Text-Mining & NLP

Report

Élèves :
Victor MICHA
Yann MILLET

Enseignants :
Michalis VAZIRGIANNIS
Davide BUSCALDI
Guokan SHANG
Hadi ABDINE

14 mars 2025

Table des matières

1 Introduction

The rapid advancement of NLP has lead to the development of powerful language models capable of many tasks, including text summarization. However deploying large models in resource-constrained environments remains a challenge. This project aims to address this by fine-tuning a compact language model for summarization using a dataset of 5000 articles sourced from French Wikipedia. We began by generating high-quality summaries with the Qwen2.5-14B-Instruct model, a 14-billion-parameter large language model known for its strong linguistic capabilities, which efficiently runs on an NVIDIA T4 GPU. These summaries then served as training data to fine-tune the smaller Qwen2.5-0.5B-Instruct model, with 0.5 billion parameters.

2 Background

Large language models (LLMs) have transformed natural language processing, but their computational demands often make them impractical for smaller entities with limited resources. In this project, we use several methods to show that good performance can be obtained with constrained resources, and we describe these methods in this section. Fine-tuning involves adapting a pretrained model to a specific task, like summarization, by further training it on task-specific data, in order not to start from scratch. Because annotating a big dataset is long and painful, using a bigger LLM to generate synthetic data is a good compromise between quality and quantity. To make the learning of Language Models efficient, LoRA is a PEFT technique that modifies only a small subset of the model's parameters, reducing memory and compute requirements. Quantization, such as reducing a model from 16-bit to 4-bit precision, reduces memory consumption as well. Distillation takes this further by transferring the knowledge from a large model (e.g. 14B parameters) to a smaller one (e.g. 0.5B parameters), while keeping much of the original knowledge in a compact form.

3 Dataset

3.1 Selection

At first we wanted to chose a dataset that will be useful for us in the real life, like scientific articles. However, because of the imposed model size, it appeared obvious that our results will never be at the SOTA level and that it was preferable not to lose too much time gathering a complicated dataset.

That's why we decided to collect Wikipedia articles in french. We've chosen not to specialize the dataset on a precise domain, which could have improved the results by hyperspecializing the model. We made this choice for the sake of the code simplicity.

After gathering 5000 articles of a size between 500 and 4000 words, we cleaned the texts in order to get rid of the section titles. Indeed, Wikipedia encodes title between "===" tags, which confuses the language models, as they better understand natural language.

Idealy we would have liked to deleted sections like "Notes and references" because it is no useful content. But since there is no real consistency across articles, some articles have "Notes" title, some "References", "External Links", combination of these, it would have been an non-exhaustive cleaning, so we decided to bet on the fact that such sections would fall under the 4000 words limit for most articles (and for the other, too bad).

Finally, we save this into a json file in order to share and reuse it easily.

3.2 Annotation

In order to fine-tune our Small Language Model for a precise task, we need to create annotations for each data sample, which in our case signifies generating a "goal" summary for each article. We use for that another model, larger than the one we will fine-tune but which fits in the memory of a T4 GPU.

We have been informed later that we could have used a bigger model API because the imposed limit on the annotation model size was more to avoid us paying for bigger GPU than just forcing the model size within 16GB. It was too late for us to re-annotate and fine-tune again so we used "Qwen2.5-14B-Instruct" to generate the summaries. Our first try was to use "Qwen2.5-32B-Instruct" but we couldn't manage to make it fit into the T4 GPU, even when quantizing as much as we could.

We quantized the model in 4-bits and we compute in 16-bits. We used a temperature of 0.7 for a relatively high creativity in the generations, and a nucleus sampling of 0.7, which is relatively high but still in a average range in our opinion. We did some other tests by generating few summaries with other parameters (temperature in .2, .5, .7, nucleus in .5, .7) and we qualitatively evaluated (0.7, 0.7) to be the best pair of parameters. We are aware that this evaluation process is probably biased because of the few number of summaries evaluated, but we needed to see if there were high differences between the different parameters values. Despite some values being (logically) less creative, the quality for the worst parameters was not as bad as we thought, with less repeating or incoherent words than in previous labs of this course.

