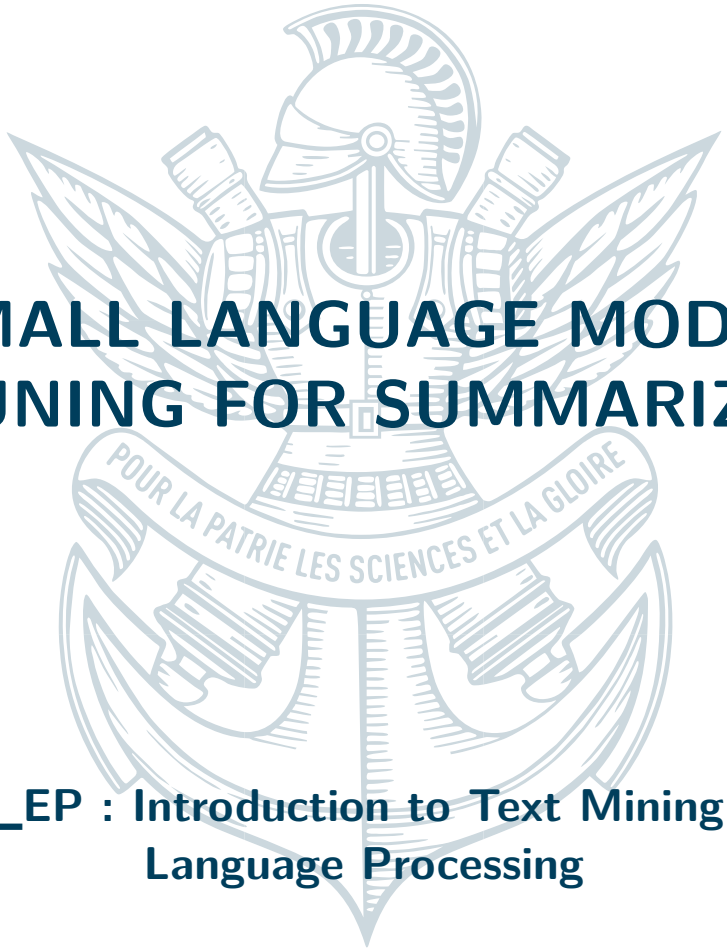

SMALL LANGUAGE MODEL FINETUNING FOR SUMMARIZATION

CSC_52082_EP : Introduction to Text Mining and Natural Language Processing

August 18, 2025

Ayoub Melliti, Gabriel Mercier

CONTENTS

INTRODUCTION

Text summarization is a fundamental task in Natural Language Processing (NLP) that aims to condense a long piece of text into a shorter, coherent version while preserving its essential meaning. This task has gained significant attention due to the exponential growth of digital content, necessitating efficient information extraction methods.

With the recent advancements in Large Language Models (LLMs), state-of-the-art models have demonstrated impressive capabilities in generating high-quality summaries. However, deploying these models for real-world applications often faces practical challenges, including computational constraints and response time requirements. In this project, we explore the fine-tuning of small-scale language models for text summarization, aiming to balance performance and efficiency.

Our work is structured into two main parts. First, we generate a high-quality dataset of French text summaries using a large pre-trained model, Qwen2.5-32B-Instruct, and apply preprocessing techniques such as quantization and prompt engineering to optimize inference. Second, we conduct a comparative study of different small-scale models, including a decoder-only model (Qwen2.5-0.5B-Instruct) and encoder-decoder architectures (T5-base and mT5-base), assessing their effectiveness in summarization tasks under resource-constrained conditions.

To achieve optimal performance while mitigating hardware limitations, we employ Low-Rank Adaptation (LoRA), a parameter-efficient fine-tuning method. LoRA enables us to train models with significantly reduced memory consumption, making it possible to deploy summarization systems on limited hardware. We investigate the impact of LoRA's rank on model performance, exploring trade-offs between training efficiency and output quality.

We conduct extensive evaluations using standard metrics such as ROUGE and BERTScore to measure the summarization quality. Our results highlight the strengths and weaknesses of different architectures and fine-tuning strategies. Specifically, we assess how LoRA influences the trade-off between model size and performance and determine the optimal configurations for summarization tasks under constrained computational resources in this context.

