09 | 0.09 | 0.15 | 0.08 | 0.04 | 0.07 | 0.15 | 0.08 | 0.03 |
| my | 0.19 | 0.03 | 0.05 | 0.04 | 0.03 | 0.29 | 0.01 | 0.02 | 0.03 | 0.29 | 0.01 |
| crew | 0.06 | 0.05 | 0.05 | 0.06 | 0.05 | 0.06 | 0.21 | 0.04 | 0.05 | 0.06 | 0.31 |

# Attention

We use these scores to compute the attention.

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

For *ship:*

```
attention_ship = (0.03 * embeddings_dict['i'] +
                  0.03 * embeddings_dict['must'] +
                  0.03 * embeddings_dict['go'] +
                  0.03 * embeddings_dict['back'] +
                  0.04 * embeddings_dict['to'] +
                  0.05 * embeddings_dict['my'] +
                  0.55 * embeddings_dict['ship'] +
                  0.03 * embeddings_dict['and'] +
                  0.05 * embeddings_dict['to'] +
                  0.03 * embeddings_dict['my'] +
                  0.13 * embeddings_dict['crew'])
```

where the *ship* vector received 13% of its value from *crew*

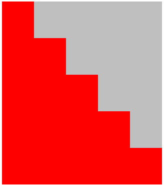
Is this possible in an autoregressive setting?

# Decoder Masking

Masking is used in the training step of a decoder

We use an upper triangular matrix $U_{-\infty}$ to prevent a look ahead

We just replace the gray cells with $-10^9$ and keep the original values of the red cells



$$\text{MaskedAttention}(Q, K, V, U_{-\infty}) = \text{softmax}\left(\frac{QK^\intercal}{\sqrt{d_k}} + U_{-\infty}\right)V.$$

# After Masking

The attention scores after masking.

| | i | must | go | back | to | my | ship | and | to | my | crew |
|---|---|---|---|---|---|---|---|---|---|---|---|
| i | 1.00 | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| must | 0.42 | 0.58 | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| go | 0.44 | 0.22 | 0.35 | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| back | 0.29 | 0.11 | 0.19 | 0.40 | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| to | 0.20 | 0.20 | 0.16 | 0.17 | 0.27 | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| my | 0.30 | 0.05 | 0.08 | 0.07 | 0.04 | 0.45 | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| ship | 0.04 | 0.04 | 0.04 | 0.05 | 0.06 | 0.05 | 0.73 | **0.00** | **0.00** | **0.00** | **0.00** |
| and | 0.14 | 0.10 | 0.09 | 0.13 | 0.16 | 0.12 | 0.05 | 0.21 | **0.00** | **0.00** | **0.00** |
| to | 0.12 | 0.12 | 0.10 | 0.11 | 0.16 | 0.10 | 0.04 | 0.08 | 0.16 | **0.00** | **0.00** |
| my | 0.20 | 0.03 | 0.05 | 0.05 | 0.03 | 0.29 | 0.01 | 0.02 | 0.03 | 0.29 | **0.00** |
| crew | 0.06 | 0.05 | 0.05 | 0.06 | 0.05 | 0.06 | 0.21 | 0.04 | 0.05 | 0.06 | 0.31 |

# Evaluating Translation

The standard metric to evaluate translation is the bilingual evaluation understudy (BLEU) algorithm

BLEU compares the machine translation of a sentence with a corresponding human translation.

- Based on the number of machine-translated words that appear in the human translation divided by the total number of words in the machine-translated sentence
- BLEU extends the computation to *n*-grams, up to 4, and scales it with a brevity penalty
- The final score is the average on the test set

Although very basic, studies have shown it correlates well with human judgement

Vaswani et al. (2017) reported BLEU scores of 41 for the French-English pair and of 26.4 for German-English

