Final Report

**1. Introduction**

**Motivation/ Importance**

Embarking on an Erasmus semester offers a unique journey of personal and academic enrichment. Amidst the excitement of immersing ourselves in a foreign country, meeting new people, and encountering diverse experiences, we encountered a common challenge: the balancing act of managing time for daily necessities like grocery shopping and cooking. This challenge is particularly pronounced for students, who grapple with limited funds, time constraints, and often, limited culinary expertise.

Luckily, as Erasmus students this semester, we found ourselves in the course Natural Language Processing, an up and coming field in computer science, that holds the promise of equipping us with the tools needed to address the culinary conundrums encountered during our Erasmus studies. It was within the framework of our NLP course that the idea took root: why not leverage NLP techniques to streamline the process of meal preparation?

Motivated by this realization, we embarked on the journey to develop a Recipe Model as part of our semester group project. Our project not only offers a practical solution to our culinary dilemmas but also stands as a testament to the transformative potential of Natural Language Processing (NLP). Through our exploration of NLP techniques, we have discovered a powerful toolset that not only addresses the challenges of meal preparation during our Erasmus studies but also hints at a future where technology can revolutionize everyday tasks. Our endeavor underscores the promise of NLP in enhancing various aspects of our lives, paving the way for innovative solutions to common problems.

**Problem**

Advertisments, too long, text too long, need quick shopping list, ideas on what to cook based off limited funds,

**Fit with the course**

data base, seine Auflagen aufgreifen

**Summary of implementation and the results**

What transformers, what LM, and basic result

**2. Problem Definition and Algorithm**

2.1 Task Definition

BOIS: different tasks we attempted to tackle

2.2 Algorithm/ modell Definition

The journey to our final model involved us creating multiple versions attempting to derive the best model for our task at hand. In this chapter we will describe the general composition of our model and in the following chapter we will discuss pivotal points of the model that we altered in order to improve upon our initial version of the model.

1. Data Loading and Preprocessing

Our journey in developing the RM (Recipe Model) commenced with a meticulous approach to data acquisition and preprocessing. We obtained three distinct sources of recipe data: "recipes\_raw\_nosource\_ar.json," "recipes\_raw\_nosource\_epi.json," and "recipes\_raw\_nosource\_fn.json." These datasets were selected to encompass a wide array of cuisines and cooking styles, laying the foundation for a rich and varied recipe collection.

Upon acquiring the datasets, we initiated the data loading process using the pandas library in Python. Each dataset was meticulously read into a DataFrame structure, a pivotal step that facilitated efficient manipulation and analysis of the recipe data. Leveraging the read\_json function with the orient='records' parameter ensured seamless parsing of the JSON format into DataFrame rows, streamlining subsequent processing steps.

Data integrity is paramount in any machine learning endeavor, laying the groundwork for robust model performance during subsequent training and evaluation phases. Identifying and rectifying missing values were pivotal steps undertaken to ensure the robustness and completeness of our dataset. We addressed missing values by dropping rows where essential information, such as recipe titles, ingredients, instructions, or picture links, was absent. This approach was crucial in maintaining the quality and integrity of our dataset,

Effective representation of textual data is essential for model learning and generation. To enhance the quality of textual data within the dataset, we performed text cleaning operations. Specifically, we focused on the "ingredients" column and removed extraneous text, such as the word "ADVERTISEMENT." This cleaning step was pivotal in promoting consistency and clarity in the representation of recipe ingredients.

2 Tokenization and Data Preparation

Our quest for an optimized tokenization process and data preparation pipeline marked a crucial phase in the development of the RM (Recipe Model). We began by reimagining the tokenization process, recognizing the pivotal role of the recipe title in generating coherent outputs. One of the key adjustments involved reordering the arguments of the formatting\_func() tokenization function, prioritizing the title as the first argument rather than the last. This simple yet impactful alteration streamlined the tokenization process, enhancing code readability and efficiency.

