Project #4
Gotta get good Goethe

Deadline: 03.02.2025, 12:00h

🎉 **`Happy new year!`**

GPTs (**G**enerative **P**re-trained **T**ransformers) are seemingly everywhere nowadays. While they can seem like magic, the fundamental principles of transformer-based language models are not very hard to implement.
In this project, you will create your own transformer from scratch. Since you are hopefully very devoted to your university and probably can't get enough of Goethe either, you will be happy to hear that the project consists of writing a language model trained to generate text in the style of Goethe's works.

Training data

To train your model, you will need Goethe, a lot of it. Thankfully, Project Gutenberg provides that. To save you some work, you can find a suitable training data file here:

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

This file contains texts from various of Goethe's works concatenated into a single file.[1]
As a first step, clean the data of interpunction and train the model with this simplified dataset.
To validate your trained model, it is advisable to split the full dataset into a training dataset (e.g. 90%) and a validation dataset (e.g. 10%).

Model

In Fig. 1 you find a schematic of the model architecture you should implement. A few aspects are detailed below.

[1] If you encounter problems with umlauts in your browser, try downloading the file using `wget`.

Tokenization LLMs like ChatGPT use complex tokenization techniques, where a token is often a whole word or a subword in the input sequence. This leads to large vocabulary sizes. For this project it suffices to consider a character level language model, where each token represents a single letter.
Hint: For the given input file, the vocabulary size is by my counting 109.

Embedding Choose a suitable embedding. You might consider an embedding table with trainable weights, however, a one-hot embedding should also suffice. Regarding the positional embedding, you might use an encoding of the absolute positions. Another simple option to consider is ALiBi, see the lecture for details.

Transformer block Implement a transformer block as you saw in lecture. You *can* implement everything yourself, relying minimally on PyTorch's predefined modules, but you are also allowed to use everything that's available in PyTorch for your implementation.
Keep in mind, that the end goal should be text generation. Thus, tokens should only attend to tokens before them, thereby obeying causality.

Output Your trained model should be capable of generating text from a small initial text snippet. Ideally, this text should be German words and make sense, but it is likely that you will not reach that amount of accuracy.

Minimum requirements

Your project should fulfill some minimum requirements:

- Implement a GPT model that is trained on works of Goethe. Start with a dataset cleaned of interpunction (you may in addition want to consider all lowercase letters) and later use the full dataset. The trained model should be able to generate text in the style of Goethe. The text produced by the trained model should at least somewhat look like German. It is accepted if the generated text makes no sense or if a few words are gibberish.
- Your implementation should feature at least one of the following:
  - ALiBi positional encoding
  - a beam search variant from the lecture
  - expressive attention.
- Document your training efforts in a suitable format.

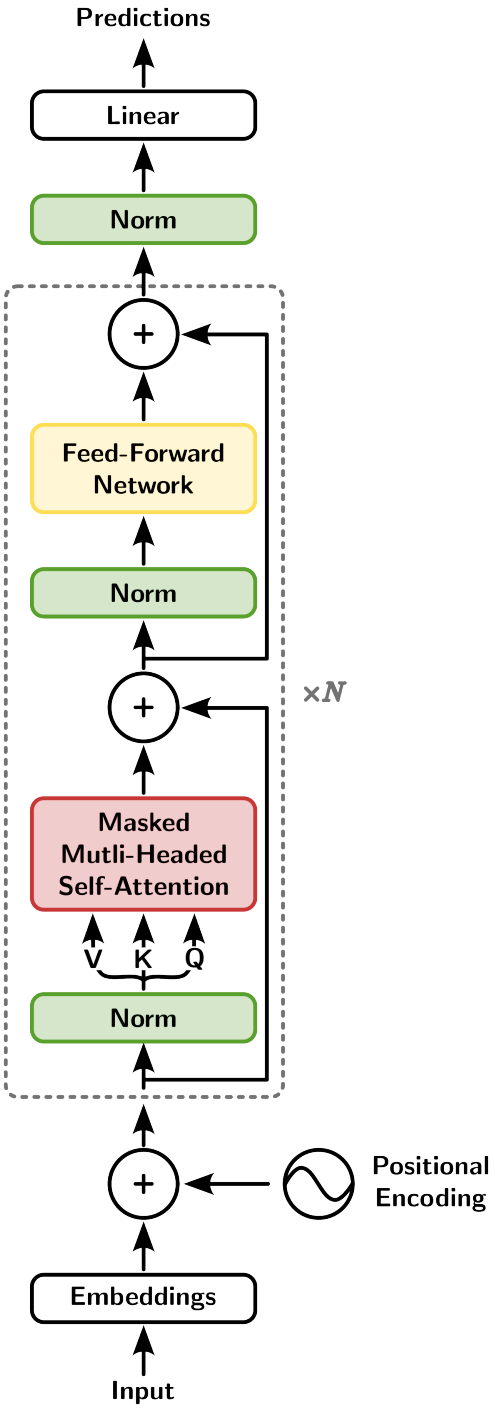

Figure 1: Schematic of the model architecture. The gray region comprises a transformer block.