HANDS-ON MACHINE LEARNING (HOME)

Lecture/Lab – 1

An Introduction to Machine Learning Using LLMs as an Example

Goal: Run an LLM (Mistral 7B or Llama 2) on a local machine and chat with it

Ghulam Rasool, PhD

Department of Machine Learning, Moffitt Cancer Center

HANDS-ON MACHINE LEARNING (HOME)

- Lectures / Labs (Coding)
  - Once a month or after 6 weeks.
  - Mixed contents - labs and lectures
- Target Audience
  - Anyone interested in learning about AI/ML
  - No specific background or skill set is required
  - It is OK if you do not want to code, you will still learn

HANDS-ON MACHINE LEARNING (HOME)

- What is HOME?
  - Hands-on
  - Current machine learning topics, e.g., LLMs, VLM, MML
  - Modern tools
  - Discussions around how we can use these modern tools and methods to solve our day-to-day problems
- What HOME is not
  - ~~State-of-the-art from the ML/research community~~
  - ~~My research~~
  - ~~Review of AI/ML models or algorithms~~

HANDS-ON MACHINE LEARNING (HOME)

- Lecture/Lab – 1
  - An Introduction to Machine Learning Using LLMs as an Example
  - Terminology
  - Platforms
    - LM Studio, Hugging Face, VS Code
  - LangChain
  - APIs
- Lecture/Lab – 2
  - Prompt Engineering
  - Retrieval-Augmented Generation (RAG)
- Lecture/Lab 3
  - Fine-tune LLM on local machine

LARGE LANGUAGE MODELS

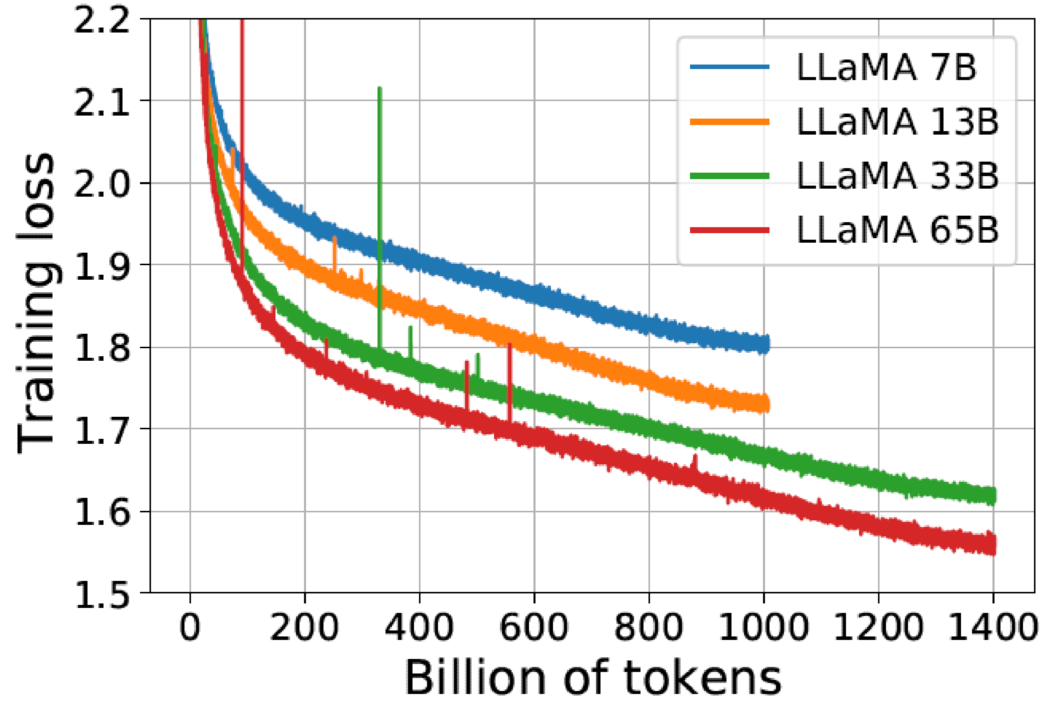
- Why Large?
- Brief History
  - Transformers – Attention - 2018
  - Bidirectional Encoder Representations from Transformers (BERT) - 2018
    - Masked language model (MLM) Task
    - The "next sentence prediction" task
  - Generative Pre-trained Transformer - 1(GPT-1) - 2018
    - Improving language understanding with unsupervised learning
    - Using "transformers," "unsupervised pre-training," and "supervised fine-tuning"
  - GPT-2 - 2019
  - GPT-3 - 2020
  - InstructGPT and many others - 2022

LLAMA AND MISTRAL

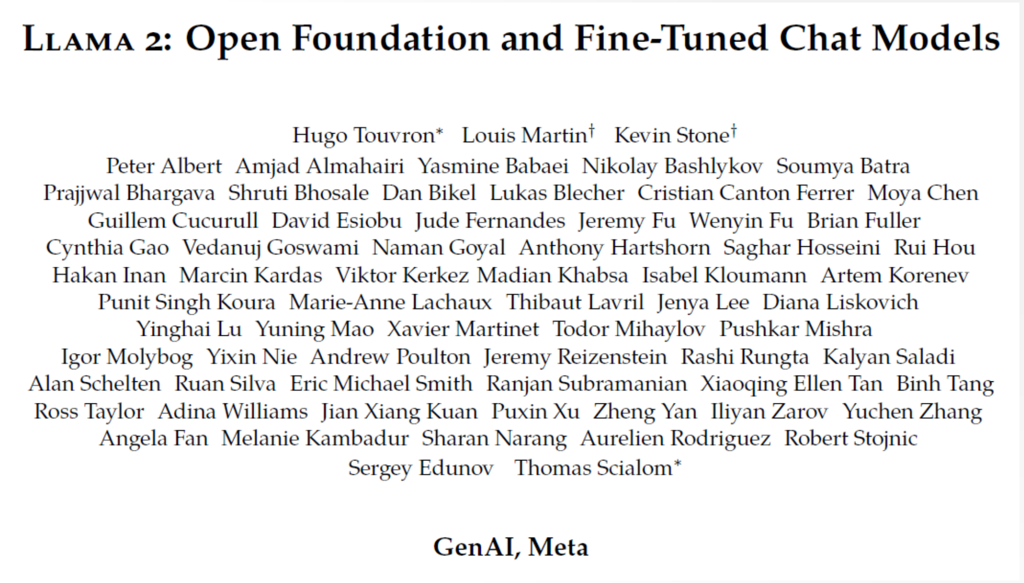
- GPTs - 2018 onwards
- Large Language Model Meta AI (LLaMA)
  - Llama-1 - Feb 2023
  - Llama-2 - Jul 2023
- Mistral AI - Oct 2023

LLAMA-1

- Datasets
  - English CommonCrawl, C4, GitHub, Gutenberg and Books3, ArXiv, Stack Exchange
- Tokenizer
  - Byte-pair encoding (BPE) - Search for the most frequent pair of existing tokens. That most frequent pair is the one that will be merged, and we will rinse and repeat for the next step.

