

E1: Unimodal Language Representations

Prof. Dr. Anna Rohrbach,
Prof. Dr. Marcus Rohrbach,
Hector Garcia Rodriguez, M. Sc.
Jonas Henry Grebe, M. Sc.



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Deadline Wednesday 21st May, 2025

General Information

This exercise will give you practical experience with language modelling and basic language model architectures. For the coding tasks of this exercise, we will work with a Transformer Language Model (LM) and text from [The Lord of the Rings](#).

To hand in your solutions to this exercise, you will have to submit the following:

- Your report as a PDF named **E1_GROUP_<YOUR_GROUP_NUMBER>.pdf** containing your answers to the question part and any additional explanations, descriptions or discussions required for the coding part. **You must clearly indicate which answers belong to which tasks in your submitted report.** Parts of your solution which are not clearly linked to a task won't be graded.

Please use \LaTeX and the minimal [TUDaReport](#) template to compile a PDF for the submission in the official TU Darmstadt design. The university provides [ShareLaTeX](#) to students that you can use to collaborate.¹ We do not provide customised templates for every exercise sheet.

- Additionally, a [Google Colab link](#) (with edit access to everyone) to your copy of the provided notebook has to be submitted to Moodle. Ensure that the results mentioned in your report are reproducible from the submitted checkpoint of your notebook.

Important: Before you share it, you have to save the checkpoint of your notebook, which gives it a time stamp and shows it in the revision history (File → Save and checkpoint), where you can rename it to **Final Submission (Group <YOUR_GROUP_NUMBER>)**. Every change after this submission checkpoint won't be graded. If there is no pinned revision that is marked as your submission, we will just grade the last existing checkpoint from before the deadline. *If you used Kaggle, please upload your notebook to Google Colab again to provide this link.*

In case of questions regarding the submission process, please use the **Admin Q&A** forum on Moodle.

¹You can adapt the `accentcolor` to our orange by replacing 9c with 8b in the `documentclass` command

To summarize the submission process:

1. Start a [TUDaReport](#) project on [ShareLaTeX](#) and invite your group members to collaborate on the report.
2. Ideally, meet in person to work on the coding part together.
3. Once you are done with the exercises, do the following to submit:
 - a) Checkpoint your current state of the Google Colab notebook (File → Save and checkpoint)
 - b) Go to File → Revision History and rename this checkpoint to **Final Submission (Group <YOUR_GROUP_NUMBER>)**.
 - c) Go to Share and create a link with edit permissions for everyone to your notebook.
 - d) Compile your \LaTeX report and rename it to **E1_GROUP_<YOUR_GROUP_NUMBER>.pdf**
 - e) Upload your report to Moodle and include your generated edit link to your notebook in the additional submission text.

Exercises

This exercise has a maximum score of 25 points: up to 7 points for the question part and 18 points for the coding part.

Note: GPU acceleration is not required for Tasks 1–3. To avoid unnecessary use and potential penalties from Google Colab from not using assigned GPUs, it is recommended to keep the GPU disabled until Task 4. This also helps conserve your GPU hours.

Question Part (7 Pts.)












Answer the following questions before and during your work on the coding part of this exercise:

1. 🍌 Explain what a *token* is and why *tokenization* is used in language modelling. (1 Pts.)
2. 🍌 Explain subword tokenization and why it could be a better tokenization strategy than character-level tokenization. Then contrast them by naming one advantage and one disadvantage for each of them. (3 Pts.)
3. 🍌 Explain the differences between encoder-only, decoder-only, and encoder-decoder Transformer models. (2 Pts.)
4. 🍌 Is the model provided in the coding part an encoder-only, decoder-only, or encoder-decoder model? Explain why. (1 Pts.)

Coding Part (18 Pts.)

In Moodle, we provide the initial resource material to get you started quickly which you need to complete and extend for the tasks below. Please download the [training text file](#) and open the provided [Google Colab Starter Kit](#). Ensure that you uploaded the `lotr.txt` to the instance's directory.

1. **Tokenization** (6 Pts.): Take a look at the provided code and understand how the basic character-level tokenization class `CharTokenizer` is implemented. It treats every character that occurs in the data as its own token.
 - a) 🍌 Implement a `SubwordTokenizer` class with the same interface, which derives a tokenization process from subword frequencies and uses a greedy longest-match procedure during the encoding. Handle special characters like white spaces just the like any other character. The `SubwordTokenizer` class shall do the following during its *training*:
 - i. Start with a vocabulary V that contains only all occurring characters (like the `CharTokenizer`).

- ii. Then repeat until `len(vocab)` is equal to the wanted `vocab_size`:
 - A. Compute frequencies of token pairs over V in the text
 - B. Take the most frequent pair of tokens
 - C. Merge them into a new token and update the vocabulary V
 - iii. Build a simple [prefix tree \(Trie\)](#) after training to accelerate the search for the longest matching prefix token during encoding. This will modify the logic of your `encode(...)` method but significantly improve its performance. You may use the [pygtrie](#) package to implement the data structure.
2. **Model Architecture** (2 Pts.): The `TransformerBlock` and `TransformerModel` classes are provided but incomplete. It is your task to implement their forward methods.
- a)  Complete the `TransformerBlock` class to implement the forward pass with the following provided components:
 - A masked `MultiheadAttention` mechanism that uses a *causal mask* to prevent attending to future positions in the sequence.
 - A small feed-forward neural network applied after the attention layer.
 - Residual connections and `LayerNorm` applied after both.Ensure the *causal mask* is correctly constructed and applied during self-attention.
 - b)  Complete the `TransformerModel` class. Ensure to handle both, the default case of not having a specified position embedding (`pos_emb`) class, and the case of having one. You might want to get back to this after the next task.
3. **Position Embeddings** (3 Pt.): As you learned in the lecture, Transformers don't have any built-in notion of sequence order like RNNs or CNNs. We add position embeddings to the token embeddings to help the model understand the order of tokens in a sequence. These tell the model where each token is in the sequence.
- a)  Implement the `SinusoidalPositionEmbeddings` class, which was not presented in the lecture. It is the type of positional encoding used and described in [Attention is All You Need](#) (Vaswani et al. 2017).
 - b)  Learn about the downsides and strengths of this type of positional embedding. Briefly describe it and contrast it with a naive approach of learning a separate embedding for every sequence position.
 - c)   Implement a helpful visualisation of your implemented positional embeddings to support your words in (b). Be creative and include the result together with an explanation in your report.
4. **Training and Evaluation** (7 Pts.) Finally:
- a)  Train both tokenizers on the full training data: the initial `CharTokenizer` and your own `SubwordTokenizer`. Ensure that this experiment is reproducible in an extra cell of the notebook. After that, write code to inspect their learned vocabularies for (b).
 - b)  What are the most and least frequent tokens from the vocabularies that are used when the trained tokenizers to the training data again? What are the longest ones? Briefly describe and provide examples.
 - c)  Train and evaluate your final models (obtained with the two tokenizers), w/ and w/o the positional embeddings (four combinations). Make sure to have a reproducible cell in the notebook that re-runs these experiments. *As a reference, with your position embeddings and with the SubwordTokenizer, the perplexity with the default configuration should be around 50. Such a training run with the default configuration should take less than 5 minutes.*
 - d)  Compare and briefly discuss your findings. Support them, e.g. with a table and some samples.
 - e)  Discuss whether the provided perplexity evaluation reflects the generative abilities of your model.
 - f) **Optional** (Only Karma Points): Share your best LM-generated story about Gandalf in the **Content Q&A** forum thread that we created. Feel free to change the default configuration for that.