Text Summarization Report

Ramtin Asgarianamiri

Introduction

Text summarization is a method of compressing a given document so as to summarize it, while retaining the general idea of the original text. Summarization has become a necessary tool in information processing since there are enormous amount of textual data for any kind of article. Hence importance of extracting valuable information. The enhancement of productivity can be resulted by reducing time spent on long documents. For instance summarization enables better decision making in in healthcare.

Text summarization can be divided into two methods: abstractive and extractive. Abstractive summarization focuses on understanding the semantics of the original text and refining its ideas and concepts to achieve semantic reconstruction.

Summaries in their nature are abstractive. The summaries generated by extractive summarization methods are usually long and redundant, which bring bad reading experience We focus on abstractive summarization. Abstractive summarization may add new words or phrases when generating summaries, which is usually viewed as a sequence-to-sequence learning problem.

In my project I have implement two state of art transformer-based models, BART and T5 to do the summarization process. Both of the models are pretrained on large datasets, make them appropriate for utilizing in summarizing in terms of text summarization.

In the next step I have chosen BART to fine tune on the method of Reinforcement Learning. RL has been employed to optimize abstractive summarization models beyond standard supervised fine-tuning. By treating text generation as a sequential decision-making problem, RL fine-tunes the model parameters to directly maximize desired evaluation metrics, such as ROUGE.

Method

Rush [1] et al first used the Seq2Seq model to generate summaries, which generates summaries based on understanding the semantics of the text, which is more similar to the process of generating manual summaries. Bahdanau et al [2] introduced an attention mechanism in the model by calculating weights for each hidden state of the encoder based on the hidden states of the decoder, with the aim of making the decoder focus on certain key words in the document when generating each word.

Extractive summarization involves extracting meaningful information from the original text. Lewis et al. [3] introduced the BART (Bidirectional and Auto-Regressive Transformers) model: “Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation and Comprehension”. BART is a transformer encoder-encoder (seq2seq) model with a bidirectional (BERT(Bidirectional Encoder Representations from Transformers)-like) encoder and an autoregressive (GPT(Generative Pre-trained

Transformer)-like) decoder.[3]

BERT models are known for their ability to understand the context of sentences and words deeply, keeping relevant information to produce informative summaries.[4] Furthermore, the BART (Bidirectional and Autoregressive Transformers) method is a model capable of producing extractive summarization. BART has the advantage of producing more creative summaries than BERT because BART's advantage lies in its ability to produce more general summaries but still reflect the essence of the information.

On the other hand, Raffel et al. [5] introduced the T5-base model in "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". T5-base model is an encoder decoder model that has been pre-trained on a multi-task combination of unsupervised and supervised workloads, the T5 (Text-To-Text Transfer Transformer) method is a model that is also capable of abstractive summarization. T5 is known for its versatile approach, where text is also considered input and output in text form. This provides flexibility in designing various natural language processing tasks, including summarization [6].

BART and T5 models follow a sequence-to-sequence architecture with an encoder-decoder structure. The encoder gets in the document to create a dense representation which allows the model to have semantic and syntactic awareness on the input. The decoder generates the summary based on the representation. It predicts one token at a time with focusing on the previous generated tokens.

Why These Models?

BART has been selected due to its great results and exhibition while performing the abstractive summarization task with its strong pre-training methods. Its architecture follows summarization requirements, hence generating high values of output. T5 serves a mixed-use scope, a little less specialized, serving as a useful point of comparison but since it has been used in the initial paper, I have decided to use both models. This will allow the evaluation of both models in their capability summarize individually.

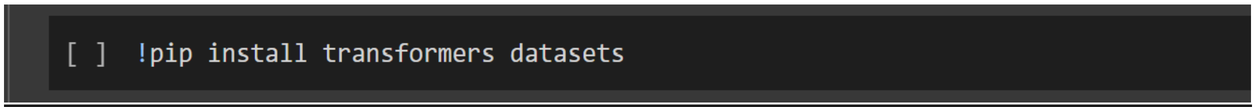
Fine-tuning:

For the next step, the purpose was to reward the model by implementing reinforcement model on fine tuning of BART. Many capable large language models (LLMs) are developed via self-supervised pre-training followed by a reinforcement-learning fine-tuning phase, often based on human or AI feedback. During this stage, models may be guided by their inductive biases to rely on simpler features which may be easier to extract, at a cost to robustness and generalisation.[7] Reinforcement Learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment to achieve a goal. The method improves its behavior over time by receiving rewards or penalties based on the outcomes of its actions. This approach shifts the training process from predicting the next token to directly optimizing the model to produce high-quality summaries as measured by an evaluation metric. After generating a summary, it receives a reward based on how well the generated summary aligns with the reference summary, calculated using the ROUGE metric. The model updates its parameters to increase the likelihood of generating sequences that yield higher rewards.

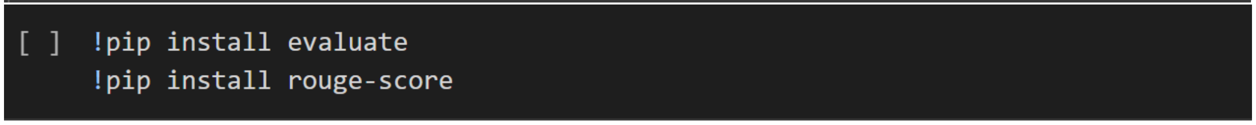
Implementation

BART and T5

In this section, the code will be explained in detail. Each part of the code has been screenshotted and the following if each figure is the description of that section of coding.