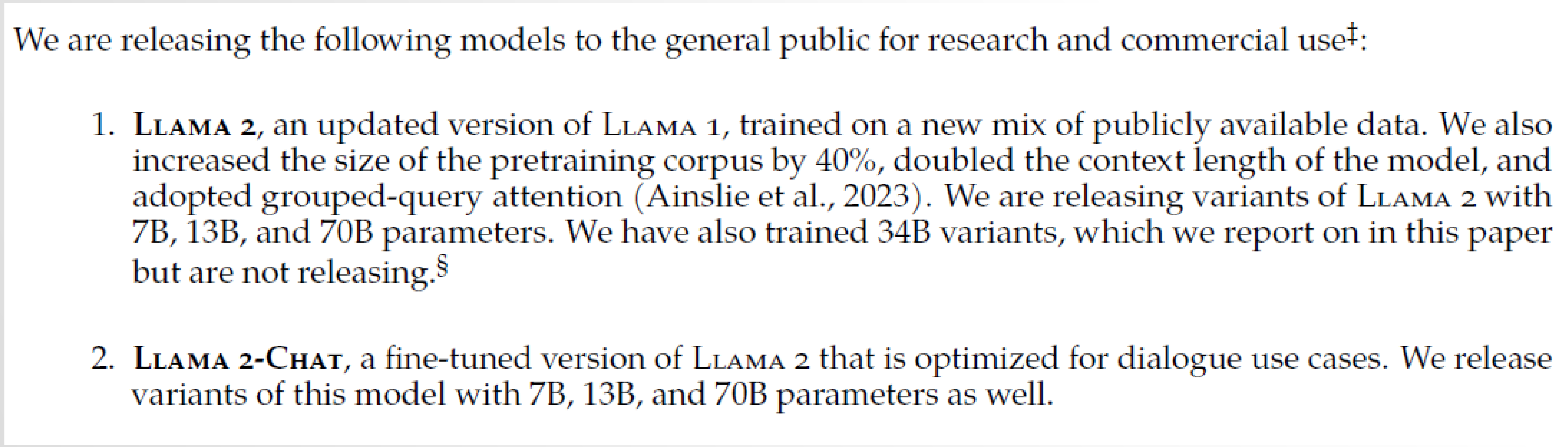


LLAMA-2


Llama 2: Open Foundation and Fine-Tuned Chat Models

Hugo Touvron[*] Louis Martin[†] Kevin Stone[†]
Peter Albert Amjad Almahairi Yasmine Babaei Nikolay Bashlykov Soumya Batra
Prajjwal Bhargava Shruti Bhosale Dan Bikel Lukas Blecher Cristian Canton Ferrer Moya Chen
Guillem Cucurull David Esiobu Jude Fernandes Jeremy Fu Wenyin Fu Brian Fuller
Cynthia Gao Vedanuj Goswami Naman Goyal Anthony Hartshorn Saghar Hosseini Rui Hou
Hakan Inan Marcin Kardas Viktor Kerkez Madian Khabsa Isabel Kloumann Artem Korenev
Punit Singh Koura Marie-Anne Lachaux Thibaut Lavril Jenya Lee Diana Liskovich
Yinghai Lu Yuning Mao Xavier Martinet Todor Mihaylov Pushkar Mishra
Igor Molybog Yixin Nie Andrew Poulton Jeremy Reizenstein Rashi Rungta Kalyan Saladi
Alan Schelten Ruan Silva Eric Michael Smith Ranjan Subramanian Xiaoqing Ellen Tan Binh Tang
Ross Taylor Adina Williams Jian Xiang Kuan Puxin Xu Zheng Yan Iliyan Zarov Yuchen Zhang
Angela Fan Melanie Kambadur Sharan Narang Aurelien Rodriguez Robert Stojnic
Sergey Edunov Thomas Scialom[*]