1 DATA GENERATION

The first part of the project consists of creating a dataset with long texts in French, and a **high quality summary** for each one, using a Large Language Model. We worked with the **MLSUM dataset** and **Qwen2.5-32B-Instruct**. To accommodate the hardware constraints imposed by a single GPU limited to 16 GB of VRAM, we opted to use a lab computer equipped with an **NVIDIA RTX A4000 GPU** featuring 16,376 MiB of VRAM. This choice enabled us to move away from relying on Google Colab's free tier, which could abruptly terminate training sessions without allowing us any control over the interruption.

1.1 Dataset choice

We chose to work with the Multilingual Summarization Corpus [5] from which we extracted 5000 texts. The texts of this corpus are extracted from articles of online newspapers. Their token length is 927 on average, and ranges from 100 tokens up to 6000 tokens.

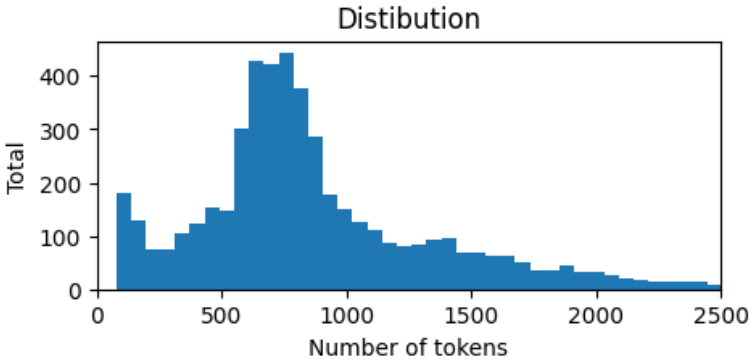

Figure 1: Distribution of the number of token in the extracted texts

One typical example of a text we will summarize is this article (`dataset[246]`) of length about 1000 tokens :

Des universitaires américains ont découvert que Twitter pouvait prédire avec une fiabilité de 90 % l'évolution du principal indice de la Bourse de New York, le Dow Jones, avec plusieurs jours d'avance. [...]

1.2 High quality summary generation

We implemented Qwen2.5-32B-Instruct in order to create high-quality summaries of our texts. To reduce the ressources needed to run the summary generation, we applied **4-bit quantization** to our large language model (LLM) using the *BitsAndBytes* library. Specifically, we employed the NF4 (normalized float4) quantization type with double quantization enabled. Computations during inference were performed in half-precision (FP16).

1.2.1 • Prompt engineering

The LLMs in general, and Qwen2.5-32B itself, are not created for summarization specifically. With our resources, it is not possible for us to finetune such models for the specific task. Thus, these LLMs, and even more with Qwen, which is an autoregressive model, can only be used to complete text. Thus, the first words used to introduce the text to sum up are really important : namely, we needed to do **prompt engineering**.

Our prompt must be in French, and must guide the model through the specific task of summarization. We first tried some very simple prompts such as : 'résume ce texte' (translation: 'summarize this text') and the results were not really satisfying. With multiple try and error iterations, we incorporated in the prompt instructions to specify the task, in order to improve (with respect to our appreciation over 10 examples picked randomly) the summary quality. We specified the language, the maximum size and some rephrasing instructions. We added only instructions for things that were missing in the outputs (for instance rephrasing) or for things that were irrelevant in the outputs (for instance opinions).

- **"without adding opinions or comments"**: The model frequently appended unnecessary notes after summaries, negatively affecting dataset quality. Although this instruction reduced the issue, it wasn't sufficient alone. Observing that Qwen2.5-32B-Instruct consistently inserted line breaks before notes, we implemented a smart truncation after these line breaks.
- **"100 words maximum"**: Repeating this constraint twice notably improved performance, as the model initially struggled to produce summaries under 100 words.