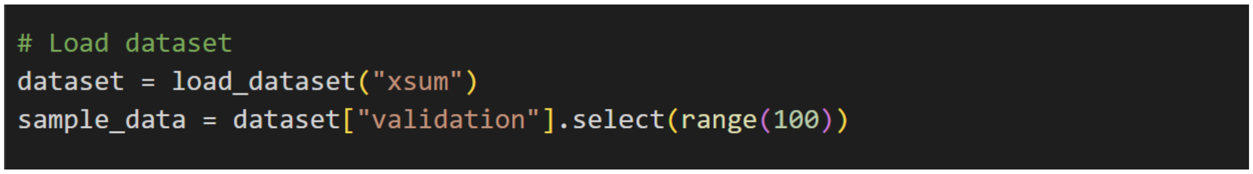


```
[ ]  !pip install transformers datasets
```



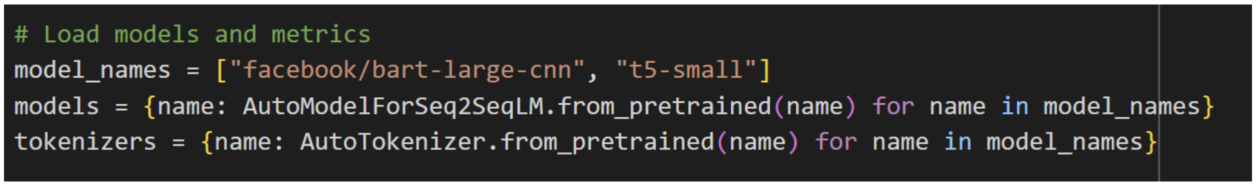
```
[ ]  !pip install evaluate
     !pip install rouge-score
```

Transformers has been installed to access pre-trained models and Dataset provide access to XSum dataset. Evaluate for evaluation metrics and rouge-score for ROUGE metric calculations.



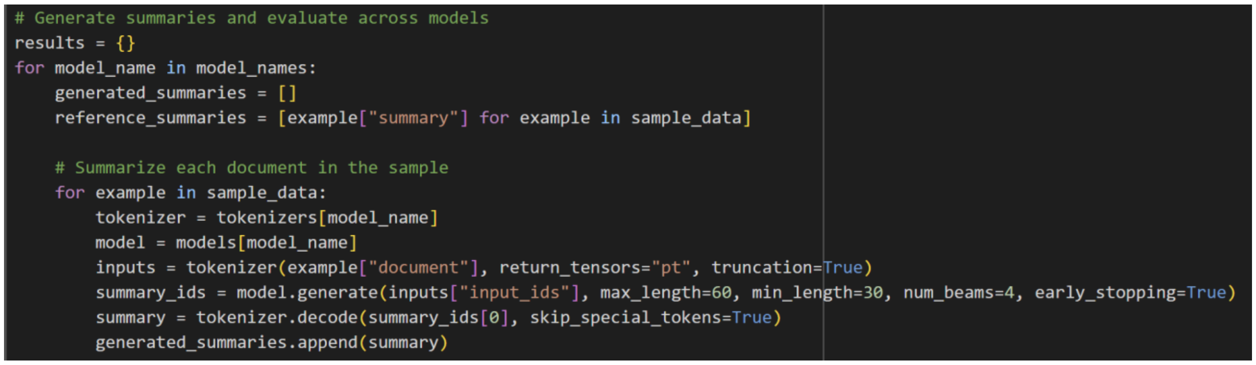
```
# Load dataset
dataset = load_dataset("xsum")
sample_data = dataset["validation"].select(range(100))
```

In this stage we download our dataset which is XSum from the library and for sample data we choose 100 of the documents and their gold summary to evaluate and present some results from them.



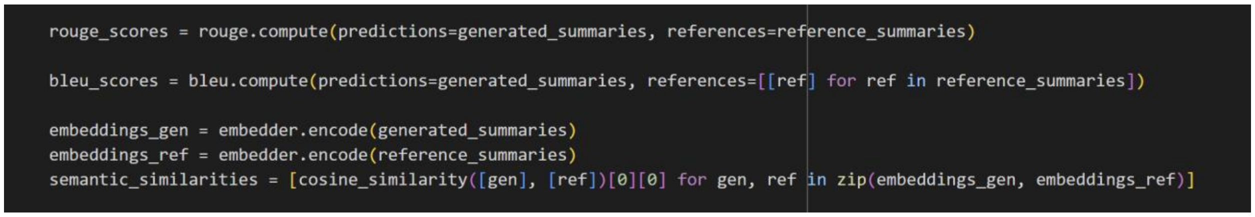
```
# Load models and metrics
model_names = ["facebook/bart-large-cnn", "t5-small"]
models = {name: AutoModelForSeq2SeqLM.from_pretrained(name) for name in model_names}
tokenizers = {name: AutoTokenizer.from_pretrained(name) for name in model_names}
```

“facebook/bart-large-cnn”, a pre-trained BART model for summarization and “T5-small” have been chosen. We define tokenizer and the model from BART and T5 separately and employ them for our model.



```
# Generate summaries and evaluate across models
results = {}
for model_name in model_names:
    generated_summaries = []
    reference_summaries = [example["summary"] for example in sample_data]

    # Summarize each document in the sample
    for example in sample_data:
        tokenizer = tokenizers[model_name]
        model = models[model_name]
        inputs = tokenizer(example["document"], return_tensors="pt", truncation=True)
        summary_ids = model.generate(inputs["input_ids"], max_length=60, min_length=30, num_beams=4, early_stopping=True)
        summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
        generated_summaries.append(summary)
```

We define a dictionary for storing the results and generated summaries, and we store references.