GenAI, Meta

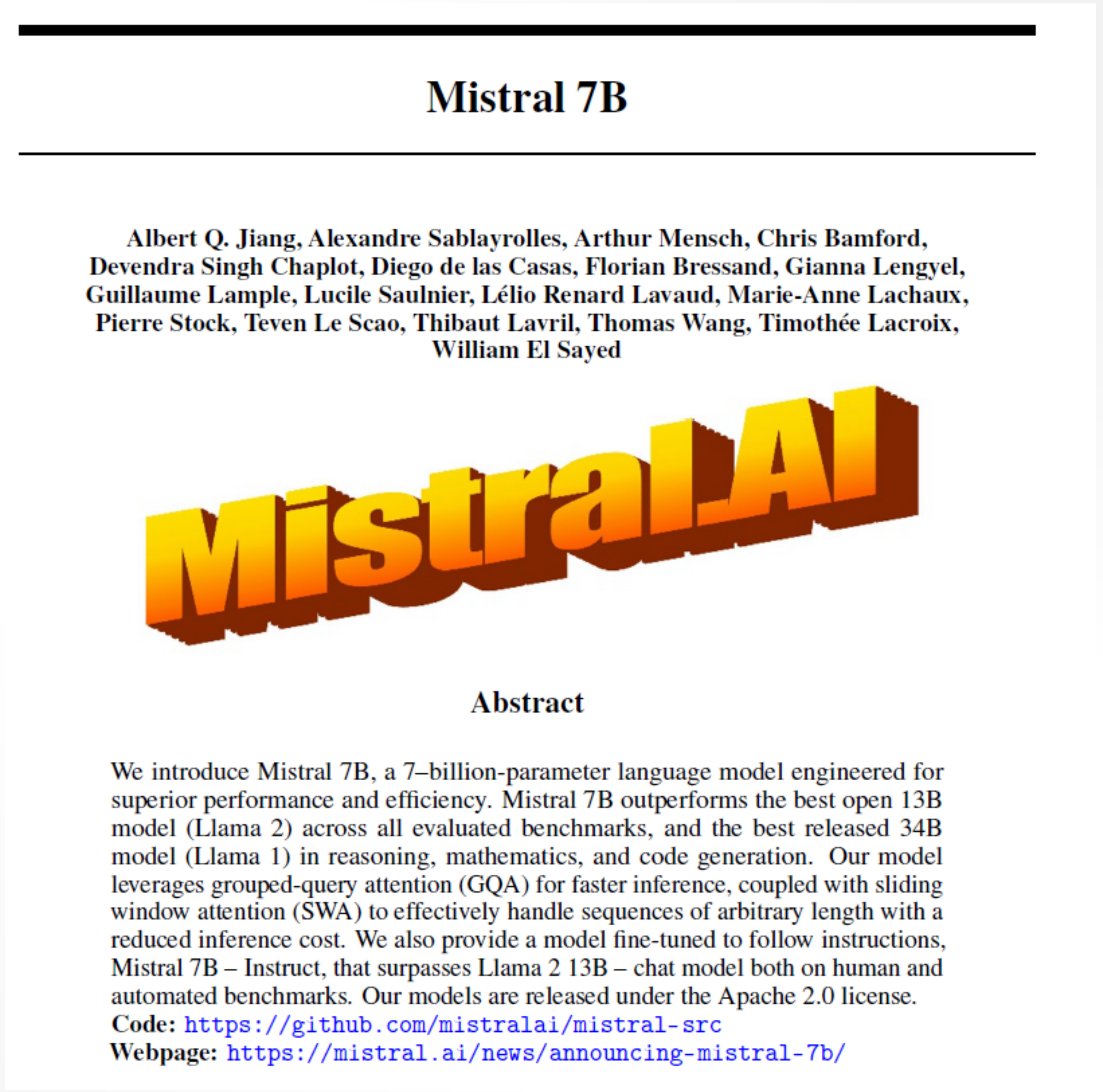

We are releasing the following models to the general public for research and commercial use[‡]:

1. **Llama 2**, an updated version of Llama 1, trained on a new mix of publicly available data. We also increased the size of the pretraining corpus by 40%, doubled the context length of the model, and adopted grouped-query attention (Ainslie et al., 2023). We are releasing variants of Llama 2 with 7B, 13B, and 70B parameters. We have also trained 34B variants, which we report on in this paper but are not releasing.[§]

2. **Llama 2-Chat**, a fine-tuned version of Llama 2 that is optimized for dialogue use cases. We release variants of this model with 7B, 13B, and 70B parameters as well.

MISTRAL

- 7.3B parameter model
- Outperforms Llama 2 13B on all benchmarks
- Outperforms Llama 1 34B on many benchmarks
- Approaches CodeLlama 7B performance on code while remaining good at English tasks
- Uses Grouped-query attention (GQA) for faster inference
- Uses Sliding Window Attention (SWA) to handle longer sequences at a smaller cost



Mistral 7B

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, William El Sayed

Mistral.AI

Abstract

We introduce Mistral 7B, a 7–billion-parameter language model engineered for superior performance and efficiency. Mistral 7B outperforms the best open 13B model (Llama 2) across all evaluated benchmarks, and the best released 34B model (Llama 1) in reasoning, mathematics, and code generation. Our model leverages grouped-query attention (GQA) for faster inference, coupled with sliding window attention (SWA) to effectively handle sequences of arbitrary length with a reduced inference cost. We also provide a model fine-tuned to follow instructions, Mistral 7B – Instruct, that surpasses Llama 2 13B – chat model both on human and automated benchmarks. Our models are released under the Apache 2.0 license.
Code: https://github.com/mistralai/mistral-src
Webpage: https://mistral.ai/news/announcing-mistral-7b/

LARGE LANGUAGE MODELS

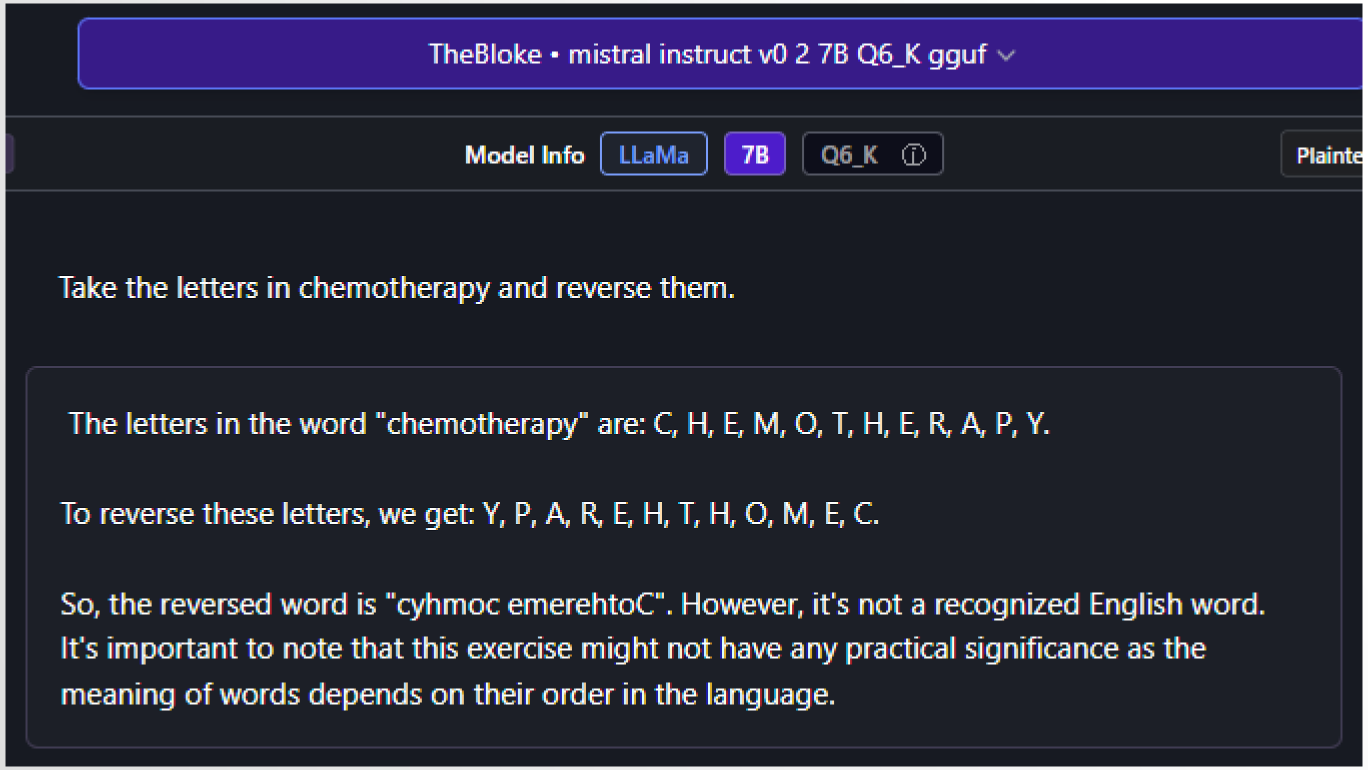
- Text Pre-Processing
  - Tokenization
  - Embeddings
- Local Installation and Usage – without and with coding (APIs)
- Prompt Engineering
  - Manual
  - Using libraries such as LangChain
  - Retrieval Augmented Generation (RAG)
    - Using local documents with vector databases and open-source LLMs
- Local Fine-tuning

WHAT IS NATURAL LANGUAGE PROCESSING (NLP)?