Our final prompt is the following : "Résume précisément le texte suivant en français en 100 mots maximum. Concentre-toi sur les points essentiels sans ajouter d'opinions ni de commentaires. Évite les phrases inutiles et

reformule les idées clairement. [text] Résumé concis et structuré (100 mots maximum) :"

Translation: "Summarize precisely the following text in French, using a maximum of 100 words. Focus on essential points without adding opinions or comments. Avoid unnecessary sentences and clearly reformulate the ideas. [text] Concise and structured summary (100 words maximum):"

1.2.2 • Inferences

Performing inference with this model under our hardware constraints proved challenging. Indeed, generating the complete dataset required **more than 25 hours**. After multiple failed attempts, we implemented periodic checkpoints every 400 texts and used *tmux* (a terminal multiplexer) to ensure continuity across sessions. An additional difficulty arose from the shared nature of the lab computers, accessible to all students: occasionally, the machine had to be restarted by other users, resulting in the loss of unsaved progress.

1.2.3 • Summary postprocessing

Even with the large language models and with a really precise prompt that insists on the conciseness of the summary, we had to cut the end of some summaries. Thus, **we truncated after 125 words**. The figure 2 shows the distribution of our summaries in length.

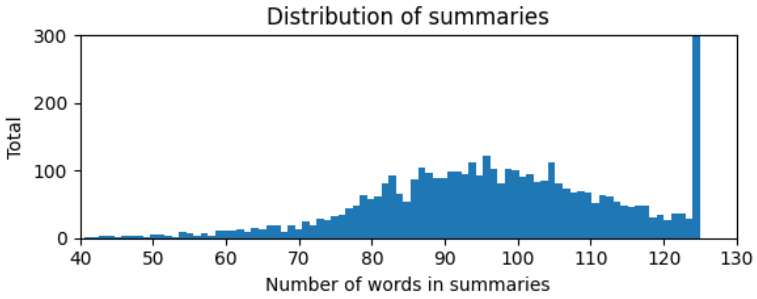

Figure 2: Distribution of summaries length. Summaries of 125 words and more were cut at this length

Then, we **removed from the dataset all the texts with more than 2500 tokens**. Indeed, later on, we encountered memory issues that were not linked to the parameter storage, the gradient, or anything inside the model. The memory consumption was going above 16GB during the training, but necessarily at the moment the training started. Monitoring the RAM, the memory consumption was heavily varying during the training. We made the link with the fact that in our dataset a small fraction of the texts were more than 5000 tokens long, which is 5 times our typical text length. For this reason, we decided to exclude all text with length more than 2500 tokens from the dataset. The following figure shows the distribution of the number of tokens in the texts.

Finally, here is, for instance, the summary of the example text above :

Des chercheurs de l'université d'Indiana ont analysé 9,8 millions de tweets pour prédire l'évolution du Dow Jones avec 86,7% de précision. Ils ont mesuré l'"humeur collective" via deux outils d'analyse de sentiments [...]

2 SMALL MODEL COMPARISON

2.1 Different types of models

For text summarization tasks, which require generating coherent and concise outputs, two main types of language models are typically employed: decoder-only (autoregressive) models and encoder-decoder models. For LLMs, both types of models can generate high-quality summaries. However, **encoder-decoder architectures are specifically designed for sequence-to-sequence tasks**, and according to the literature on text summarization, encoder-decoder models generally outperform decoder-only models.

- A **decoder only, or auto-regressive model**, completes the prompt by generating iteratively the next token based on the information it already has (the prompt and the input text). Qwen2.5-0.5B-Instruct is an example of a such model.
- **An encoder-decoder model**, on the other hand, processes the input text differently. The encoder first converts the entire input into a fixed-length representation, which the decoder then uses to generate the summary. This architecture allows the model to better capture the overall context before generating any output, which can be beneficial for summarization tasks, especially for smaller models where memory efficiency and information retention are crucial. T5 (Text-to-Text Transfer Transformer) is a well-known example of an encoder-decoder model (mainly for English). and Mt5 is the multilingual version of t5.

As encoder-decoder models might perform better than decoder-only models for summarization, as they do not rely solely on auto-regressive token generation but rather leverage a more structured encoding of the input, we conducted experiments comparing the performance of a decoder-only model (**Qwen2.5-0.5B-Instruct**) with encoder-decoder models of similar size (**mT5-Base** and **T5-Base**).

