Language Technology
http://cs.lth.se/edan20/
Chapter 13: Transformers: The Decoder

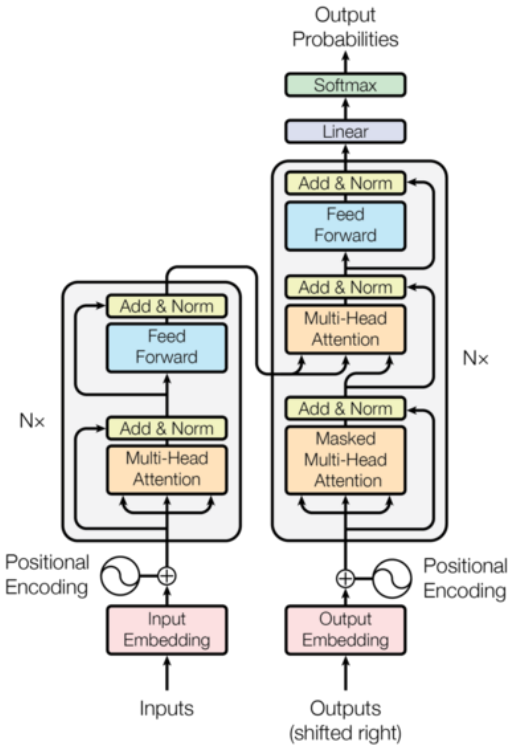
Pierre Nugues

Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

October 12, 2023

Transformers: The Decoder

Transformers were originally developed for machine translation
Initial language pairs were: English-French, English-German, and the reverse



The input corresponds to words in the source language (say English) and the output in the target language (French or German)

Language Models

- A language model is a statistical estimate of a word sequence.
- Originally developed for speech recognition
- A causal or autoregressive language model enables us to predict the next word given a sequence of previous words:
- Given the sequence $x_1, x_2, \ldots, x_{n-1}$, predict x_n

$$P(x_n|x_1, x_2, \ldots, x_{n-1})$$

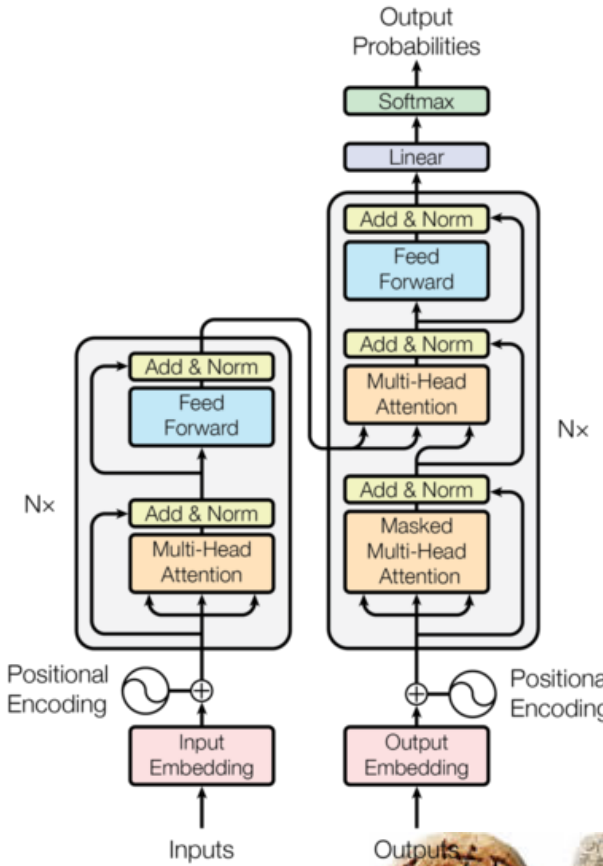
- We approximated sequences with:
  - Bigrams, sequences of two words, given the previous word predict the next one
  - N-grams, sequences of N words
- Evaluation with the negative log-likelihood:

$$L(\mathbf{x}) = -\sum_i \log(x_i|x_{i-N}, x_{i-N+1}, \ldots, x_{i-1}; \theta)$$

Decoder (I)

The right part of the initial transformer
Same components:

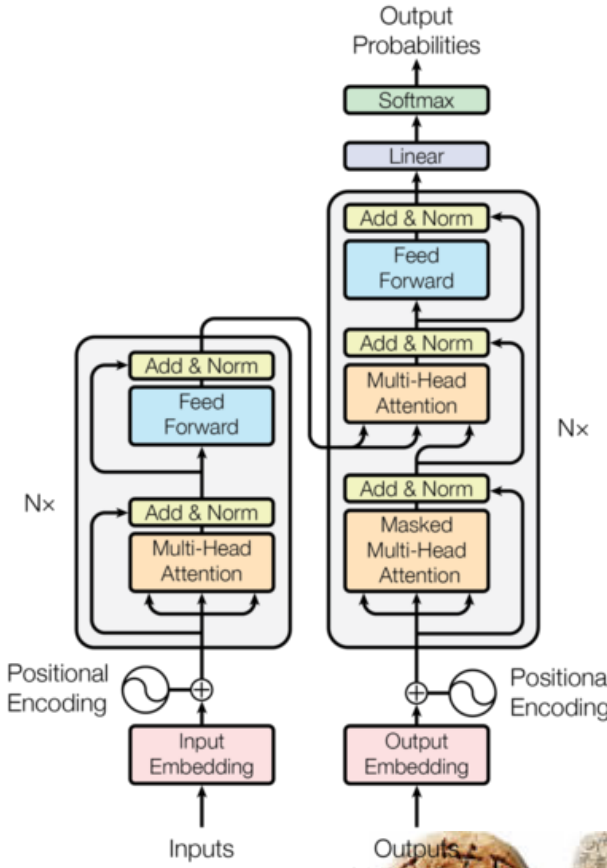
1. Attention
2. Add and norm
3. Feed-forward



Decoder (II)

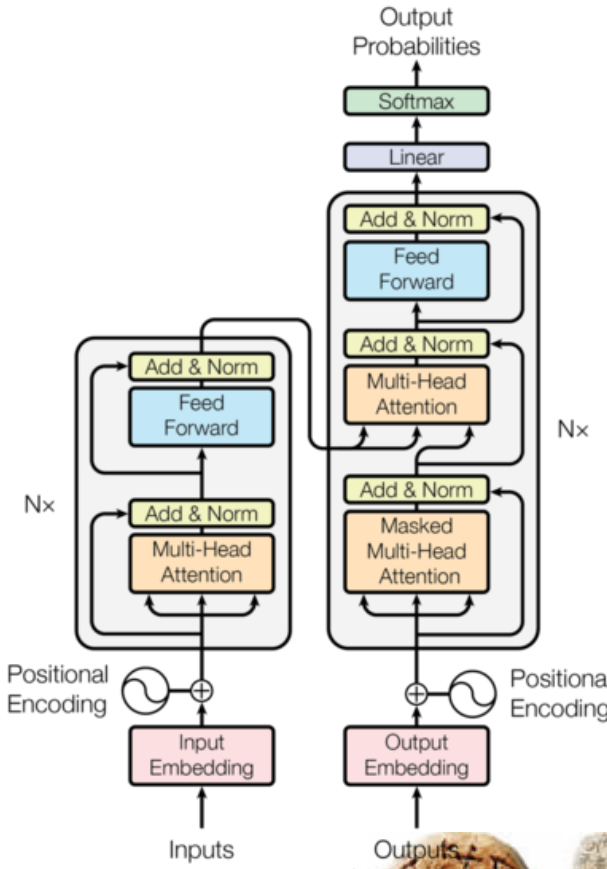
But:

1. Two attentions:
   1. **Masked** self-attention on the inputs
   2. **Cross-attention** with the encoded source
2. Second attention module:
   $Q = X_{\text{dec\_sublayer}}$, $K = Y_{\text{enc}}$, $V = Y_{\text{enc}}$;
3. The decoder predicts the input shifted by one (linear and softmax)