- NLP is a field of linguistics and machine learning
- Common NLP tasks
  - Classifying whole sentences
  - Classifying each word in a sentence
  - Generating text content
  - Extracting an answer from a text
  - Generating a new sentence from an input text
- Computers don’t process information in the same way as humans.
- The text must be processed to enable the model to learn from it.

https://huggingface.co/learn/nlp-course/chapter1/2

TOKENIZATION



Take the letters in chemotherapy and reverse them.

The letters in the word "chemotherapy" are: C, H, E, M, O, T, H, E, R, A, P, Y.

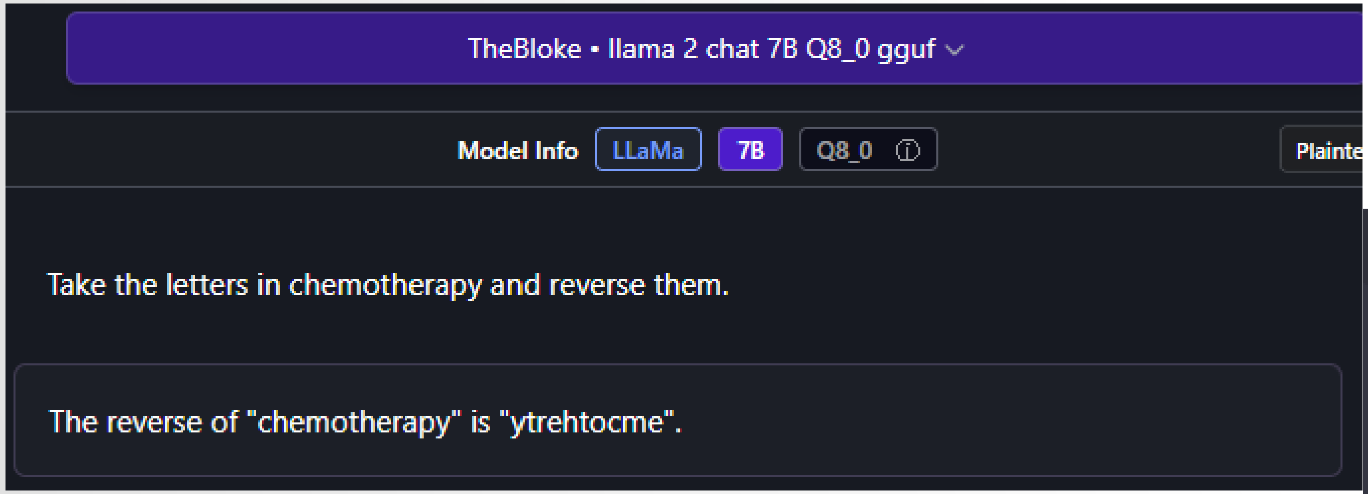
To reverse these letters, we get: Y, P, A, R, E, H, T, H, O, M, E, C.

So, the reversed word is "cyhmoc emerehtoC". However, it's not a recognized English word. It's important to note that this exercise might not have any practical significance as the meaning of words depends on their order in the language.

anthropomorphize

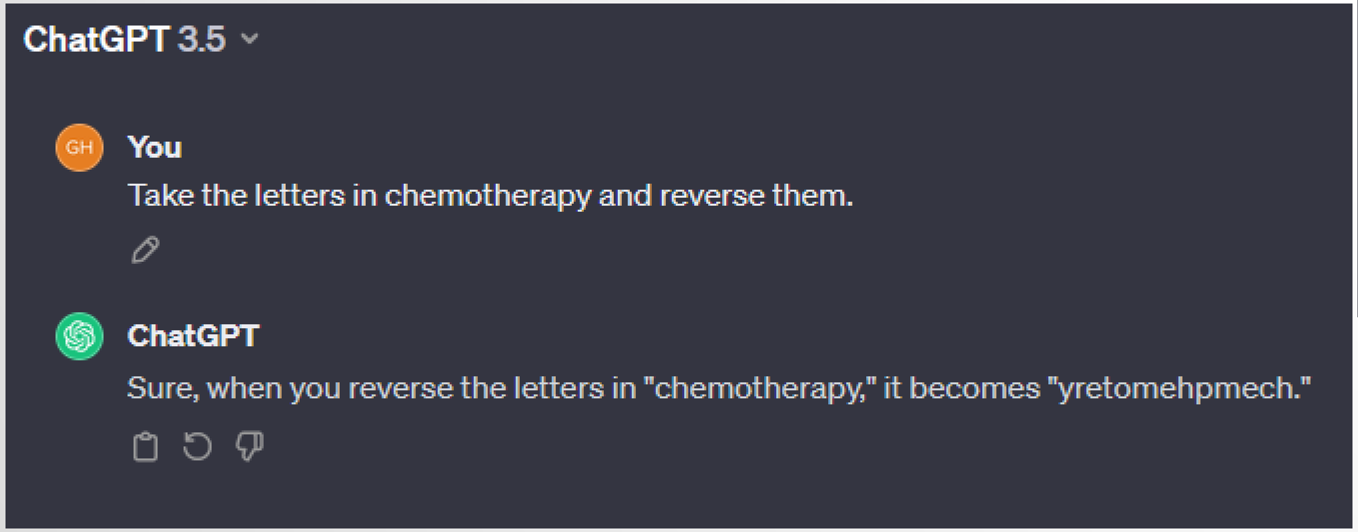
attribute human characteristics or behavior to (a god, animal, or object).