Here is the prompt that we used :

Code 1 : Prompt for generating summaries python

```python
prompt_prefix = "Résume en quelques phrases en français l'article suivant.
↪  Il faut que le résumé soit court et représente les informations les
↪  plus importantes."

prompt_template = """<human>: {prompt_prefix}

ARTICLE: {article_text}

<assistant>:"""
```

The format of ARTICLE is not between tags like <human> and <assistant>, we didn't do it on purpose and it didn't seem to penalize the quality of the results. The styling of the prompt is less important to the model than it could be visually for us.

When assessing the quality of summaries with the best parameters pairs, we wrote a piece of code to review some summaries (mainly for consistency and meaning of the sentences) and sort the bad ones in order to resummarize the samples. We didn't put that into practice because there were no real difference from two summaries of the same article, and the random factor of the generation was not as strong as anticipated.

4 Data Splitting

In order to use our data for fine tuning, we split it in 3 parts : a train, a validation and a test sets, in 80/10/10 proportions (4000, 500 and 500 samples respectively).

The idea here is to train the model on the train data (nothing new here), and we assess potential overfitting of the model at each step with the validation set. Straightforwardly we will test after each epoch on the validation set.

We did save these split in json file in order to share them more easily, but also ensure that after splitting we always use the same data for training/validating/testing. This is quite redundant with fixing a random seed at the beginning of the code if only for reproducibility.

5 Fine-Tuning

For this part of the project, we looked for resources other than the lectures of the courses in order to get different point of view ([1], [2]). For fine-tuning Qwen2.5-0.5B-Instruct, we used LoRA with 4-bit quantization for an efficient processing of memory. This was used in the lab 6 of the course as well, we took inspiration from it because it worked well. LoRA was applied to attention projection layers q_proj and v_proj. We applied LoRA

on a lot of layers, which makes the model go from 136178560 learnable parameters among 315119488 (43% of the parameters are learnable) to 17596416 learnable parameters (down to 5% of the parameters). Additionally, with 4-bit quantization (using bitsandbytes) we can load and train the model on our limited hardware and still achieving nice improvements.

We use the Trainer API in order to make the management of the fine-tuning easier. We added checkpoint every 200 iterations as well.

In order to prepare the dataset, we created a structured prompt where each training example consists of a human prompt asking for a summary, then the article text and finally the corresponding assistant response which is the expected summary. This format is nice to fit to instruction-tuned models like our. The tokenizer was configured to pad sequences to a fixed length (1024 tokens) to standardize inputs for training. We use a batch size of 2, with a gradient accumulation of 2, which puts the effective batch size to 4. We did that because the GPU was too small to handle batch size of 4. For the optimization, we used the AdamW optimizer with a learning rate of 2e-4 and a warmup of 100 iterations. This is maybe a bit high for a learning rate, but if we understood well this discussion [3], a gradient accumulation can be compensated by a lower learning rate with a larger batch size, so we thought that a higher learning rate would not be a big problem with the gradient accumulation. Finally we used fp16 precision to reduce even more the memory consumption.

The training process was 3 epochs with evaluation every 50 steps and checkpoints every 200 steps. This way we can track progress and avoid overfitting. Same than the 14B model for the annotation generation, the model's configuration has a temperature of 0.7 and nucleus sampling of 0.7.

6 Model Evaluation

We monitored the fine tuning of the model by tracking training and validation loss, and measured the quality of the generated summaries using ROUGE metrics.

As we can see from Figure 1, the training loss exhibits erratic fluctuations, ranging from 2.7379 at step 50 to 2.2637 at step 3000. In contrast, the validation loss steadily decreased from 2.6352 to 2.3417 over 3000 steps, eventually plateauing around 2.34-2.35 after approximately 2000 steps. Since the loss on the validation set is no longer decreasing, we can infer that the model has been trained enough (and training it longer would risk overfitting it).

