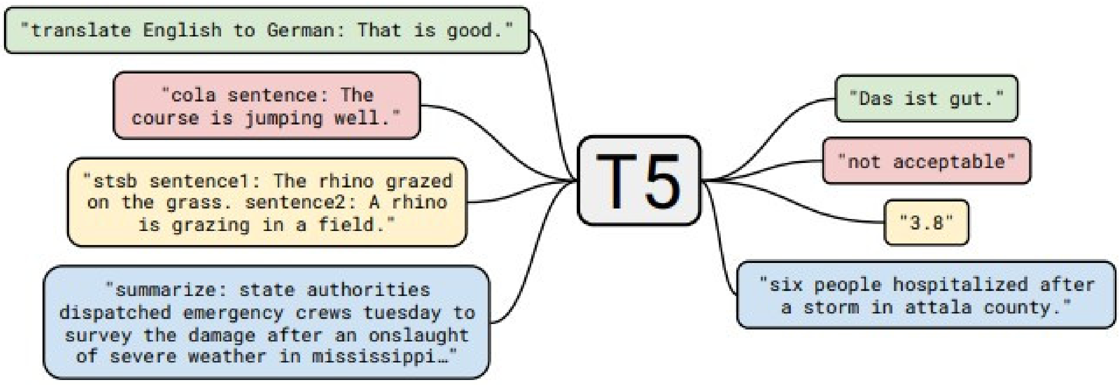
Large Language Models (Homework 2)

Due date : 2024/12/1 23:55:00 (Hard Deadline)

1 Text Summarization (56%)

You will receive a dataset for a text summarization task, sampled from "CNN-DailyMail News Text Summarization" dataset. It is an English-language dataset containing over 300K unique news articles written by journalists from CNN and the Daily Mail. The current version supports the evaluation for both extractive and abstractive summarization. You need to enable and use the Google-T5-small model to transform input articles into their summaries without requiring any prompts or instructions.

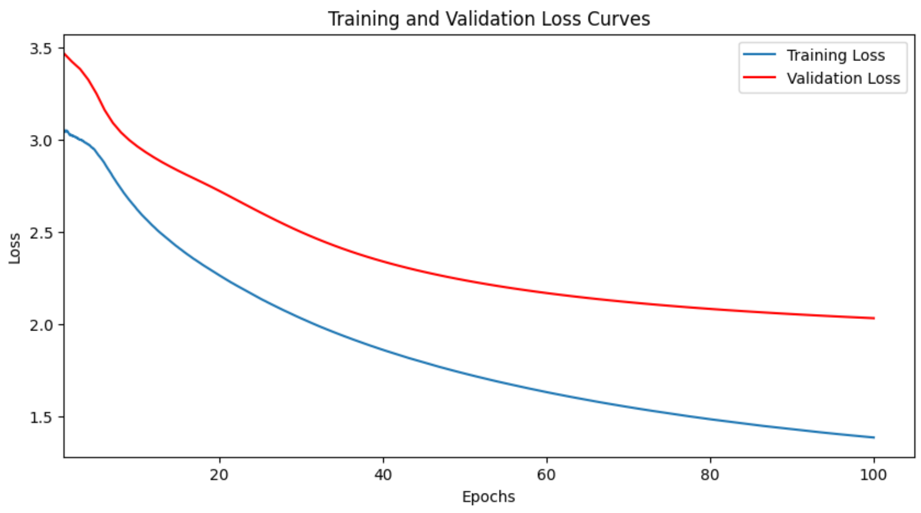

"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Dataset description:

- This dataset includes three files: train.csv, validation.csv, and test.csv.
- train.csv contains approximately 2,900 entries, validation.csv around 130 entries, and test.csv around 110 entries.
- Each file contains three columns: "id", "article" and "highlight".
- You will use the data samples in train.csv to fine-tune the model, validate the training process with validation.csv, and finally evaluate the model performance using test.csv.

1. Please complete the summarization task according to the following instructions.
   - Please use the Google-T5-small model for fine-tuning. This model is designed for "sequence-to-sequence" tasks. Detailed information can be found in Hugging Face by the following link: "Google-T5-small".
   - Be sure to use the specific model version indicated in the files.
   - Please use "article" as the model input and "highlight" as the target to train the model. Note that in this part please do not add any prompts or prefixes to the input.
   - You may need to use Kaggle's GPU for programming.
   - There are no restrictions on any of the training parameters. You will need to evaluate them by yourself.

(a) Plot the learning curves for training and validation losses of training data during training. (15%)



Note: Figure above shows an example. The result might be different.