TOKENIZATION



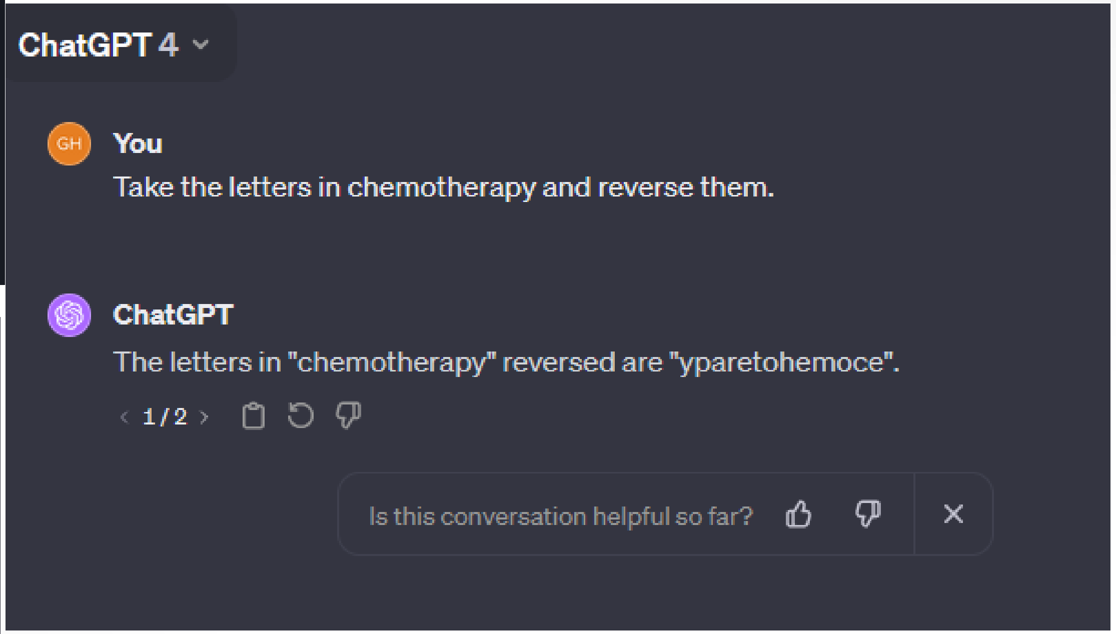
Take the letters in chemotherapy and reverse them.

The reverse of "chemotherapy" is "ytrehtocme".



You
Take the letters in chemotherapy and reverse them.

ChatGPT
Sure, when you reverse the letters in "chemotherapy," it becomes "yretomehpmech."



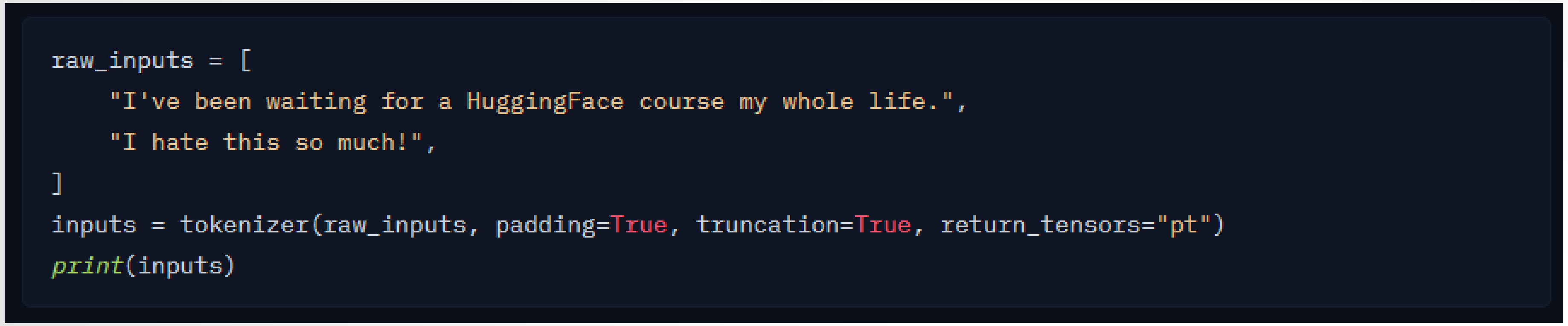
You
Take the letters in chemotherapy and reverse them.

ChatGPT
The letters in "chemotherapy" reversed are "yparetohemoce".

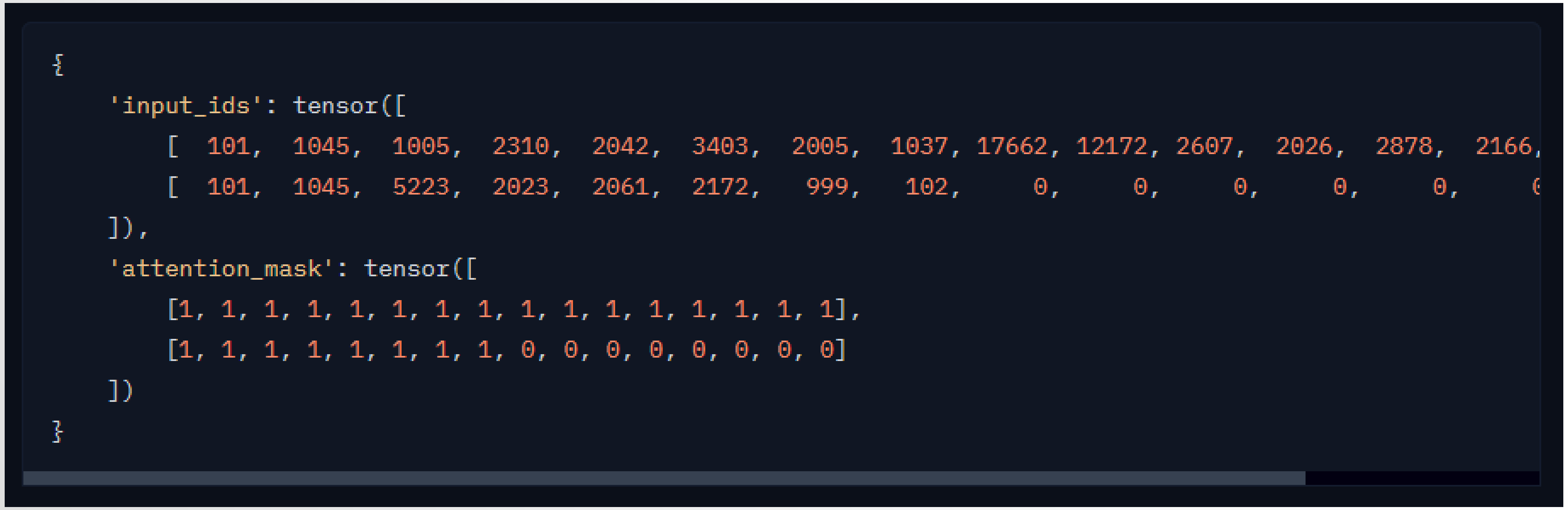
TOKENIZATION

- ML models cannot process raw text directly
- The first step is to convert the text inputs into numbers.
- Steps
  - Splitting the input into **words**, **subwords**, or symbols (like punctuation) that are called **tokens**
  - ~ 4 characters per token, ~ ¾ of an average word
  - Mapping each token to an integer
  - Adding additional inputs that may be useful to the model
  - Convert integer tokens into embeddings (which are vectors of real numbers)

