Exercise Sheet #8

Problem 1 (*Whitespace Tokenizer*)

The first step in NLP (**N**atural **L**anguage **P**rocessing) is always to tokenize the natural language input. This step of NLP is equally very annoying and very important, as many problems can arise from poor tokenization. In this problem we want to build our own tiny tokenizer. Consider as an example the tiny-Goethe dataset

https://itp.uni-frankfurt.de/~nevermann/teaching/goe_full.txt

Sophisticated tokenizers are a bit cumbersome to implement, thus we take the easy route and work with a whitespace tokenizer.

(a) Implement a whitespace tokenizer function that ignores punctuation and newlines, i.e. for a given input text, the function should remove punctuation (`.,;:?!` etc.) and newline characters (`\n`) and return a list of the words.

(b) For the Goethe dataset, find all unique word tokens and generate lookup tables that assign an integer to a token and vice versa.

(c) Implement an encode function and a decode function that encodes a text to a list of integer tokens and decodes a list of integer tokens back to a text, respectively.

(d) Test encoder and decoder with a funny (lol) example sentence, such as

```
Wir müssen Faust finden und ihm sagen, dass wir ihn ab
sofort Hand nennen.
```

(e) Why is this tokenizer suboptimal?

Problem 2 (*Tokenizers: How the Pros do it*)

Most LLMs, such as ChatGPT, tokenize texts based on subwords. The tokenizer used in ChatGPT models is called *tiktoken*, an open-source project by OpenAI. You can find the repository here.

(a) Play around with tiktoken and compare with the tokenizer you implemented in Problem 1.

(b) A web app is available for tiktoken which visualizes the inner workings of the tokenizer.

`https://tiktokenizer.vercel.app/`

Play around with the web app. Do you recognize patters where a single token might be a full word and where it might only be a subword? Experiment with different input texts. Does the token representation depend on capitalization?