Furthermore, our approach extended beyond individual datasets, encompassing all textual components—titles, ingredients, and instructions—simultaneously. By tokenizing multiple datasets in parallel, we optimized resource utilization and accelerated overall processing speed.

However, efficiency wasn't the sole focus; effective resource management was equally paramount. To strike a balance between resource utilization and model training effectiveness, we made the strategic decision to constrain the dataset size for processing. We narrowed our focus to the first 200 rows. This adjustment expedited the tokenization process while capturing sufficient variation in recipe content for robust model learning.

We conducted a thorough inspection of data lengths and distribution, gaining valuable insights into the characteristics of our dataset. Histograms plotting the lengths of input IDs for titles and ingredients provided critical information for setting appropriate maximum length thresholds and padding strategies during tokenization. Based on our analysis, we specified a maximum length of 365, effectively mitigating the risk of memory overflow and enhancing computational efficiency during subsequent model training stages.

3 Model Training and Fine-Tuning

Transitioning to model training and fine-tuning, we leveraged preprocessed data from the tokenization stage to facilitate learning and optimization. Several key adjustments and configurations were made to enhance the training process and ensure the effectiveness of model fine-tuning.

Firstly, we prepared the training data by combining tokenized titles, ingredients, and instructions. This amalgamation of textual components into a cohesive training dataset is essential for facilitating comprehensive learning and enabling the model to generate coherent recipe outputs.

To control the training process effectively, we specified the number of training steps (max\_steps) as 500. This parameter governs the duration of the training process, ensuring sufficient iterations for model convergence while mitigating the risk of overfitting or excessive resource consumption.

Additionally, we designated separate output directories for saving model checkpoints and logs, specifying ./output for model checkpoints and ./logs for logging purposes. This organizational strategy enhances the manageability of training artifacts and facilitates post-training analysis and evaluation.

By implementing these adjustments and configurations, we established a robust framework for model training and fine-tuning, optimizing efficiency and effectiveness in learning from the provided recipe datasets. The specified training steps, output directories, and data preparation strategies collectively contribute to the successful refinement of the Mistral language model for recipe generation.

4 Model Evaluation

The next crucial involved systematically evaluating the model's performance in generating recipe outputs. We defined evaluation prompts to guide the model and computed metrics such as BLEU, ROUGE-1, and METEOR to quantitatively assess the quality of generated text.

We define the evaluation prompt (eval\_prompt) to guide the model in generating recipe outputs for assessment. The evaluation prompt comprises the title as the first part, followed by the ingredients and instructions. This structured format ensures consistency in the evaluation process and enables comprehensive analysis of the model's outputs.

As for the metrics used for evaluation: BLEU calculates precision by comparing the n-grams of the generated text with those of the reference texts. ROUGE-1 focuses on the recall of n-grams between the generated text and the reference texts. Specifically, ROUGE-1 calculates the overlap of unigram (single word) sequences. With both BLEU and ROGUE-1, a higher score signifies a higher degree of overlap or similarity between the generated text and the reference texts.

METEOR evaluates the quality of generated text by considering both exact word matches and semantic similarity. It compares the generated text with reference texts using various linguistic resources, including synonyms and stems. A higher METEOR score indicates better agreement between the generated text and the reference texts.

We chose to implement these metrics to comprehensively evaluate the performance of our model in generating coherent recipe outputs. By considering both lexical similarity (BLEU, ROUGE-1) and semantic relevance (METEOR), we gain insights into different aspects of the model's performance, ensuring a more holistic evaluation.

To visualize the evaluation results effectively, we plot the computed metrics based on the generated text and reference text. The plotted metrics, including BLEU, ROUGE-1, and METEOR, offer a clear and concise representation of the model's performance, enabling stakeholders to assess its strengths and identify areas for improvement.