TOKENIZATION



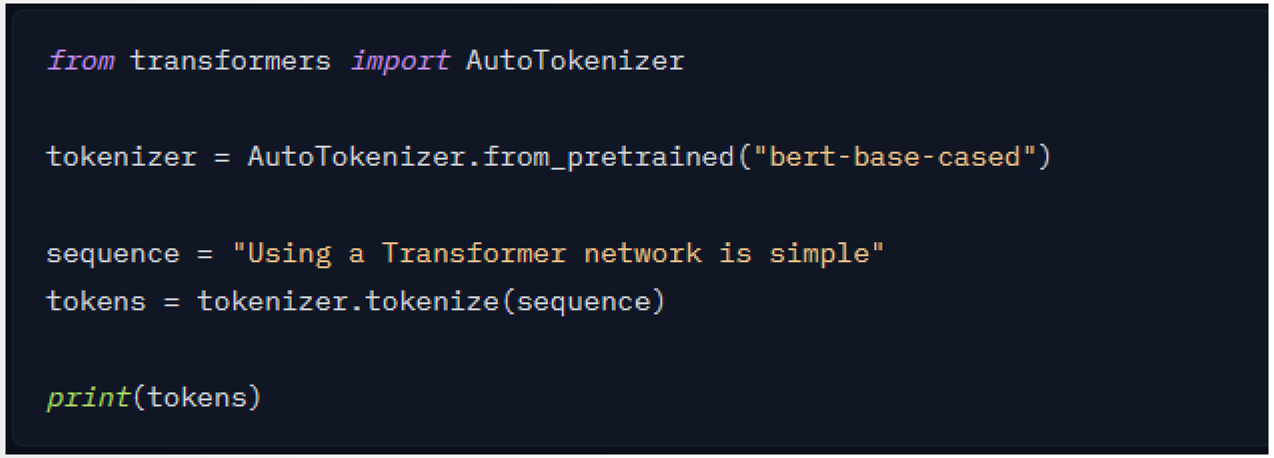
```
raw_inputs = [
    "I've been waiting for a HuggingFace course my whole life.",
    "I hate this so much!",
]
inputs = tokenizer(raw_inputs, padding=True, truncation=True, return_tensors="pt")
print(inputs)
```



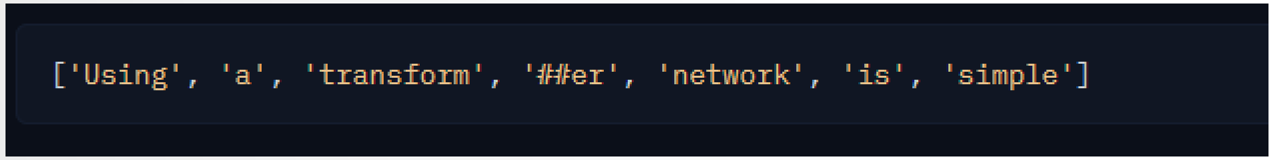
```
{
    'input_ids': tensor([
        [  101,  1045,  1005,  2310,  2042,  3403,  2005,  1037, 17662, 12172, 2607,  2026,  2878,  2166,
        [  101,  1045,  5223,  2023,  2061,  2172,   999,   102,     0,     0,     0,     0,     0,
    ]),
    'attention_mask': tensor([
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
    ])
}
```

Examples from the Hugging Face NLP Course

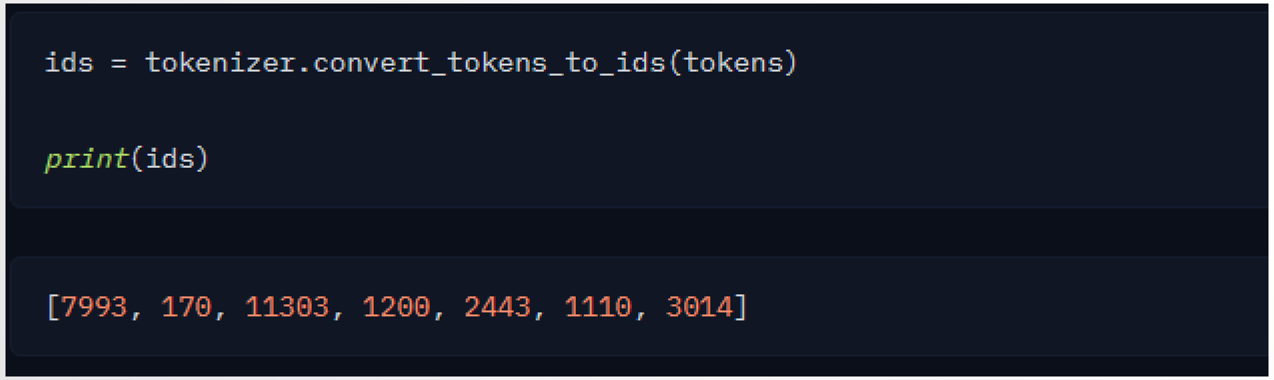
TOKENIZATION



```python
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

sequence = "Using a Transformer network is simple"
tokens = tokenizer.tokenize(sequence)

print(tokens)
```



```
['Using', 'a', 'transform', '##er', 'network', 'is', 'simple']
```