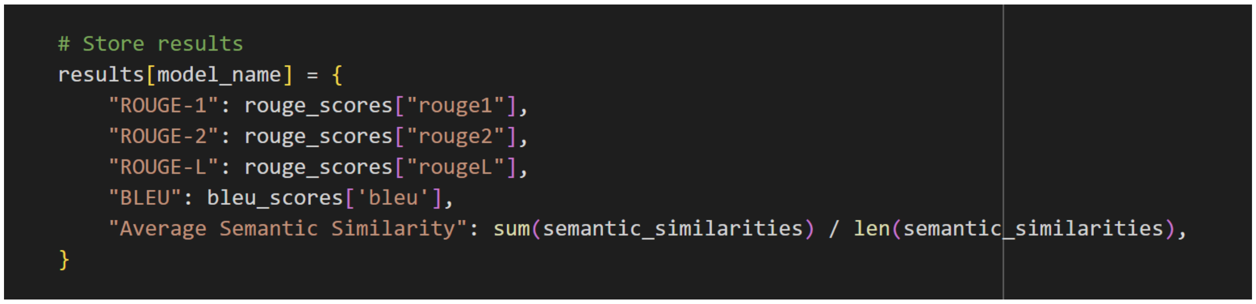


```
rouge_scores = rouge.compute(predictions=generated_summaries, references=reference_summaries)

bleu_scores = bleu.compute(predictions=generated_summaries, references=[[ref] for ref in reference_summaries])

embeddings_gen = embedder.encode(generated_summaries)
embeddings_ref = embedder.encode(reference_summaries)
semantic_similarities = [cosine_similarity([gen], [ref])[0][0] for gen, ref in zip(embeddings_gen, embeddings_ref)]
```

In the for loop each example from the samples gathered from XSum, will be given to the chosen model (either BART or T5) First it will be tokenized by the encoder and be defined as inputs. Input will be given to the model to generate a representation for the summary of the sample. The tokenized summary will be given to a decoder so it will be textual. The generated text will be stored in generated_summaries.

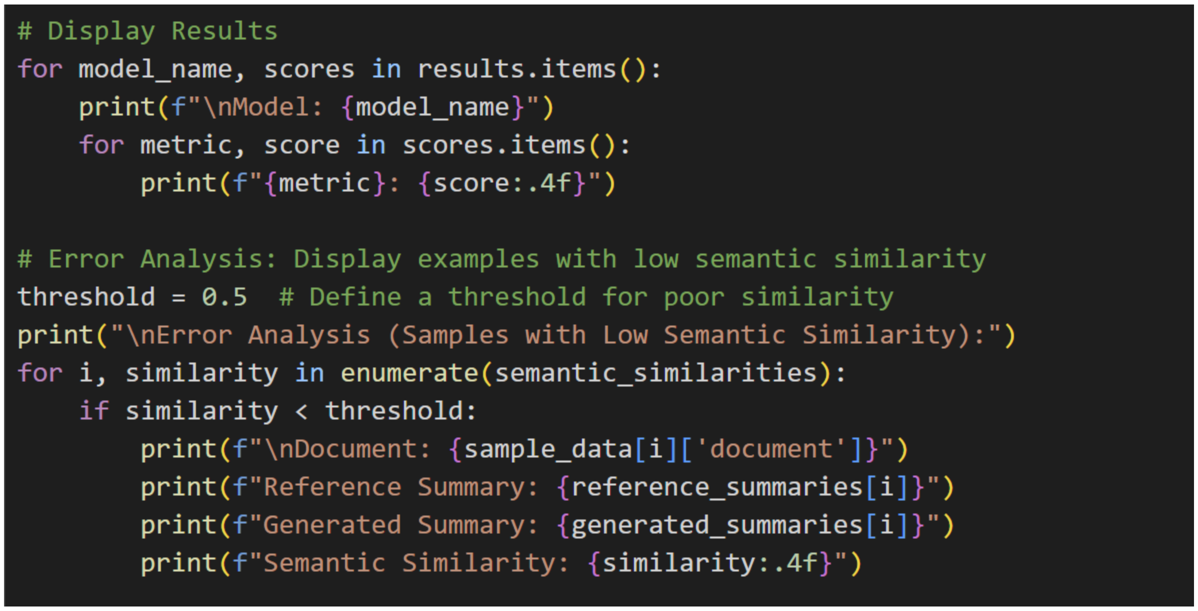
After generation of summary, rouge. compute will calculate the score for ROUGE (e.g., ROUGE-1, ROUGE-2, ROUGE-L) by comparing generated summaries with reference summaries.