2.2 Finetuning with low rank adaptation

• Resources constraints

To fine-tune the model with limited resources, we applied **double quantization** along with Low-Rank Adaptation (LoRA) [1] at **rank 16** (for all models, ensuring a fair comparison) on the attention projection layers ("q_proj", "k_proj", "v_proj", "o_proj"), which correspond respectively to the query, key, value, and output transformations within the attention mechanism. We targeted **roughly 1∼2 %** of the total model parameters using this strategy and fine-tuned Qwen2.5-0.5B-instruct (0.68%): using the same prompt engineering methodology as Qwen2.5- 32B-Instruct, mT5-Base (0.70%), and T5-Base (2.27%) without any prompt engineering.

• Splitting

Before starting training, we split our dataset into training, validation, and test sets with proportions of **60%, 20%, and 20%**, respectively. This resulted in 2,655 samples for training, and 885 each for validation and testing. The total number of samples ended up being less than 5,000 (11.5% reduction) due to truncation performed during post-processing as described above. Ideally, this truncation should have been conducted before dataset generation; however, because of the complexity and time constraints involved, we proceeded with the truncated dataset.

- **Training**

For training, we fine-tuned the models over **3 epochs** (sufficient for fine-tuning LLM's [8]) using a **learning rate of** 2×10^{-4}. Due to hardware constraints, we set a per-device batch size of 1 with gradient accumulation over 4 steps, effectively simulating a batch size of 4. The optimizer used was 8-bit Paged **AdamW** [3], combined with a cosine learning rate scheduler and a warm-up ratio of 5 %. Additionally, training was conducted in BF16 precision to enhance efficiency. Model evaluation was performed at the end of each epoch.

The training lasted between **45 min and 2 hours** for each of the three models. We followed the evolution of the model performance by computing a loss on both the train and the validation dataset (see Appendix for additional details).

2.3 First evaluation and model choice

The evaluation of automatic summarization relies on metrics that compare the generated summary with our SLM to one or more human-written references. Among these metrics, **ROUGE** (Recall-Oriented Understudy for Gisting Evaluation) [2] and **BERTScore** [7] are widely used.

ROUGE measures the n-gram overlap between the generated summary and the reference summary, providing an indication of how much of the reference text is preserved in the generated output. The main ROUGE variants include:

- **ROUGE-1/2**: Measures the overlap of unigrams and bigrams.
- **ROUGE-L**: Captures the longest common subsequence (LCS) between the generated and reference summaries, reflecting the preservation of sentence-level structure.
- **ROUGE-LSum**: A variant of ROUGE-L adapted for evaluating summaries at the document level.

BERTScore evaluates the semantic similarity between the generated and reference summaries rather than just lexical overlap. It leverages contextual embeddings from transformer-based models (e.g., BERT) to compute similarity at the token level. BERTScore provides three main scores:

- **Precision**: Measures how many words in the generated summary are semantically similar to words in the reference summary.
- **Recall**: Measures how well the reference summary is covered by the generated summary in terms of semantic similarity.
- **F1-score**: The harmonic mean of precision and recall, providing a balanced evaluation.

While ROUGE remains a standard metric for summarization evaluation due to its simplicity and efficiency, BERTScore offers a more nuanced assessment by capturing semantic meaning rather than just surface-level word overlap.

We computed these scores before and after fine-tuning for each model: Qwen2.5-0.5B-Instruct, mT5-Base, and T5-Base. The evaluation phase lasted about the same time as the training, namely $\sim$1 hours for each. The figure 3 recapitulates all the scores of our three models after the LoRA fine-tuning described above.