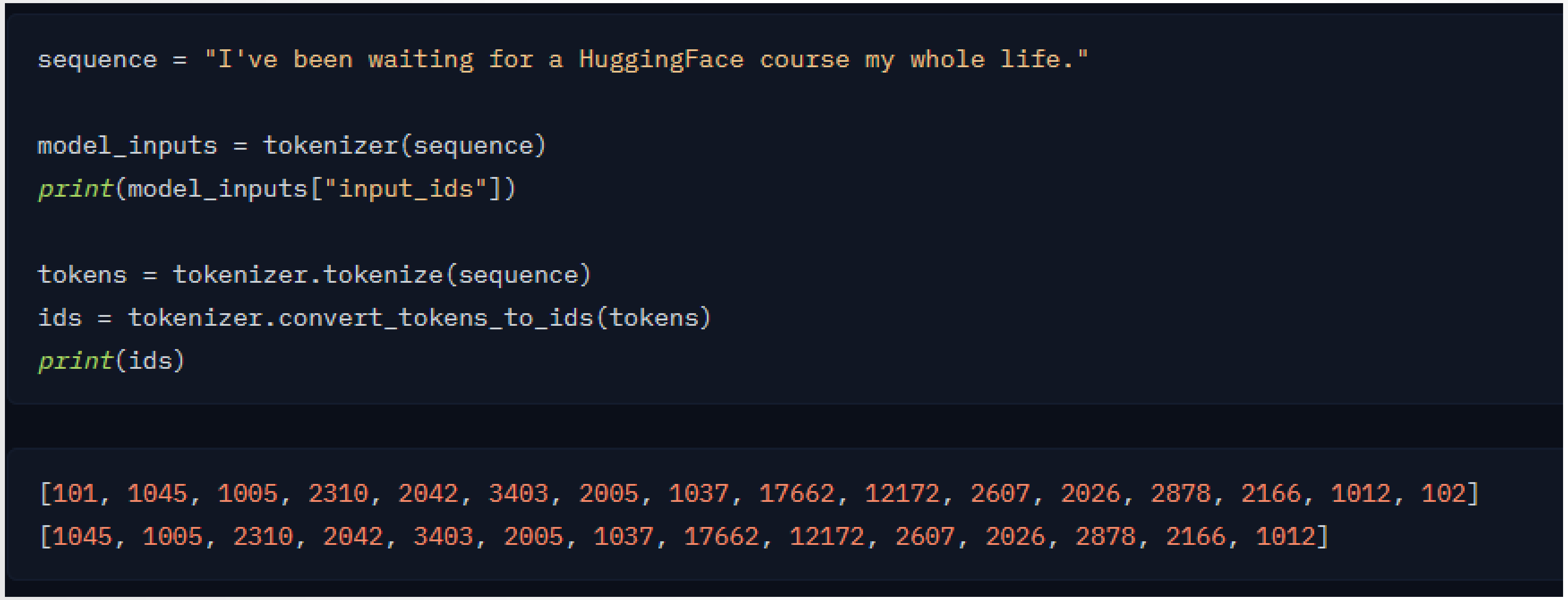


```python
ids = tokenizer.convert_tokens_to_ids(tokens)

print(ids)
```

```
[7993, 170, 11303, 1200, 2443, 1110, 3014]
```

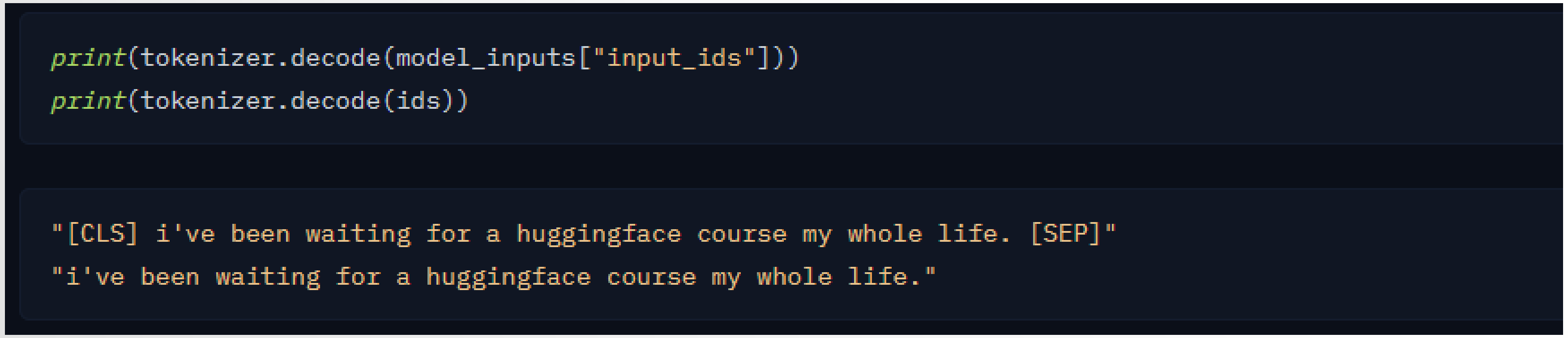
Examples from the Hugging Face NLP Course

TOKENIZATION



```python
sequence = "I've been waiting for a HuggingFace course my whole life."

model_inputs = tokenizer(sequence)
print(model_inputs["input_ids"])

tokens = tokenizer.tokenize(sequence)
ids = tokenizer.convert_tokens_to_ids(tokens)
print(ids)
```

```
[101, 1045, 1005, 2310, 2042, 3403, 2005, 1037, 17662, 12172, 2607, 2026, 2878, 2166, 1012, 102]
[1045, 1005, 2310, 2042, 3403, 2005, 1037, 17662, 12172, 2607, 2026, 2878, 2166, 1012]
```



```python
print(tokenizer.decode(model_inputs["input_ids"]))
print(tokenizer.decode(ids))
```

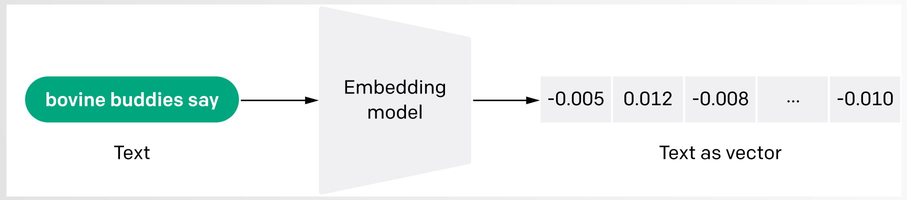
```
"[CLS] i've been waiting for a huggingface course my whole life. [SEP]"
"i've been waiting for a huggingface course my whole life."
```

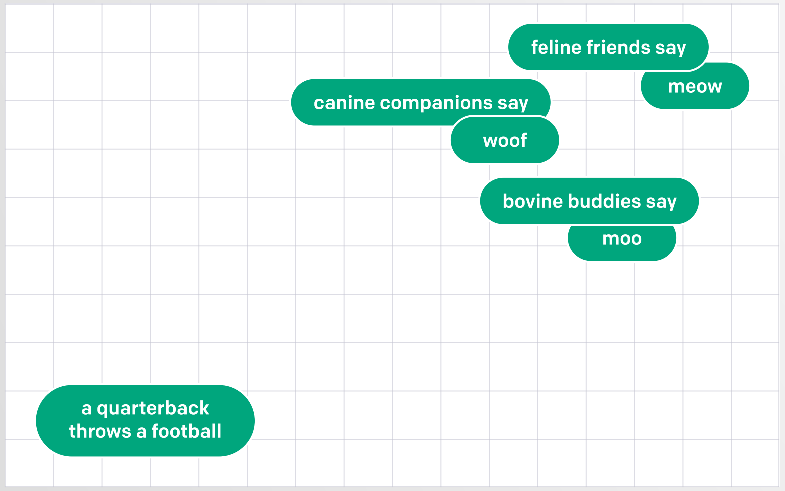
https://huggingface.co/learn/nlp-course/chapter2/2

Examples from the Hugging Face NLP Course

EMBEDDINGS

- Embeddings are numerical representations of concepts converted to number sequences, making it easy for computers to understand the relationships between them.