For calculating the similarity for each summary pair firstly the embedder. encode with prepare the embeddings of both generated and referenced summary. Then the cosine similarity will be calculated.



```
# Store results
results[model_name] = {
    "ROUGE-1": rouge_scores["rouge1"],
    "ROUGE-2": rouge_scores["rouge2"],
    "ROUGE-L": rouge_scores["rougeL"],
    "BLEU": bleu_scores['bleu'],
    "Average Semantic Similarity": sum(semantic_similarities) / len(semantic_similarities),
}
```

results[model_name]: Stores evaluation metrics for the current model, including ROUGE, BLEU, and average semantic similarity.

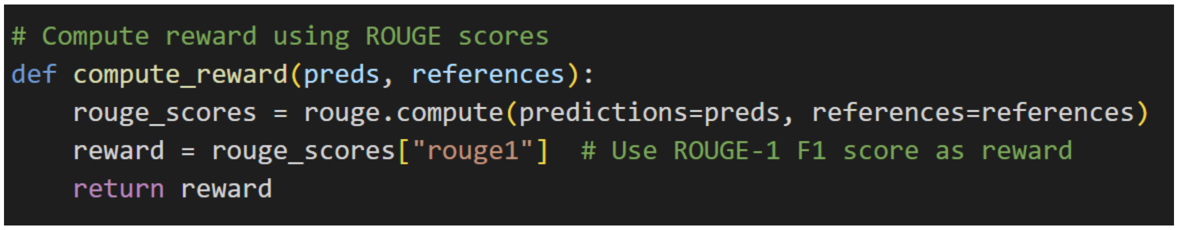


```
# Display Results
for model_name, scores in results.items():
    print(f"\nModel: {model_name}")
    for metric, score in scores.items():
        print(f"{metric}: {score:.4f}")

# Error Analysis: Display examples with low semantic similarity
threshold = 0.5  # Define a threshold for poor similarity
print("\nError Analysis (Samples with Low Semantic Similarity):")
for i, similarity in enumerate(semantic_similarities):
    if similarity < threshold:
        print(f"\nDocument: {sample_data[i]['document']}")
        print(f"Reference Summary: {reference_summaries[i]}")
        print(f"Generated Summary: {generated_summaries[i]}")
        print(f"Semantic Similarity: {similarity:.4f}")
```

In this part, the evaluation metrics will be printed for each model. For understanding better, the weak points of the model, the second part of the code has been set. It basically will look for summaries that have similarities calculated less than the threshold which is 0.5. In this case, we will be given examples of the poor summarization to give us insights.

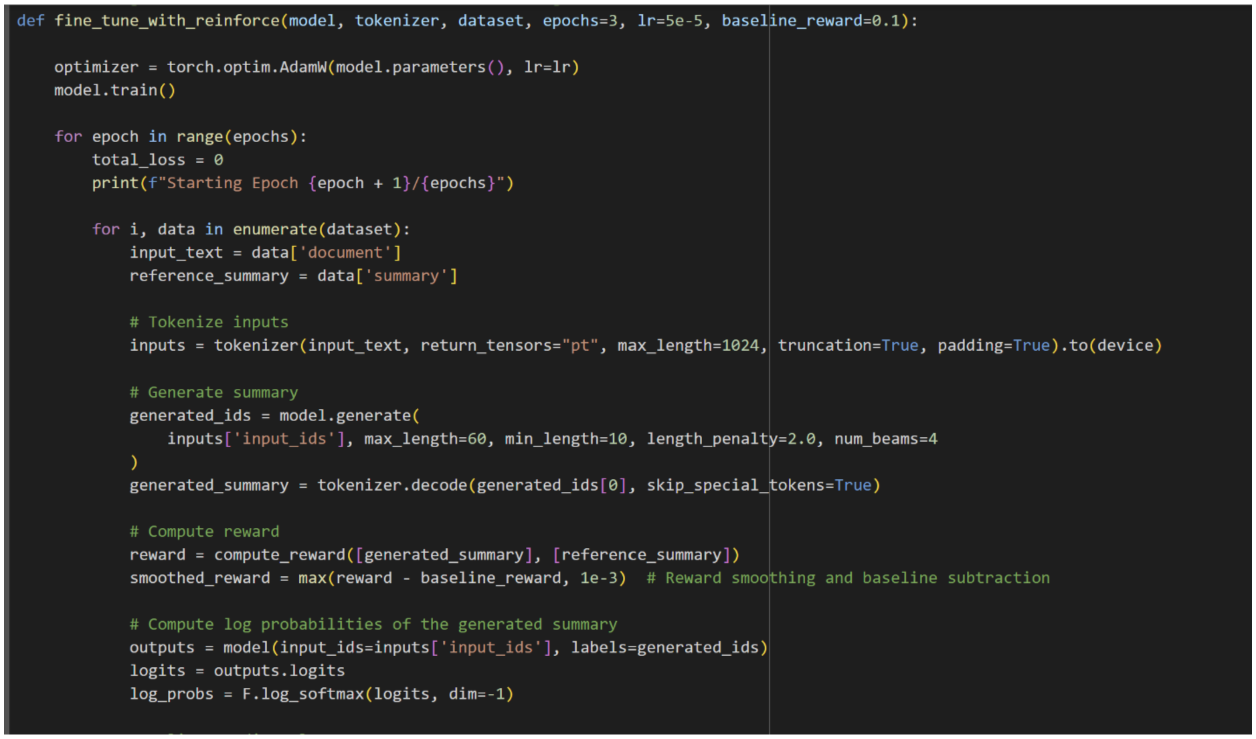
Fine-Tuning

In the following the description and explanation of the code for fine tuning BART model will be purposed.



```python
# Compute reward using ROUGE scores
def compute_reward(preds, references):
    rouge_scores = rouge.compute(predictions=preds, references=references)
    reward = rouge_scores["rouge1"]  # Use ROUGE-1 F1 score as reward
    return reward
```