Transformer Embeddings

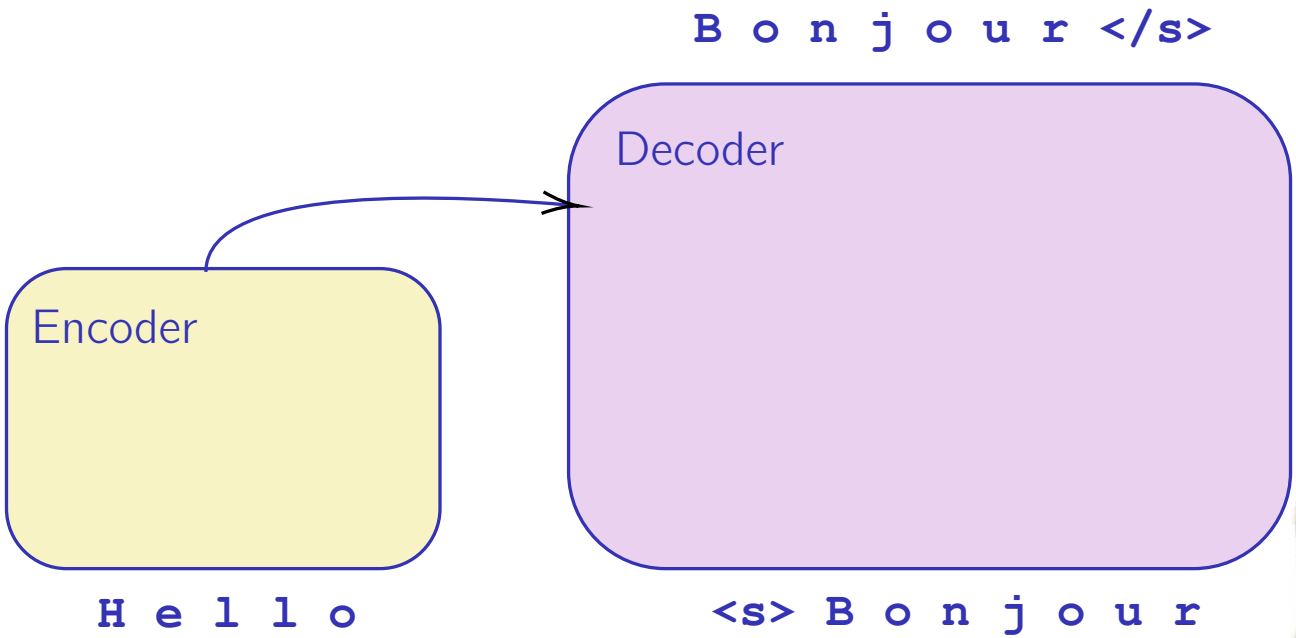
In Vaswani's paper, the input and output embeddings and the last linear module share the same matrix (weights)
Same with BERT in the pretraining step for the input embeddings and last linear layer.
See code: `https://github.com/google-research/bert/blob/master/run_pretraining.py#L240`)



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 score

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

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

	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.