# Code Example

**Experiments:**

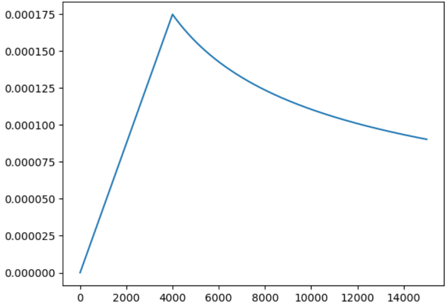
- Jupyter Notebook: `17_01_masked_attention.ipynb`
- 6th laboratory on machine translation

# Learning Rate

In other labs, we used a constant learning rate
Vaswani et al. (2017) used a variable rate:

$$\text{lrate} = d_{model}^{-0.5} \cdot \min(\text{step\_num}^{-0.5}, \text{step\_num} \cdot \text{warmup\_steps}^{-1.5})$$

where warmup_steps $= 4000$.



# Label Smoothing

True labels and predictions:

$$\text{Truth:} \quad (0,0,1,0) \quad \text{Prediction:} \quad (0.1, 0.2, 0.4, 0.3)$$

Cross-entropy loss we used so far:

$$\begin{aligned} -(0,0,1,0)\cdot\log(0.1,0.2,0.4,0.3) &= -(0\cdot\log 0.1 + 0\cdot\log 0.2 + 1\cdot\log 0.4 + 0\cdot\log 0.3) \\ &= -\log 0.4 \end{aligned}$$

Label smoothing: we dedicate a small amount to the wrong predictions:

$$(1-\epsilon)\mathbf{e}_i + \frac{\epsilon}{N_{\text{subwords}} - 1} \sum_{j=1, j\neq i}^{N_{\text{subwords}}} \mathbf{e}_j,$$

Cross-entropy loss with $\epsilon = 0.1$:

$$-\left(\frac{0.1}{3}, \frac{0.1}{3}, 0.9, \frac{0.1}{3}\right)\cdot\log(0.1,0.2,0.4,0.3) = -\left(\frac{0.1}{3}\cdot\log 0.1 + \frac{0.1}{3}\cdot\log 0.2 + 0.9\cdot\log 0.4 + \frac{0.1}{3}\cdot\log 0.3\right)$$

# Beam Search

1. In the 6th lab, you will use a greedy decoding: You keep the highest prediction
2. A beam search uses the $N$ highest predictions at each step
3. You rank the prediction sequences (the paths) with the probability product
4. $N$ is called the beam diameter

# PyTorch Decoders

PyTorch has a class to create a decoder layer (https://pytorch.org/docs/stable/generated/torch.nn.TransformerDecoderLayer.html):

```
decoder_layer = nn.TransformerDecoderLayer(d_model, nheads)
dec_layer_output = decoder_layer(input, memory)
```

and another to create a stack of $N$ layers (https://pytorch.org/docs/stable/generated/torch.nn.TransformerDecoder.html):

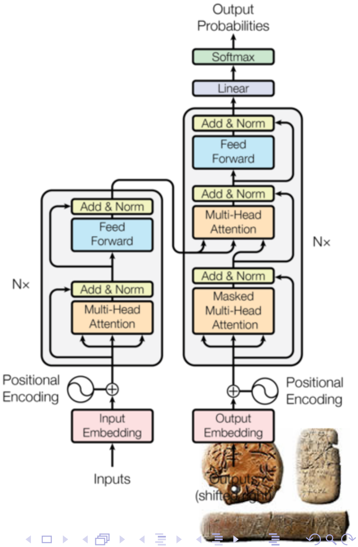
```
decoder = nn.TransformerDecoder(decoder_layer, num_layers)
```

```
decoder = nn.TransformerEncoder(decoder_layer, 6)
dec_output = decoder(input, memory)
```

# Autoregressive Models

Reference papers:

1. GPT: *Improving Language Understanding by Generative Pre-Training* by Radford et al. (2018) https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
2. GPT-2: *Language Models are Unsupervised Multitask Learners* by Radford et al. (2018) https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
3. Source code:
   - https://github.com/openai/gpt-2
   - https://github.com/karpathy/nanoGPT
   - https://github.com/karpathy/nanochat/