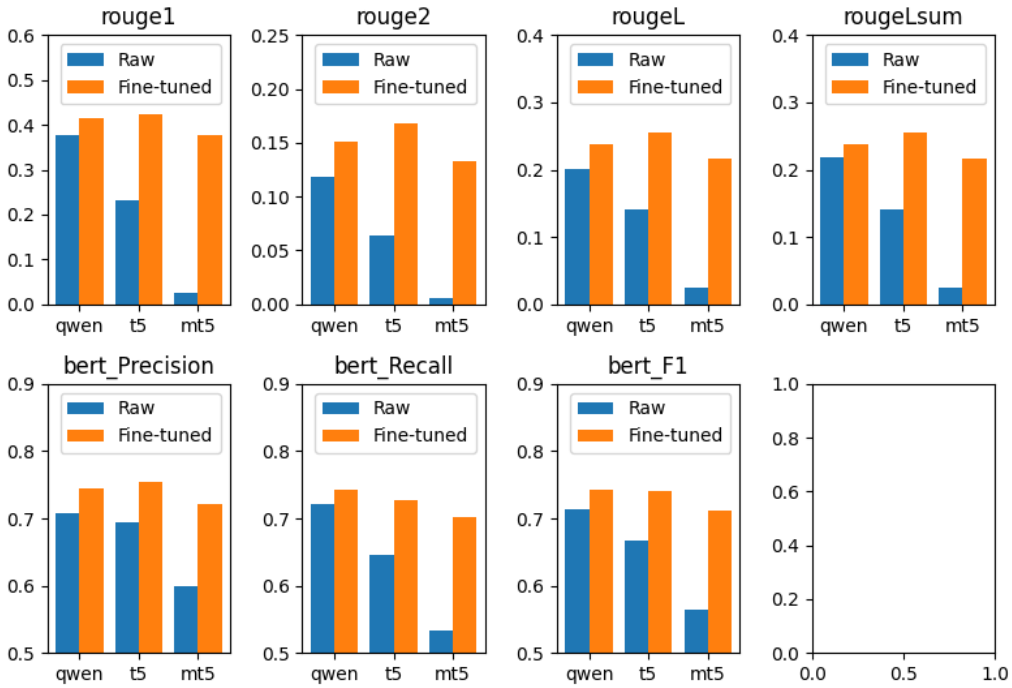

Figure 3: ROUGE and Bert scores of the three small language models before and after finetuning

Note that the scores of mT5 without fine-tuning are not really relevant, since it seems that the model is not generating a summary at all before fine-tuning (confirmed with examples). For all models, the fine-tuning significantly increases the performance in all metrics.

After fine-tuning with LoRA and $r = 16$, the model that gives the **overall best scores is T5**. However, the different metrics are not unanimous, and the difference is not big. We wanted to pick a single model to investigate more possibilities of improving the performance, other than the choice of the best initial model.

We may explain the strong performance of Qwen2.5-0.5B-Instruc by considering that the LLM used to generate the dataset is also a Qwen model. This suggests that both models were likely trained on similar datasets and share a comparable architectural framework. As a result, they may exhibit similar patterns in their outputs.

Thus, we decided to continue our project by trying to **optimize our process with T5 only**. T5 gives the best overall performance; it is an encoder-decoder model which is better according to the literature, and it is the **smallest of our three models**. Thus, we followed the Occam's Razor principle. To continue our investigations, we changed the rank of the LoRA configuration and the target layers fine-tuned.

3
MORE T5 FINETUNING

T5-base is a model from Google [4], with about 200M parameters. We wanted to further investigate the effect of the fine-tuning with LoRA on the performance of T5 on our dataset. In particular, we wanted to show the trade-off between training time, performance, and simplicity, measured with ROUGE and BERT score.

3.1 Effect of the rank of the perturbations

By varying only the rank of the perturbations, we can shed light on this tradeoff. We conducted multiple fine-tunings of T5 on our dataset with ranks $r = 8, 16, 32, 64, 128$. The figure 4 shows the evolution of the scores when the rank changes.

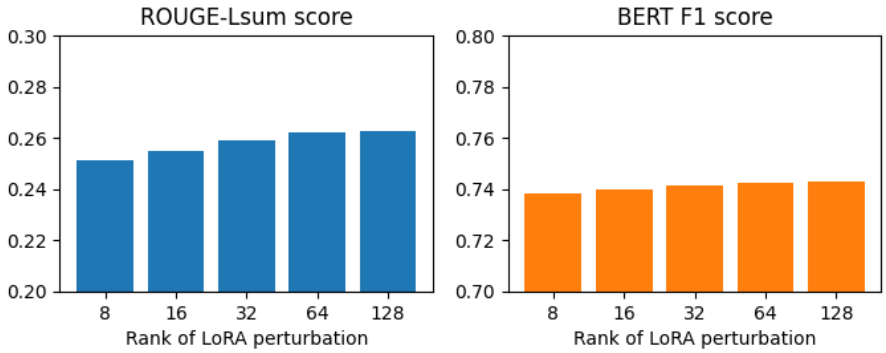

Figure 4: Effect of the rank of the LoRA perturbation on the performance of the fine-tuned T5 model with respect to ROUGE-LSum and BRT F1 scores