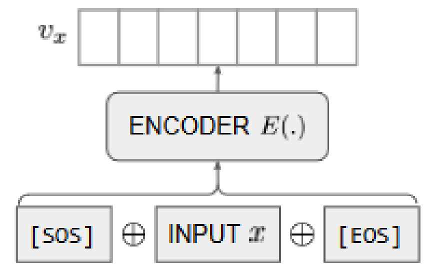


Figure 2. The encoder E maps input x to embedding v_x. Special tokens, [SOS] and [EOS], are appended to the start and end of the input sequence respectively. The last layer hidden state corresponding to the token [EOS] is extracted as the embedding of the input sequence.

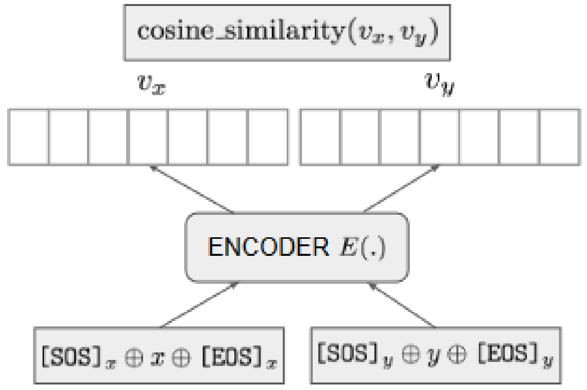
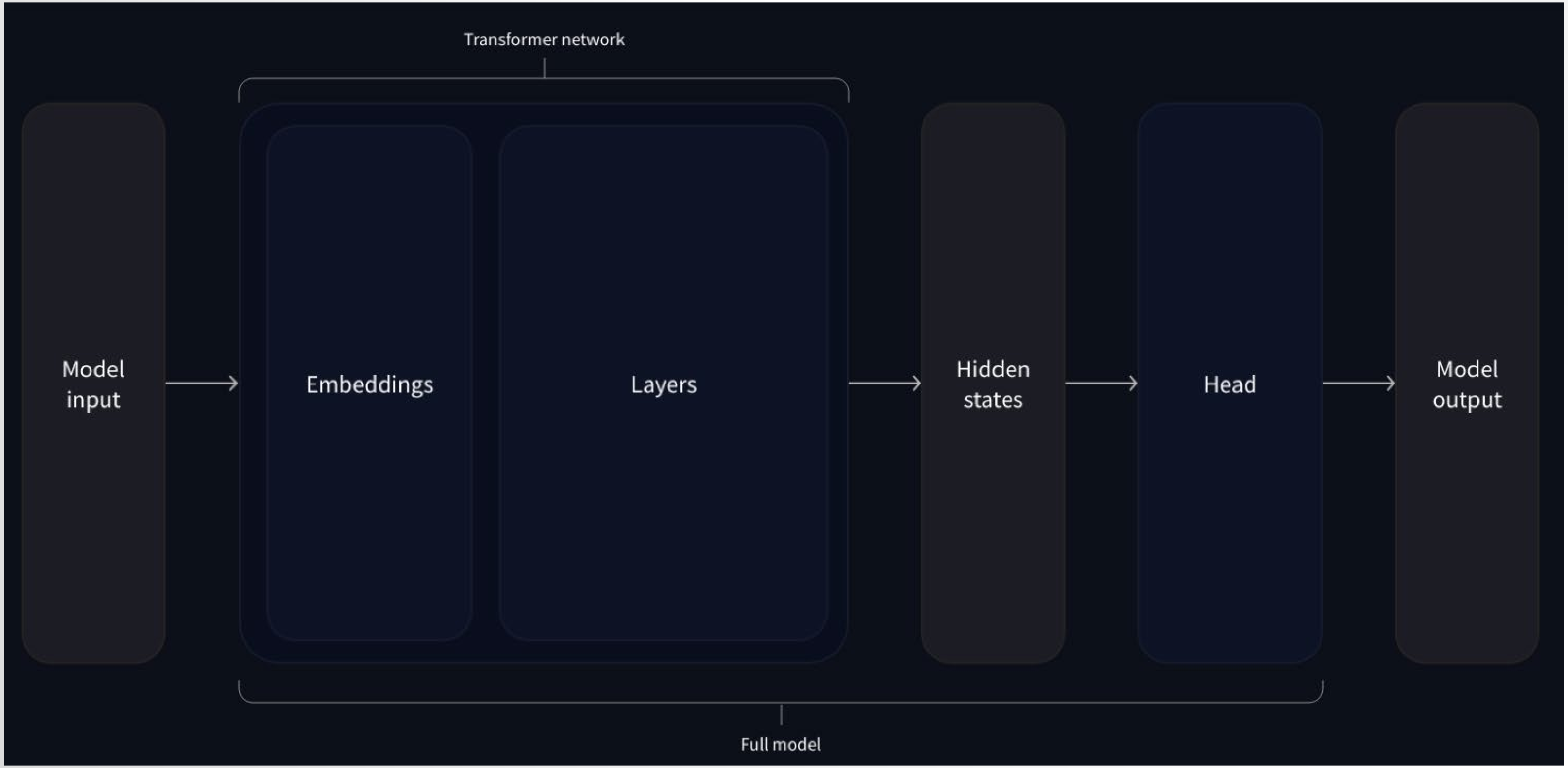

cosine_similarity(v_x, v_y)
v_x
v_y
ENCODER $E(.)$
$[\text{SOS}]_x \oplus x \oplus [\text{EOS}]_x$
$[\text{SOS}]_y \oplus y \oplus [\text{EOS}]_y$

Figure 3. The encoder E maps inputs x and y, to embeddings, v_x and v_y independently. The similarity score between x and y is defined as the cosine similarity between these two embedding vectors.

EMBEDDINGS

- **GloVe (Global Vectors for Word Representation)**
  - These are pre-trained word vectors by Stanford, based on aggregating global word-word co-occurrence matrices from a corpus and then performing dimensionality reduction.
- **Word2Vec**
  - Developed by Google, it uses neural networks to learn word associations from a large text corpus. It has two architectures: Continuous Bag-of-Words (CBOW) and Skip-Gram.
- **FastText**
  - Created by Facebook, FastText extends Word2Vec to consider sub-word information (like morphemes), allowing it to handle out-of-vocabulary words better.

TOKENIZATION



RUNNING AN LLM ON THE LOCAL MACHINE

- Instruction
  - https://github.com/grasool/HOME
- LM Studio (No coding)
  - https://lmstudio.ai/
- Using API to talk to your LLM programmatically (coding)
  - VS Code - https://code.visualstudio.com/
  - LangChain - https://www.langchain.com/