In this part the reward function is defined. In the model we have picked the score of ROUGE-1 as our reward metric.



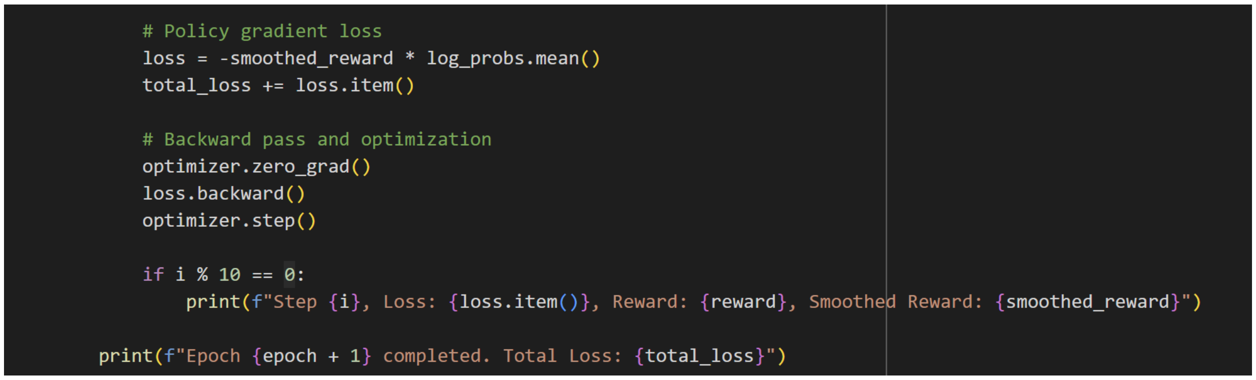
```python
def fine_tune_with_reinforce(model, tokenizer, dataset, epochs=3, lr=5e-5, baseline_reward=0.1):

    optimizer = torch.optim.AdamW(model.parameters(), lr=lr)
    model.train()

    for epoch in range(epochs):
        total_loss = 0
        print(f"Starting Epoch {epoch + 1}/{epochs}")

        for i, data in enumerate(dataset):
            input_text = data['document']
            reference_summary = data['summary']

            # Tokenize inputs
            inputs = tokenizer(input_text, return_tensors="pt", max_length=1024, truncation=True, padding=True).to(device)

            # Generate summary
            generated_ids = model.generate(
                inputs['input_ids'], max_length=60, min_length=10, length_penalty=2.0, num_beams=4
            )
            generated_summary = tokenizer.decode(generated_ids[0], skip_special_tokens=True)

            # Compute reward
            reward = compute_reward([generated_summary], [reference_summary])
            smoothed_reward = max(reward - baseline_reward, 1e-3)  # Reward smoothing and baseline subtraction

            # Compute log probabilities of the generated summary
            outputs = model(input_ids=inputs['input_ids'], labels=generated_ids)
            logits = outputs.logits
            log_probs = F.log_softmax(logits, dim=-1)
```

In the fine-tuning process, we first of all define the hyper parameters like epoch size and etc. and for each epoch the input text will be document and reference summary is the gold summary in the data set.

With encoder then the input and referenced summary will be tokenized into max length of 1024.Truncate input to fit the model’s constraints. Pt means they are PyTorch tensors.