- The performance evolution indicates that varying the rank of LoRA perturbations from $r = 8$ to $r = 128$ has **minimal impact** on model performance. This stability implies that even low-rank adaptations capture most of the relevant task-specific information.
- Moreover, the **training times were also very similar** (50 min) for all ranks. suggesting that increasing the rank does not necessarily lead to a significant improvement in summarization quality (no more than 1% improvement). It is important to note that we used a free access computer from the labs, so our measures of the training times are not very precise.

These findings suggest that, for this specific summarization task, using a **lower rank** (e.g., $r = 16$) is a reasonable choice (following once again the Occam's Razor principle).

3.2 Effect of LoRA Target Modules

In this section, we analyze the impact of applying LoRA to different sets of target modules within the model. Above, fine-tuning was performed on the attention block by applying LoRA to the query, key, value, and output projection layers (*attn*). To further investigate the effect of LoRA placement, **we also fine-tuned the model on the dense feedforward layers** (*dense*) and compared it to a setup where LoRA was applied to both the attention and dense layers (*full*) and a previous fine-tuned model on the attention block with $r = 128$ (*attn(128)*).

The performance of these configurations is evaluated using ROUGE and BERTScore, with a LoRA rank of $r = 16$ (see Figure 5).

Our observations indicate that applying LoRA to the dense layers alone (1.9% of trainable parameters) results in lower performance compared to fine-tuning the attention block (2.3%). However, when both the attention and dense layers are fine-tuned together (4.1%), the model achieves significantly improved performance.

Interestingly, this **combined configuration matches the performance of our best previous model**, which was fine-tuned with $r = 128$ (15.7%) but only on the attention block. This suggests that distributing the

adaptation across both the attention and dense layers allows the model to reach a similar level of effectiveness while using a significantly lower parameter budget.

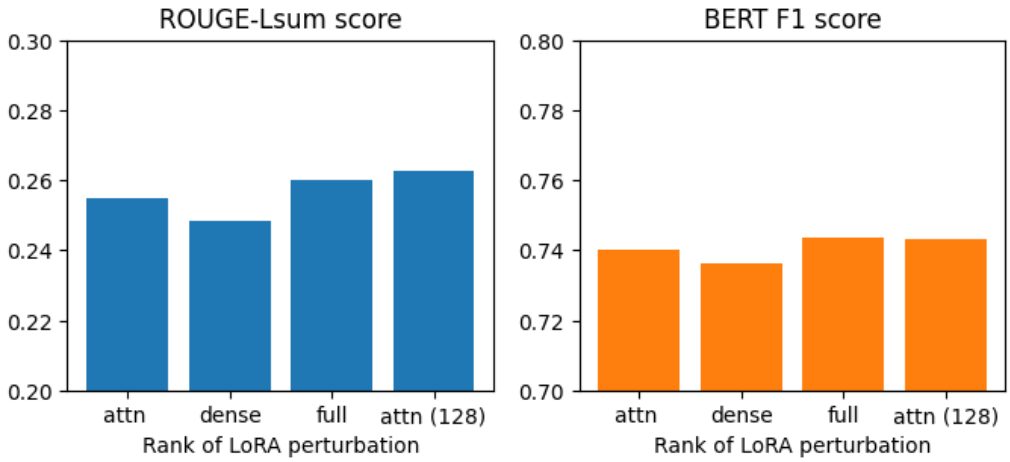

Figure 5: Effect of the target modules of the LoRA perturbation on the performance of the fine-tuned T5 model with respect to ROUGE-LSum and BRT F1 scores for rank 16 and 128 (last bar)

These findings suggest that while attention mechanisms play a dominant role in model adaptation, incorporating feedforward layers into the fine-tuning process enhances representational capacity, allowing the model to leverage its full potential with a lower rank.