(b) Please implement the functions for ROUGE-1 and ROUGE-2 to evaluate the model performance on the test.csv dataset. Directly calculating the functions, such as ``load-metric'', is not allowed. You need to write these functions from scratch without using any existing functions. (5%)

$$\text{ROUGE-}N = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{Precision} = \frac{\text{Count of overlapping } N\text{-grams}}{\text{Total } N\text{-grams in generated summary}}$$

$$\text{Recall} = \frac{\text{Count of overlapping } N\text{-grams}}{\text{Total } N\text{-grams in reference summary}}$$

$$N = 1 \text{ or } 2$$

(c) Similarly, write a function to calculate ROUGE-L to evaluate the model results. Compare the results of ROUGE-L with those of the ROUGE-1 and ROUGE-2, and make some discussion. (5%)

$$\text{ROUGE-}L = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{Precision} = \frac{\text{LCS (Generated Summary, Reference Summary)}}{\text{Length of Generated Summary}}$$

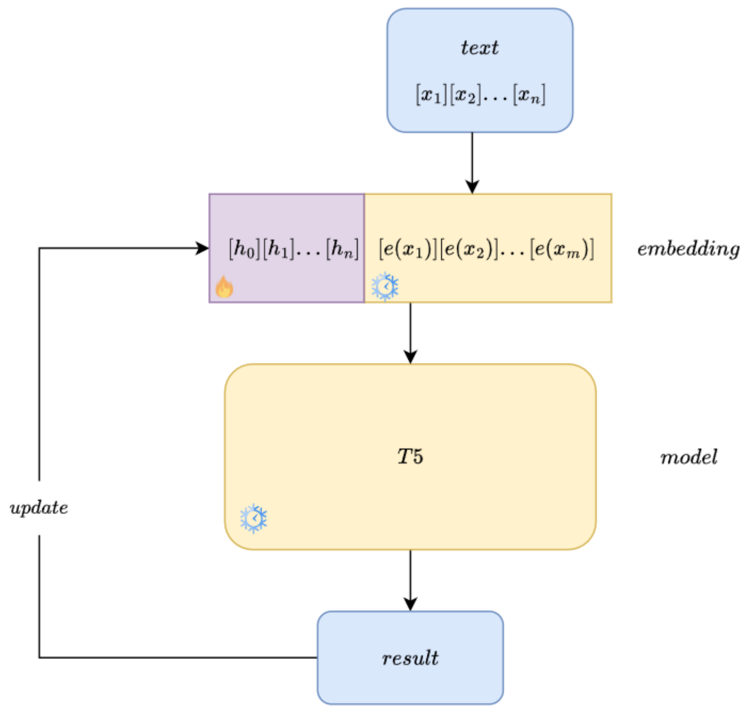
$$\text{Recall} = \frac{\text{LCS (Generated Summary, Reference Summary)}}{\text{Length of Reference Summary}}$$

Example of LCS (Longest Common Subsequence):

- Summary: Today is a good day.
  References: Today is wonderful day.
  LCS: Today is day.
- Summary: Wonderful day is today.
  References: Today is wonderful day.
  NO LCS.

2. Please use the pre-trained Google-T5-small model without any fine-tuning. Use the hard prompts template `‘‘summarize:{input text}’’` to have the model perform the summarization task on the test.csv dataset. Then, evaluate the results using your own implementations of ROUGE-1, ROUGE-2, and ROUGE-L. (3%)

3. Please use the model you fine-tuned in the first question, and apply the soft prompt method to enhance the performance of summary tasks. You need to use an prompt embedding and embed it in front of each input text embedding, allowing the model to generate an output. Then, update the soft prompt embedding based on the generated output. Note that you do not need to fine-tune the entire model again. Only the prompt embedding should be fine-tuned.



   (a) Plot the learning curves for training and validation losses of training data during training. (15%)

   (b) Evaluate the results using your own implementations of ROUGE-1, ROUGE-2, and ROUGE-L. (3%)

4. Compare the processes and the results of all the previously used methods, including fine-tuning, hard prompt, and soft prompt, and make some discussion. (10%)

NOTE: Your ROUGE score on each question needs to exceed the baseline.

```
Average ROUGE-1: 0.36151793926514275
Average ROUGE-2: 0.14854682775225891
Average ROUGE-L: 0.24718851243923656
```

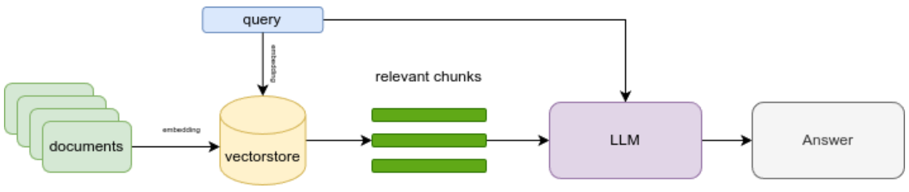
2 Retrieval-Augmented Generation (44%)

In this assignment, you will use LangChain to implement a retrieval-augmented generation (RAG) system and explore some of its variations. In addition to the correctness of the code logic, the main focus will be on the quality of the descriptions in your discussion. RAG is a technique that combines the information retrieval method with the text generation, enabling language models to generate the responses based on the relevant documents. This approach is especially useful for answering queries that require up-to-date or domain-specific knowledge.