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

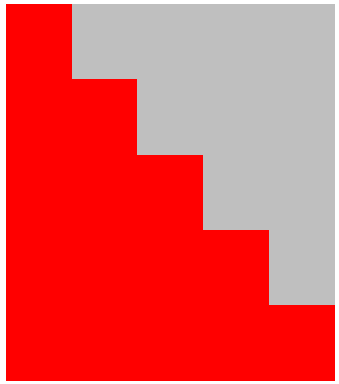
For *ship:*

```
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
This a matrix to prevent a look ahead
We just replace the gray cells with -10^9 and keep the original values of the red cells



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

Code Example

Experiments:

- Jupyter Notebook: `ch13_attention_masked.ipynb`
- 6th laboratory on machine translation

PyTorch Decoders

PyTorch has a class to create an 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):

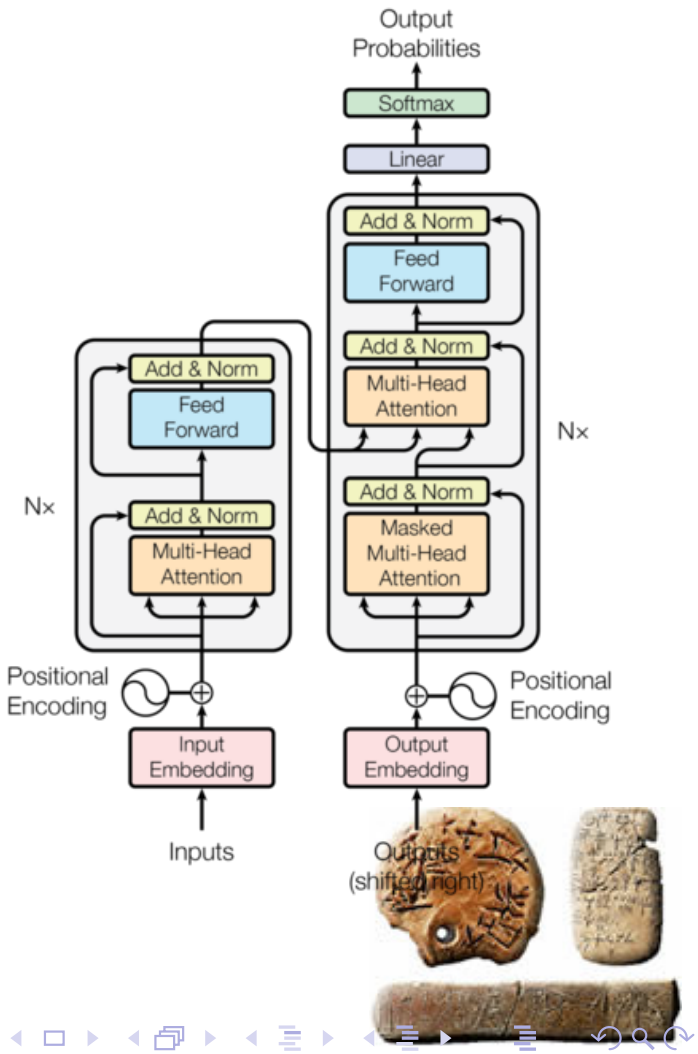
```
decoder = nn.TransformerDecoder(decoder_layer, num_layers)
```

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

Autoregressive Models

Reference papers:

1. *Improving Language Understanding by Generative Pre-Training* by Radford et al. (2018)
   Link: `https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf`
2. *Language Models are Unsupervised Multitask Learners* by Radford et al. (2018) Links:
   `https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf`
   `https://github.com/openai/gpt-2`



Transformer Decoder Training (I)