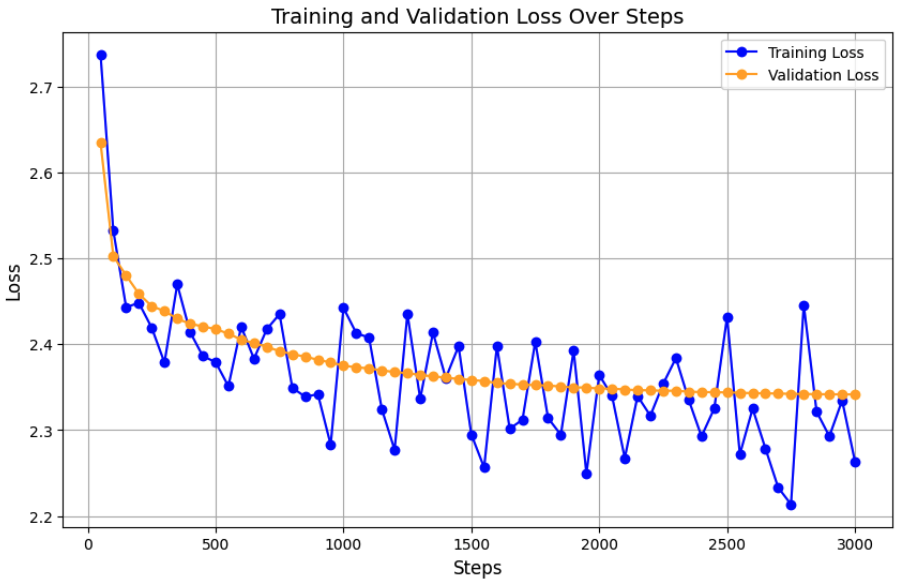

FIGURE 1 – Fine Tuning Losses

6.1 ROUGE

To evaluate the model, we use the ROUGE scores to compare the summaries. But the outputs of the model contain the entire prompt, with all the keywords : "<humain>", "<article>", and "<assistant>". However, the summary we are interested in is after the "<assistant> :" part, so we only pay attention to this part when comparing generated summaries with the actual summaries (from the 14B model). We explain the criterias we use to judge a summary's quality in the following :

ROUGE-1 measures the overlap of individual words (unigrams) between the generated and reference summaries, reflecting basic word-level similarity.

ROUGE-2 assesses the overlap of two-word phrases (bigrams), indicating how well the model captures short sequences or exact phrasing.

ROUGE-L evaluates the longest common subsequence, capturing structural similarity regardless of word order.

ROUGE-Lsum extends this to sentence-level subsequences, providing insight into broader contextual alignment.

For computational efficiency, we tested the model on a subset of 200 samples from the 500-article test set, generating summaries and comparing them to the reference summaries using ROUGE metrics. In the following subsections, we compare the model's performances before and after the fine tuning.

6.1.1 ROUGE - Pre Fine Tuning

For the model pre fine-tuning, the results yielded ROUGE-1 of 0.3178, ROUGE-2 of 0.0975, ROUGE-L of 0.1806, and ROUGE-Lsum of 0.2215, indicating a baseline performance with moderate unigram overlap but limited ability to capture bigram sequences

and structural coherence. Overall, these results are expected from a small language model of 0.5B that is not yet fine tuned for a specific task.