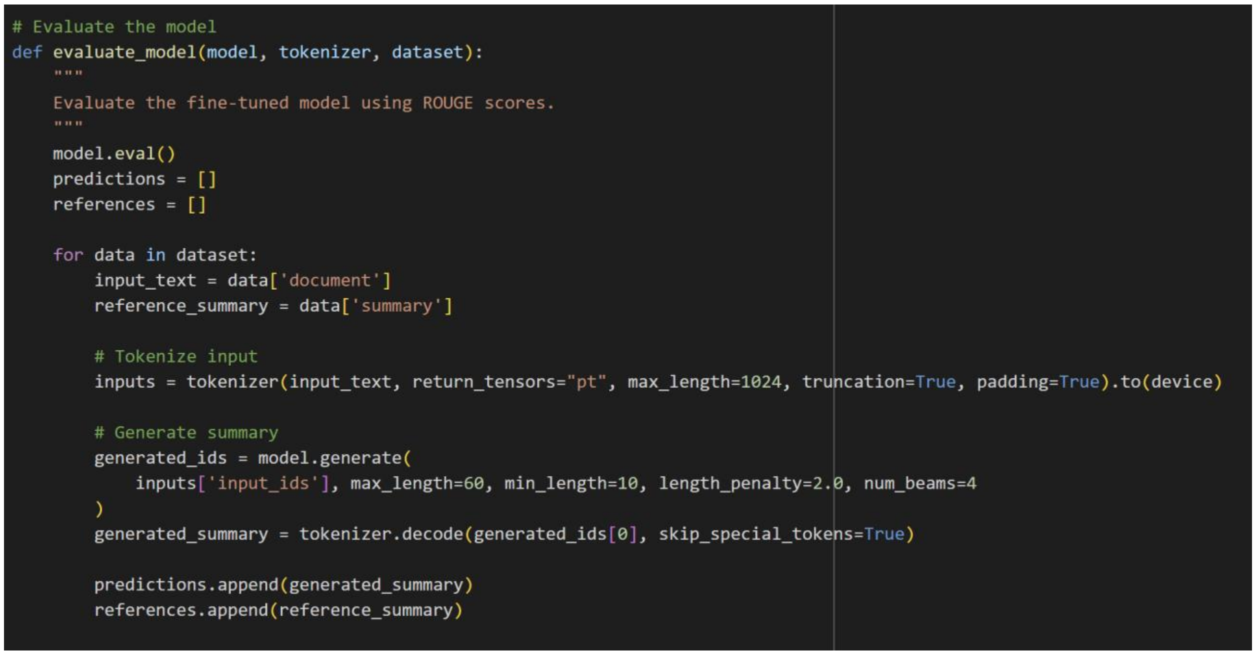
Generate function decodes input text into summary. And the amount for beam search has been set on 4. By generating the summary, reward function is used to calculate the f1 score of ROUGE-1 to be the reward.

The last part purpose is to calculate the reinforce loss. Log-probabilities of the generated tokens are calculated and saved in log_probs. Loss variant is saved to guide the model to maximize ROUGE rewards.



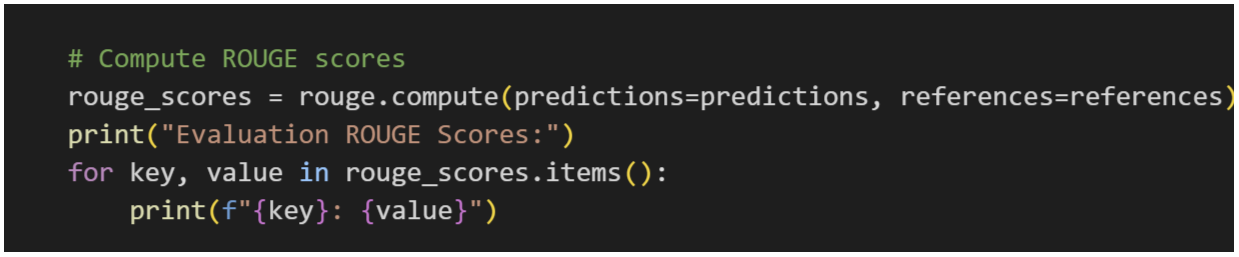
```
            # Policy gradient loss
            loss = -smoothed_reward * log_probs.mean()
            total_loss += loss.item()

            # Backward pass and optimization
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if i % 10 == 0:
                print(f"Step {i}, Loss: {loss.item()}, Reward: {reward}, Smoothed Reward: {smoothed_reward}")

        print(f"Epoch {epoch + 1} completed. Total Loss: {total_loss}")
```

The Previous gradients will be clear, the new gradients for the loos is compted and step() updates parameters with the optimizer.



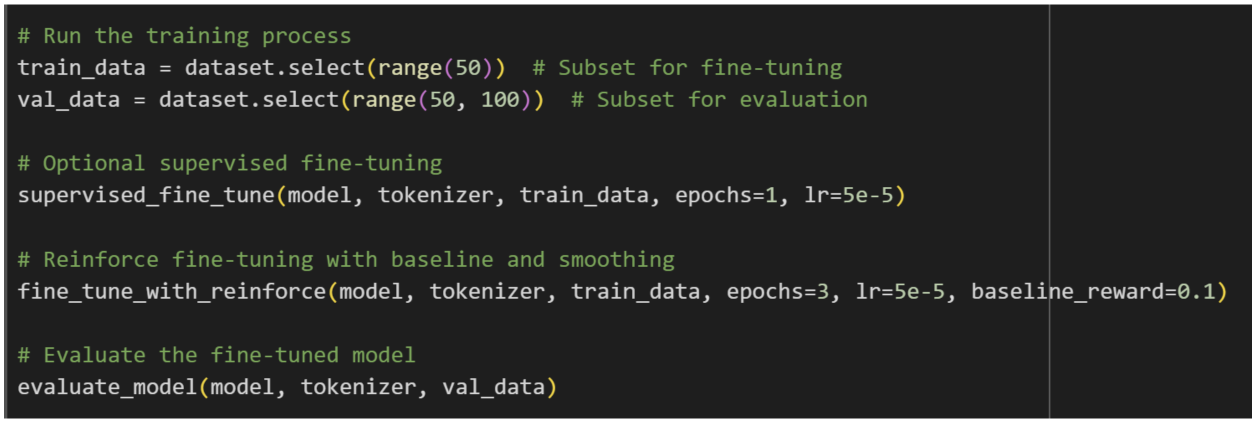
```
# Evaluate the model
def evaluate_model(model, tokenizer, dataset):
    """
    Evaluate the fine-tuned model using ROUGE scores.
    """
    model.eval()
    predictions = []
    references = []

    for data in dataset:
        input_text = data['document']
        reference_summary = data['summary']

        # Tokenize input
        inputs = tokenizer(input_text, return_tensors="pt", max_length=1024, truncation=True, padding=True).to(device)

        # Generate summary
        generated_ids = model.generate(
            inputs['input_ids'], max_length=60, min_length=10, length_penalty=2.0, num_beams=4
        )
        generated_summary = tokenizer.decode(generated_ids[0], skip_special_tokens=True)

        predictions.append(generated_summary)
        references.append(reference_summary)
```