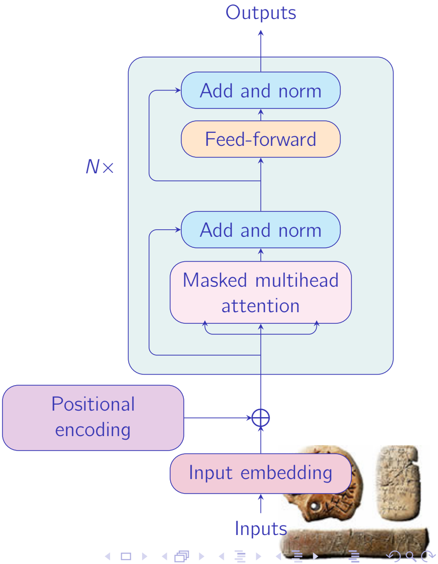


# Transformer Decoder

Pretrained as an autoregressive language model:

$$P(\mathbf{x}) = \prod_{i=1}^{n} P(x_i | x_1, \ldots, x_{i-1}).$$

Note there is no cross-attention



# Fine Tuning (I)

## Fine-tuning applications in the first paper (GPT)

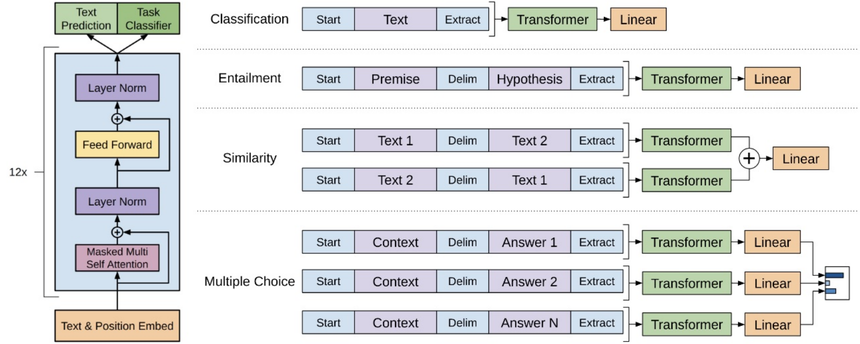



Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

# Fine Tuning (II)

The system in the second paper (GPT-2) is pretrained using the same objective function and a more general formulation:

$$P(\text{output}|\text{input})$$

Large corpora encapsulate more knowledge, such as examples of tasks:

$$P(\text{output}|\text{input}, \text{task})$$

Training examples in the second paper:

- translate to French, English text, French text;
- answer the question, document, question, answer

# Prompts

**Zero-shot**

| | |
|---|---|
| *Task description* | Translate English to French: |
| *Prompt* | cheese ⇒ |

**One-shot**

| | |
|---|---|
| *Task description* | Translate English to French: |
| *Example* | sea otter ⇒ loutre de mer |
| *Prompt* | cheese ⇒ |

**Few-shot**

| | |
|---|---|
| *Task description* | Translate English to French: |
| *Example* | sea otter ⇒ loutre de mer |
| *Example* | peppermint ⇒ menthe poivrée |
| *Example* | plush giraffe ⇒ girafe en peluche |
| *Prompt* | cheese ⇒ |

# Prompt Engineering

Many techniques to write prompts

Follow the ones recommended by the language model

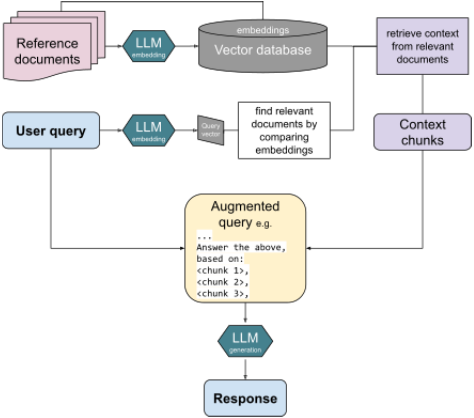
See for instance the guides by Hugging Face:

- On prompting: https://huggingface.co/docs/transformers/main/en/tasks/prompting
- Leaderboard https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard

Why does it work? Two explanations:

- https://x.com/fchollet/status/1709242747293511939
- https://arxiv.org/pdf/2212.07677

# Retrieval-Augmented Generation




Credit: Wikipedia, https://en.wikipedia.org/wiki/Retrieval-augmented_generation

Paper: https://arxiv.org/pdf/2005.11401 Benchmark:

https://huggingface.co/spaces/mteb/leaderboard

# Fine-Tuning Chat Decoders

Many large systems like chatGPT do not disclose their models.
The training procedure:

- Starts from an autoregressive decoder model pretrained on large corpora, see Llama (https://arxiv.org/pdf/2302.13971)
- Possibly train further the model on dedicated corpora: continued pretraining (https://docs.unsloth.ai/basics/continued-pretraining)
- Fined-tuned on instructions with the format: Instruction, Input, Output (https://docs.unsloth.ai/get-started/fine-tuning-llms-guide)

# Fine-Tuning Strategies

Fine-tuning large models is difficult:

1. Refit all the parameters may destroy the model
2. Unfreeze selected layers is a popular method
3. Low Rank Adaptation (LoRA) has emerged as a key technique.
4. Source: `https://arxiv.org/abs/2106.09685`

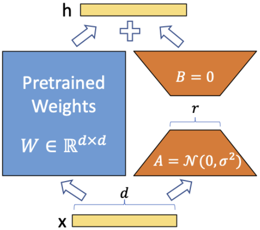



Figure 1: Our reparametrization. We only train $A$ and $B$.

# Low Rank Adaptation

The idea is:

1. Fine-tune the attention matrices only: $W^Q$, $W^K$, $W^V$, and $W^O$
2. Simplify attention with one head: $W \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$
3. Freeze the $W$ pretrained matrices
4. Add them the product of two **trainable** low-rank matrices: $A$ and $B$
5. $B \in \mathbb{R}^{d_{\text{model}} \times r}$ and $A \in \mathbb{R}^{r \times d_{\text{model}}}$ with $r = 4$ for instance
6. There are $2 \cdot r \cdot d_{\text{model}}$ parameters to adjust instead of $d_{\text{model}} \cdot d_{\text{model}}$
7. For an $\mathbf{x}$ input, the output is $W\mathbf{x} + BA\mathbf{x}$

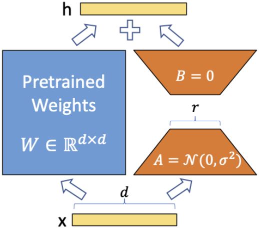



Figure 1: Our reparametrization. We only train $A$ and $B$.

Source: https://arxiv.org/abs/2106.09685

# Human Feedback

1. Large companies often apply supervised fine-tuning, see `https://arxiv.org/pdf/2307.09288.pdf`
2. Instructions could be tedious to write though
3. Answers evaluated by humans and feedback incorporated by simple reinforcement learning: Reinforcement learning with human feedback (RLHF)
    - Start with annotated data on human preferences
    - Learn a policy
    - Let the policy rank the LLM answers
4. See how to make a LLM nicer here: `https://github.com/ash80/RLHF_in_notebooks`
5. No unique solution: Rapidly developing field

# Self-Instruct

We follow self-instruct from Wang et al. (2023) to minimize human effort:

- Starts with a seed of human-written instructions: less than 200. `https://github.com/yizhongw/self-instruct/blob/main/data/seed_tasks.jsonl`
- The model generates more output from the seed instructions. The autogenerated instructions amount to 50k

Fine-tuning using the

> *concatenat[ion of] the instruction and instance input as a prompt and train the model to generate the instance output in a standard supervised way.*

From Wang et al. (2023) (`https://arxiv.org/pdf/2212.10560.pdf`, `https://github.com/yizhongw/self-instruct`)