CONCLUSION

In this project, we explored the feasibility of using small-scale language models for text summarization while maintaining a balance between efficiency and output quality. By leveraging LoRA for parameter-efficient fine-tuning, we demonstrated that it is possible to significantly reduce memory consumption while preserving strong summarization performance. Our comparative analysis of decoder-only and encoder-decoder models highlighted key differences in their summarization capabilities under computational constraints. Additionally, we showed that varying the rank of the LoRA perturbation in the attention block of T5 has a small impact on the performance, and that targeting the whole model (attention block and dense layers) is more effective.

Retrospectively, there are areas where our approach could be improved. First, refining our dataset selection by filtering 5,000 texts of similar length (between 500 and 1,500 tokens) would ensure greater coherence and consistency. Second, in the summary generation step, we could filter out outputs shorter than 125 words and implement a retry mechanism for excessively long summaries. Finally, our evaluation could be enhanced by incorporating an LLM-as-a-Judge approach [6], for instance, allowing us to obtain more nuanced assessments of summarization quality beyond traditional metrics. These refinements could further strengthen our approach and improve the applicability of small-scale models for real-world summarization tasks.

REFERENCES

[1] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[2] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[3] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[4] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[5] Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, and Jacopo Staiano. Mlsum: The multilingual summarization corpus, 2020.

[6] Hui Wei, Shenghua He, Tian Xia, Andy Wong, Jingyang Lin, and Mei Han. Systematic evaluation of llm-as-a-judge in llm alignment tasks: Explainable metrics and diverse prompt templates, 2024.

[7] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert, 2020.

[8] Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences, 2020.

APPENDIX

Percentage of trainable parameters for T5-base

- **target modules : `q_proj, v_proj, o_proj, k_proj`**

$$r = 8 : 1.149\%$$
$$r = 16 : 2.273\%$$
$$r = 32 : 4.445\%$$
$$r = 64 : 8.513\%$$
$$r = 128 : 15.690\%$$

- **target modules : `wi, wo` (dense layer)**

$$r = 16 : 1.901\%$$

- **target modules : `wi, wo, q_proj, v_proj, o_proj, k_proj` (full)**

$$r = 16 : 4.091\%$$

Loss function decrease during training

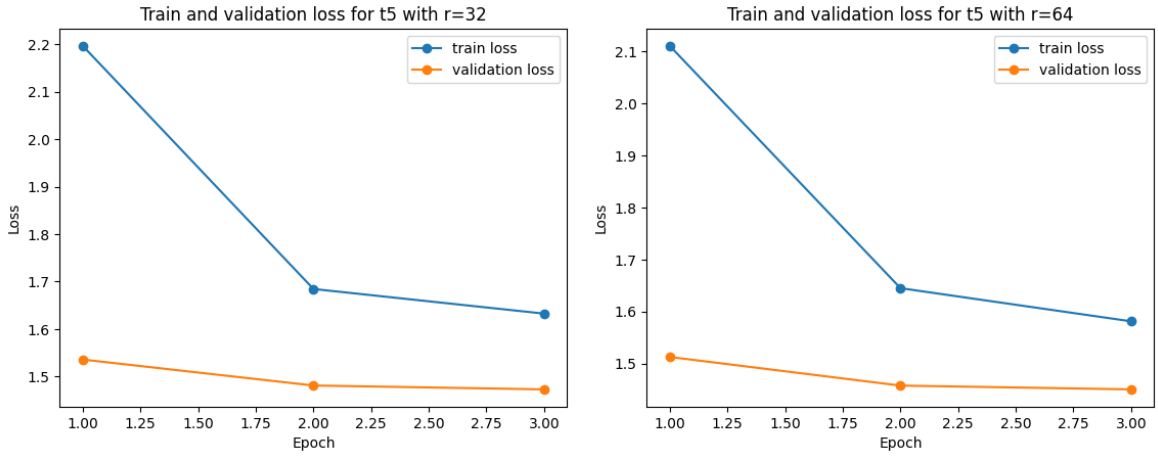

Figure 6: Loss function evaluated on training and valudation dataset for T5 with $r = 16$ and $r = 32$