This part evaluates the fined tuned model on unseen data, it will get a document and gold summary and generated summary will be concluded from the model. They will be added to the predictions which carry generated summaries and references which carry gold summaries.



```
    # Compute ROUGE scores
    rouge_scores = rouge.compute(predictions=predictions, references=references)
    print("Evaluation ROUGE Scores:")
    for key, value in rouge_scores.items():
        print(f"{key}: {value}")
```

Calculate and display ROUGE scores for the evaluation dataset.



```
# Run the training process
train_data = dataset.select(range(50))  # Subset for fine-tuning
val_data = dataset.select(range(50, 100))  # Subset for evaluation

# Optional supervised fine-tuning
supervised_fine_tune(model, tokenizer, train_data, epochs=1, lr=5e-5)

# Reinforce fine-tuning with baseline and smoothing
fine_tune_with_reinforce(model, tokenizer, train_data, epochs=3, lr=5e-5, baseline_reward=0.1)

# Evaluate the fine-tuned model
evaluate_model(model, tokenizer, val_data)
```

Due to the GPU limitation, 50 documents will be chosen for training and 50 documents for validation.

Evaluation

We present our main results on the XSum dataset. XSum The articles in the XSum dataset (Narayan, Cohen, and Lapata 2018) are from the BBC website with accompanying single sentence summaries, which are professionally written. The official splits of 204,045 articles for training, 11,332 articles for validation and 11,334 articles for test. All datasets are tokenized with the byte-pair encoding of GPT2 (Radford et al. 2019). Both models have been tested on the result of the evaluation has been demonstrated in table 1. The evaluation metrics that has been chosen for the models is inspired by the initial paper that has been presented at first. The evaluation metric is ROUGE. This metric measures the overlap of n-grams, such as unigrams or bigrams, between the generated summary and the reference summary. A higher score indicates better word or phrase-level overlap. ROUGE-1 is specifically calculating the overlap of each word to calculate the score. ROUGE-2 is the same methodology for evaluation however it focuses on the pair words. ROUGE-L (Longest Common Subsequence): This evaluates the longest common subsequence between the generated summary and the reference summary. Higher scores show better alignment of sentence-level structure.

Model	BART	T5
ROUGE-1	0.1726	0.1853
ROUGE-2	0.0302	0.0221
ROUGE-L	0.1537	0.1315
Average Similarity	0.5213	0.4582

Table 1- BART and T5 evaluation.

In the following also some examples of the generated summaries and the reference ones have been provided to be a good source of comparison.

Example 1:

Reference Summary: Google has hired the creator of one of the web's most notorious forums - 4chan.

Generated Summary: - known as "moot" online - created the site in 2003. it has gone on to be closely associated with offensive and often illegal activity. it was widely credited as being the first place where leaked images of nude celebrities were posted. **Semantic Similarity: 0.4564**

Example 2:

Reference Summary: A pedestrian has been struck by a taxi in Dundee after it mounted the pavement.

Generated Summary: - a young man - is thought to have been walking with a group of people from a graduation ceremony at the Caird Hall. the injured pedestrian - a young man - is thought to have been walking with a group of people**. Semantic Similarity: 0.2336**

Example 3:

Reference Summary: Labour leadership hopefuls Liz Kendall and Yvette Cooper have said their supporters should back anyone other than Jeremy Corbyn in the contest.

Generated Summary: Ms Kendall said the BBC Labour risked sending a "resignation letter" by electing Mr Corbyn. Ms Cooper warned there was a "serious risk the party will split" if the left-winger becomes its **leader. Semantic Similarity: 0.4675**

Fine-tuning

Moving on to the fine-tuning, Fine-tuning is the process by which a pre-trained model gradually adapts to a specific task with further training on task-specific labeled data. It usually the general knowledge learned during pre-training and refines its weights to perform well on the new task.

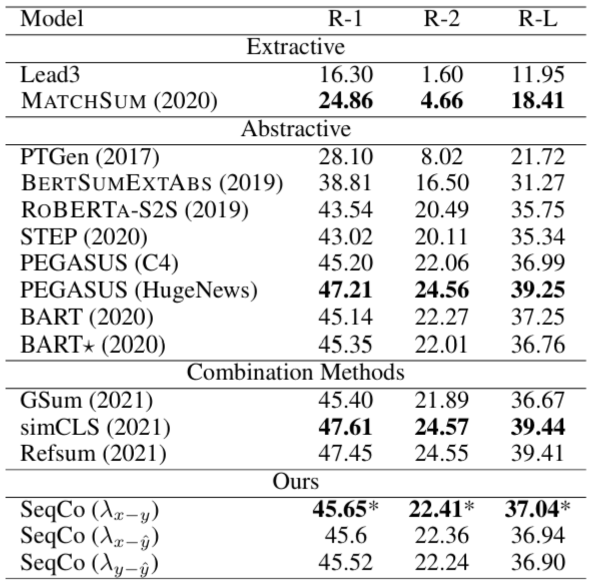
We have conducted a fine-tuning on the BART with XSum dataset provide better understating of needed summarization for the model. However, due to some obstacles that might occur in fine-tuning process. The goal wasn't achieved.