Pretrained as an autoregressive language model:

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

Fine-tuning applications in the initial paper

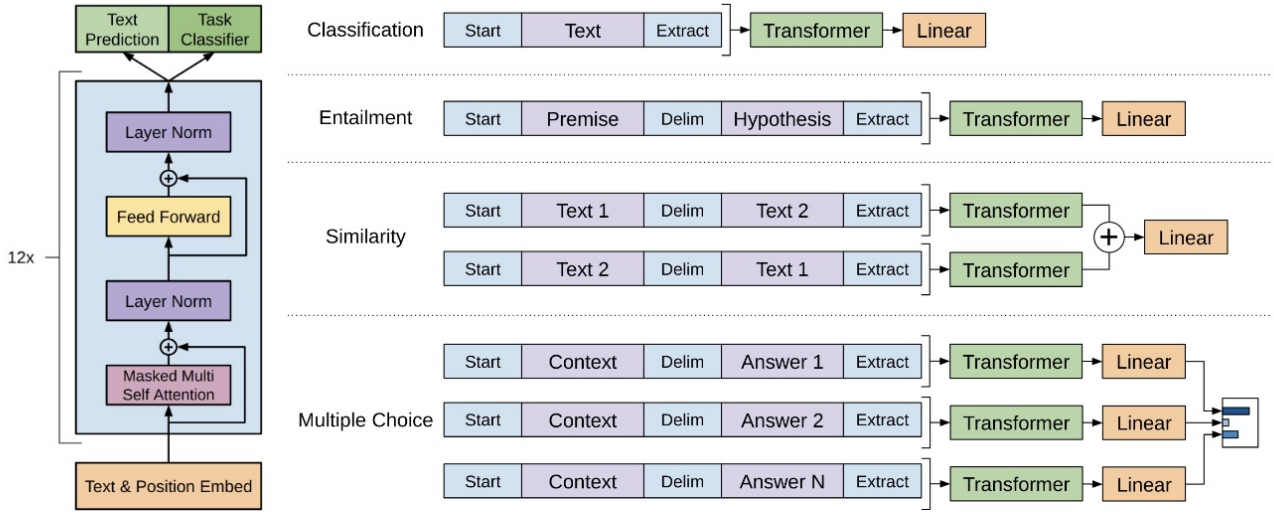


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

Note there is no cross-attention

Transformer Decoder Training (II)

The second system 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

Fine-Tuning Chat Decoders

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

- Uses an autoregressive pretrained decoder model
- Fined-tuned on instructions with the format: Instruction, Input, Output

Large companies often apply supervised fine-tuning, see
`https://arxiv.org/pdf/2307.09288.pdf`
Instructions could be tedious to write though
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

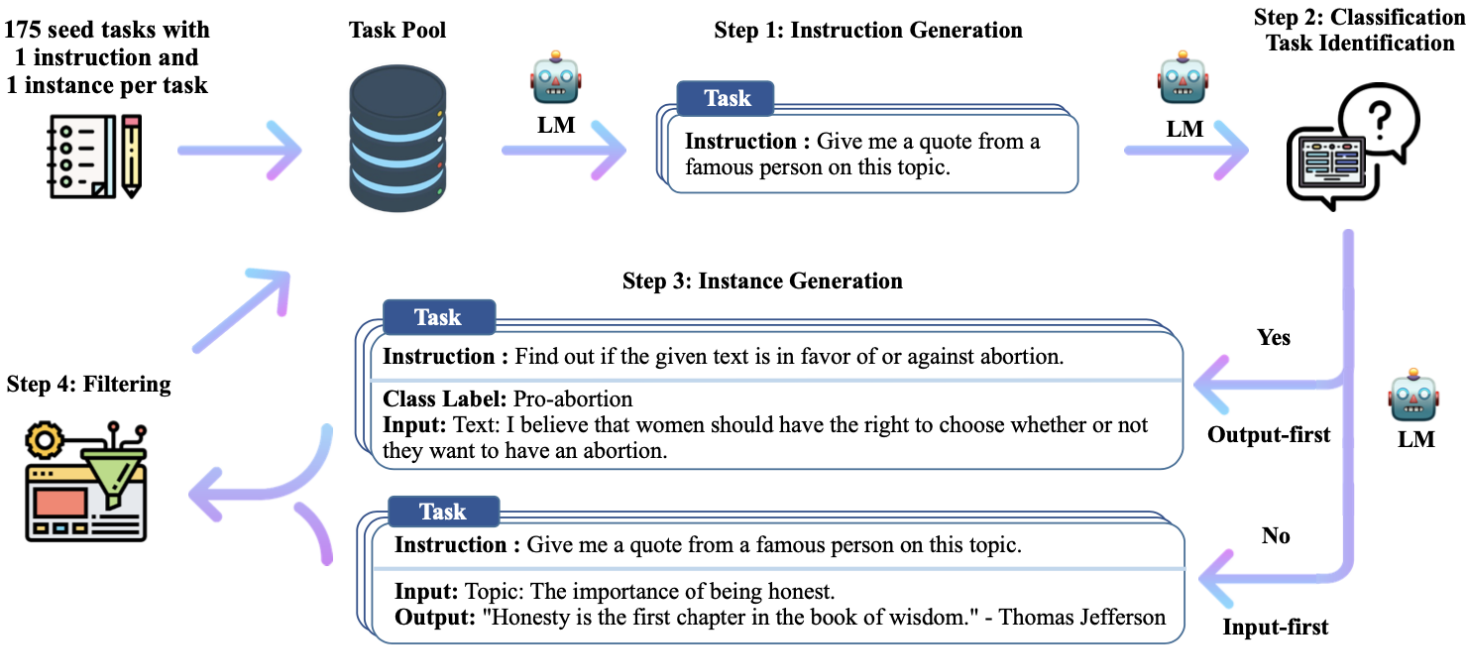


Figure 2: A high-level overview of SELF-INSTRUCT. The process starts with a small seed set of tasks as the task pool. Random tasks are sampled from the task pool, and used to prompt an off-the-shelf LM to generate both new instructions and corresponding instances, followed by filtering low-quality or 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