By systematically evaluating the model outputs using established metrics and methodologies, we gain valuable insights into its effectiveness and performance in generating recipe outputs. These insights inform further refinements and optimizations to enhance the model's capabilities and usability in real-world applications.

5 Quantization and Model Compression

To optimize the efficiency and performance of our language model, we explored the process of quantization and model compression. Implementing LORA (Lora Optimizer for Robustness and Accuracy), we significantly reduced the memory and computational requirements of the model while preserving its functionality and accuracy. Leveraging parallelization techniques further enhanced the efficiency of the quantized model, making it suitable for deployment in resource-constrained environments.

LORA is a powerful optimization technique designed to enhance the efficiency of deep learning models by reducing their memory footprint and computational requirements. The core idea behind LORA is to quantize the parameters of the model, thereby representing them with a lower precision format, such as 4-bit or 8-bit integers. This reduction in precision significantly reduces the memory and computational resources needed to store and process the model, making it more efficient for deployment in resource-constrained environments.

To implement LORA, we first prepare the model for k-bit training using the prepare\_model\_for\_kbit\_training function from the peft library. This function enables gradient checkpointing and prepares the model for quantization by optimizing its architecture and parameters.

Next, we define the configuration for LORA using the LoraConfig class, specifying parameters such as the compression ratio (r), alpha value (lora\_alpha), target modules for quantization, and dropout rate. These parameters are crucial for controlling the trade-off between model compression and accuracy, ensuring that the quantized model retains its performance while reducing its size.

Once the LORA configuration is defined, we apply it to the model using the get\_peft\_model function, which quantizes the specified target modules according to the configured parameters.

Additionally, we leverage parallelization techniques such as model parallelism and data parallelism to further enhance the efficiency of the quantized model, enabling it to leverage multiple GPUs for accelerated training and inference.

By implementing LORA and leveraging model compression techniques, we can significantly reduce the memory and computational requirements of our language model without compromising its performance. This optimization allows us to deploy the model in resource-constrained environments and improve its scalability and accessibility for a wide range of applications.

6 Model Inference

We addressed model inference by exploring the practical steps involved in utilizing the fine-tuned model for generating text. Initially, we loaded the trained model using the PeftModel class from the peft module, enabling access to the model's checkpoint containing its trained weights and configurations. Subsequently, we proceeded to the inference stage, where we provided a prompt to the loaded model. This prompt includes a title, list of ingredients, and cooking instructions concatenated together.

After tokenizing the prompt using the model's associated tokenizer, we fed the tokenized prompt into the fine-tuned model. Utilizing the generate method, we instructed the model to generate text based on the provided prompt. Finally, the generated text was decoded using the same tokenizer, resulting in human-readable output.

7 Evaluation and Visualization

In the chapter on Evaluation and Visualization, we embarked on assessing the performance of our text generation model and visualizing the evaluation results. Our approach involved computing evaluation metrics to gauge the quality of the generated text, followed by visualizing these metrics to gain insights into the model's strengths and weaknesses.

To begin with, we computed evaluation metrics by comparing the generated text against a reference text, which typically contained specific details about ingredients and the expected output title and instructions. These metrics, including BLEU, ROUGE-1, and METEOR, provided quantitative measures of various aspects of the generated text's quality, such as similarity to the reference text and linguistic fluency.

Subsequently, we proceeded to visualize the evaluation results by plotting the scores of these metrics on a bar chart. This visualization provided a clear and concise representation of how the model performed across different evaluation criteria. By analyzing these visualizations, we gained valuable insights into the model's performance, identifying its strengths and areas for improvement.

Through iterative evaluation and visualization of the model's performance, we aimed to continuously improve the text generation process. These insights guided further refinements and optimizations, ultimately enhancing the overall quality and coherence of the generated text.

**3. Experimental Evaluation**

3.1 Methodology - how we tracked the sucess of our model

3.2 Results

3.3 Discussion

**4. Future Work**

**5. Conclusion**