In the table 2, we can see the results of the evaluation of fine-tuned model.

Metrics	Results
ROUGE-1	0.1947
ROUGE-2	0.0032
ROUGE-L	0.1319

Table 2- Fine-tuned model evaluation.

To have a recall of the results on Sequence Level Contrastive Learning for Text Summarization Table-3 summarizes their results on the XSum dataset.



Model	R-1	R-2	R-L
Extractive			
Lead3	16.30	1.60	11.95
MATCHSUM (2020)	**24.86**	**4.66**	**18.41**
Abstractive			
PTGen (2017)	28.10	8.02	21.72
BERTSUMEXTABS (2019)	38.81	16.50	31.27
ROBERTA-S2S (2019)	43.54	20.49	35.75
STEP (2020)	43.02	20.11	35.34
PEGASUS (C4)	45.20	22.06	36.99
PEGASUS (HugeNews)	**47.21**	**24.56**	**39.25**
BART (2020)	45.14	22.27	37.25
BART$\star$ (2020)	45.35	22.01	36.76
Combination Methods			
GSum (2021)	45.40	21.89	36.67
simCLS (2021)	**47.61**	**24.57**	**39.44**
Refsum (2021)	47.45	24.55	39.41
Ours			
SeqCo (λ_{x-y})	**45.65***	**22.41***	**37.04***
SeqCo ($\lambda_{x-\hat{y}}$)	45.6	22.36	36.94
SeqCo ($\lambda_{y-\hat{y}}$)	45.52	22.24	36.90

Table2 :Results on the test split of XSum using full length ROUGE.

My method is showing very lower ROUGE results since it hasn't been fine tuned on the dataset. Additionally, in comparison to the paper above that has used 12 layer of BART to have deeper and complicated presentation, in my model it has only used 1 layer of Bart and T5 which resulted in much more shallow representation output for encoders that couldn't coney meaningfulness to the summary generator.

Discussion

In terms of strengths of the chosen method I should say that if the access to GPU and processing units is limited or not available these pretrained models can be used for generation of summaries which might not be as accurate as the complex pre trained models.

However, it comparison to the SeqCo model in the paper, it can be understood how effective and improving usage of contrastive learning is in sequence to sequence learning.

When we even come across the fine-tuned model we can't see any improvement. These very low ROUGE scores suggest that the summaries generated by the model are nonsensical or do not reflect meaningful content from the source text. This usually happens because of different factors during training and fine-tuning. One possible reason could be insufficient fine-tuning. The model might not have been trained enough on the dataset, as was mentioned in before, due to lack of access to GPU the model was only fine-tuned on 50 documents.

Poor initialization during reinforcement learning can create issues as well. If reinforcement learning starts with a poorly initialized model and there is no supervised fine-tuning beforehand, the outputs can become meaningless. Although using ROUGE scores as rewards in reinforcement learning is helpful, it might not provide strong enough signals for improvement, especially when the summaries have little or no overlap with the references.

There is a lack of factuality in BART and T5, since as it is noticeable in the summary generations, some of the statements do not come directly from the documents that have been given to be summarized. In result the models have problems with semantic faithfulness, meaning the summary must stay true to the original content.

The other issue that can be addressed is the float and coherence of the generated summaries readability and coherence are crucial. A good summary should flow naturally, be easy to read, and keep the audience engaged. These models produce summaries that may be informative but lack this smooth flow. The paper's approach, SeqCo, directly tackles these issues, making the summarization process more reliable and readable.

References

1. RushAM,ChopraS,WestonJ .A Neural Attention Model for Abstractive Sentence Summarization[J]. arXive-prints,2015
2. BahdanauD, BrakelP, XuK, etal. An Actor-Critic Algorithm for Sequence Prediction.
3. M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," arXiv preprint arXiv:1910.13461, 2019.
4. M. R. Ramadhan, S. N. Endah, and A. B. J. Mantau, "Implementation of Textrank Algorithm in Product Review Summarization," in ICICoS 2020 - Proceeding: 4th International Conference on Informatics and Computational Sciences, Institute of Electrical and Electronics Engineers Inc., Nov. 2020.
5. C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," arXiv preprint arXiv:1910.10683, 2019.
6. S. Badhe, M. Hasan, V. Rughwani, and R. Koshy, "Synopsis Creation for Research Paper using Text Summarization Models," in 2023 4th International Conference for Emerging Technology, INCET 2023, Institute of Electrical and Electronics Engineers Inc., 2023
7. Cruz, D., Pona, E., Holness-Tofts, A., Schmied, E., et al. (2023). Reinforcement Learning Fine-tuning of Language Models is Biased Towards More Extractable Features. arXiv.