# Seed Instructions

```
{"id": "seed_task_0",
"name": "breakfast_suggestion",
"instruction": "Is there anything I can eat for a breakfast that doesn't include eggs, yet includes protein, and has roughly 700-1000 calories?",
"instances": [
{"input": "",
"output": "Yes, you can have 1 oatmeal banana protein shake and 4 strips of bacon. The oatmeal banana protein shake may contain 1/2 cup oatmeal, 60 grams whey protein powder, 1/2 medium banana, 1tbsp flaxseed oil and 1/2 cup watter, totalling about 550 calories. The 4 strips of bacon contains about 200 calories."}],
"is_classification": false}
```

# Self-Instruct

## Self-instruct from Wang et al. (2023) (https://arxiv.org/pdf/2212.10560.pdf

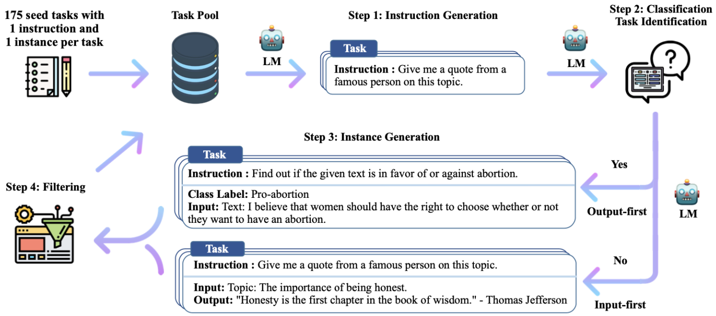



Figure 2: A high-level overview of SELF-INSTRUCT. The process starts with a small seed set of tasks as the task pool. Random tasks are sampled from the task pool, and used to prompt an off-the-shelf LM to generate both new instructions and corresponding instances, followed by filtering low-quality or similar generations, and then added back to the initial repository of tasks. The resulting data can be used for the instruction tuning of the language model itself later to follow instructions better. Tasks shown in the figure are generated by GPT3.

# Alpaca

Alpaca is another example:
https://crfm.stanford.edu/2023/03/13/alpaca.html

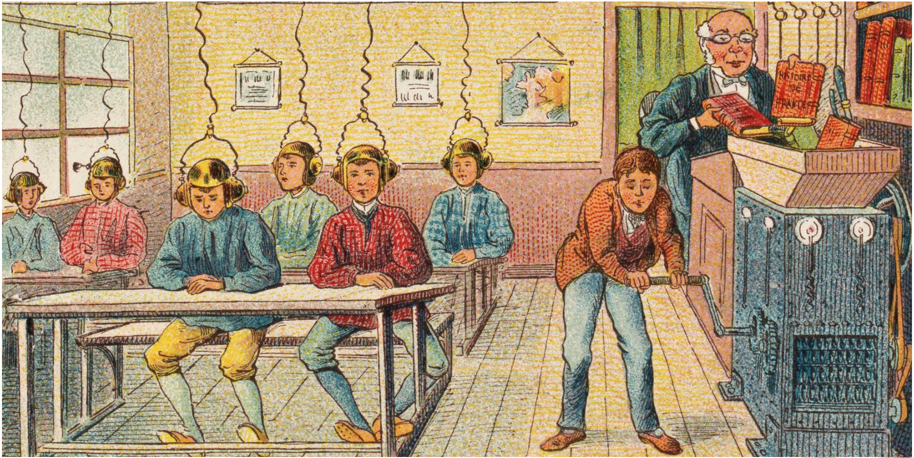
- Uses Llama and Llama 2 as autoregressive pretrained decoder model
- Fined-tuned on instructions with the format: Instruction, Input, Output

# Corpus Choice

The corpus selection is very important in the quality of answers.

1. Usually starts a large sample of web pages; see LLama and LLama 2 `https://arxiv.org/pdf/2302.13971.pdf`
2. Deduplicates the pages (`https://aclanthology.org/2020.lrec-1.494.pdf`)
3. Computes the proximity with wikipedia and discard distant pages
4. Computes the perplexity with a wikipedia model and discard too simple and too complex sentences

# The End




Credits: https://essentiels.bnf.fr/fr/image/d1ed5fad-ac11-4295-a295-3f9898fb492b-1910-villemard-imagine-e