You will be given an IPython notebook (.ipynb file) along with a PDF document to use as the source material. Please fill in the blanks to complete the required discussion according to the criteria provided. Make sure to include the output cells with the experimental results and discussions when you save the file.

You will be using LlaMA3.2 3B in this part. For those meeting the hardware requirements (detailed in the notebook), follow the "Presteps to Load Llama3.2 Locally" section. Otherwise, follow the "Presteps to Load Llama3.2 on Colab" section to run on Colab.

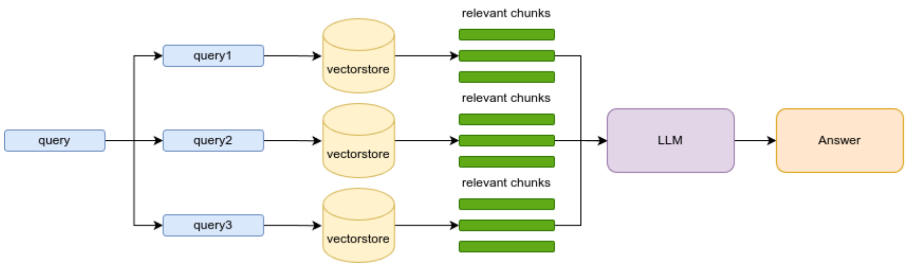
1. Standard RAG



   Reference: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

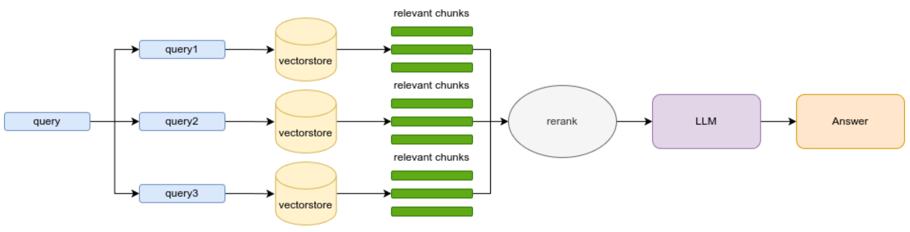
   (a) **Chain the Components:** Chain together the retriever, prompt, LLM, and output parser to build a standard RAG pipeline. (4%)
   (b) **Explain TextSplitter Settings:** Discuss how `chunk_size`, `chunk_overlap`, and other parameters in the `TextSplitter` affect the retrieval process. (4%)
   (c) **Experiment with Retriever Settings:** Try various settings for the retriever, such as k, the number of top documents returned, `search_type`, and the other configuration options. Show the retrieved documents for each configuration and discuss the scenarios where different settings would be suitable.(5%)

2. Multi-Query RAG



   (a) **Prompt Template for Multi-Query:** Design a prompt template that takes a single input query and generates multiple related queries. This prompt should guide the model to produce various angles or interpretations of the original query, improving retrieval coverage. (4%)
   (b) **Multi-Query RAG Chain:** Implement a chain where the input is a single query, but the output is the final answer generated after running each of the multiple queries through the RAG pipeline. (4%)
   (c) **Example Comparisons:** Show a standard RAG example alongside a multi-query RAG example. Compare and discuss the outputs to highlight how the multi-query approach yields a more comprehensive or nuanced answer, especially for complex or ambiguous queries. (6%)

3. RAG Fusion



Reference: RAG-FUSION: A NEW TAKE ON RETRIEVAL-AUGMENTED GENERATION.

(a) **Implement Reciprocal Rank Fusion (RRF):** Implement the `reciprocal_rank_fusion` function to aggregate the results from different queries. This function combines rankings from multiple retrievals, prioritizing documents that appear across several results. (9%)

RRF simply sorts the documents according to a naive scoring formula. Given a set of documents D and a set of rankings R, where each ranking r is a permutation of $1, \ldots, |D|$, we compute the RRF score for each document $d \in D$ as follows:

$$\text{RRFscore}(d \in D) = \sum_{r \in R} \frac{1}{c + r(d)}$$

where $r(d)$ is the position of document d in ranking r, and c is a constant.

For a more detailed explanation, refer to this reference:
Reciprocal Rank Fusion (RRF) explained in 4 mins — How to score results form multiple retrieval methods in RAG

(b) **RRF Example and c-Value Discussion:** Show an example showing the documents after re-ranking using RRF. Discuss how the c-value in RRF affects the results, and describe the scenarios where RRF is beneficial. (4%)

(c) **RAG Fusion Chain:** Implement the RAG Fusion chain in the same way as the chain in Part 2. (4%)

3 Rule

- In your submission, you need to submit two files. And only the following file format is accepted:
  - **hw2_<ProblemNumber>_<StudentID>.ipynb** file which need to contain all the results, codes and reports for each exercise (e.g. **hw2_2_0123456.ipynb**).
- Implementation will be graded by
  - Completeness
  - Algorithm correctness
  - Description of model design
  - Discussion and analysis
- Only Python implementation is acceptable.

- You may need to use the GPU for each question.
- DO NOT PLAGIARIZE. (We will check program similarity score.)