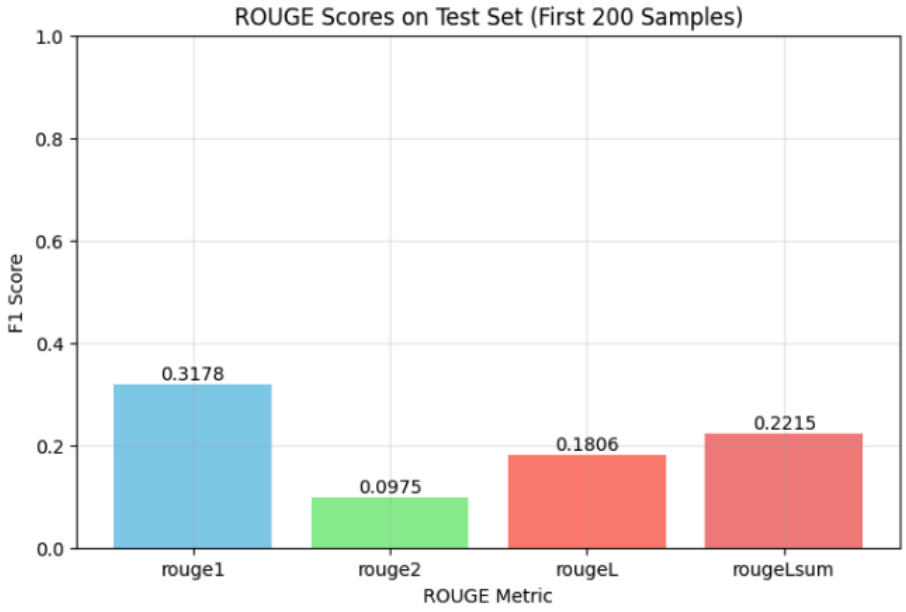

FIGURE 2 – Pre fine tuning ROUGE scores

6.1.2 ROUGE - Post Fine Tuning

For the model post fine-tuning, the results yielded ROUGE-1 of 0.3952, ROUGE-2 of 0.1435, ROUGE-L of 0.2309, and ROUGE-Lsum of 0.2726, with enhanced unigram overlap, better bigram sequence capture, and increased structural similarity.

We notice that all the rouge scores have increased compared to the results of the pre fine tuned model, which means that it has learned to improve its summarization capabilities through fine tuning on the synthetic high quality summaries.

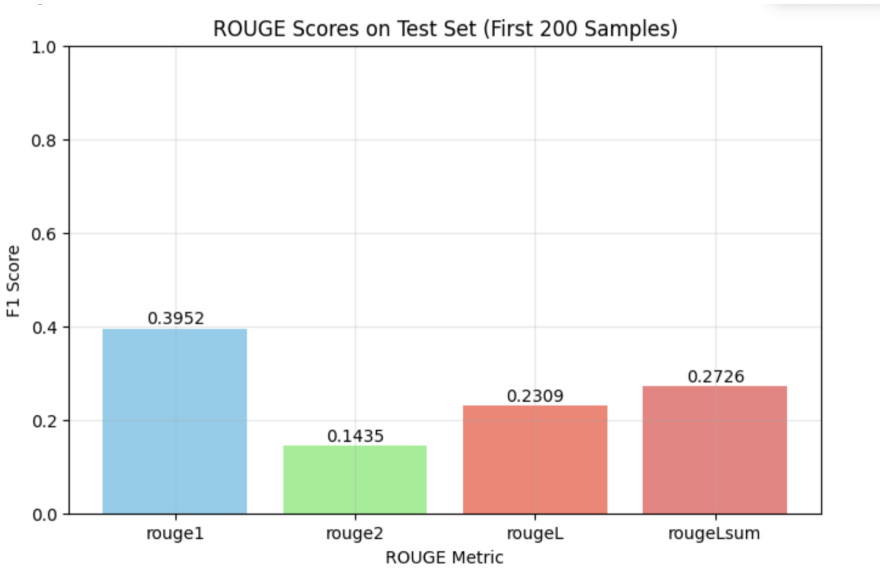

FIGURE 3 – Pre fine tuning ROUGE scores

[3]

7 Conclusion

This project showed that fine-tuning the Qwen2.5-0.5B-Instruct model on a summarization task moderately boosted performance. Through distillation from Qwen2.5-14B-Instruct, we effectively adapted the small model, proving that even with limited setups (free Google Colab resources), one can leverage efficient techniques to manage meaningful NLP results.

8 Bibliographie

[1] *How to fine-tune Small Language Models*. URL : https://medium.com/@liana.napalkova/fine-tuning-small-language-models-practical-recommendations-68f32b0535ca. (Date de consultation : 20/02/2025).

[2] *HuggingFace AutoTrain documentation*. URL : https://huggingface.co/docs/autotrain/llm_finetuning. (Date de consultation : 24/02/2025).

[3] *Gradient Accumulation thread on Reddit*. URL : https://www.reddit.com/r/MachineLearning/comments/wxvlcc/d_does_gradient_accumulation_achieve_anything/. (Date de consultation : 02/03/2025).