DSTI - MSc. Data Science & AI

Deep Learning Project:

DEVELOPMENT OF AN NLP AI REPORTING AGENT

AI generated logo 😉

Matias Bottarini (SPOC S-23)

Samd Guizani (SPOC A-23)

TABLE OF CONTENTS

SUMMARY[1]

This project is set out to build an AI agent capable of automatically generating well-structured incident reports from minimal user input. Using a standardized questionnaire (the 5W1H questions plus contingency actions), the agent produces full, readable reports for contexts such as pharmaceutical deviations and traffic accidents.

To achieve this, the authors created their own datasets because no suitable public ones existed. Over 200 synthetic cases using ChatGPT – 107 for pharmaceutical incidents and 102 for road accidents – each paired with key information fields. These datasets formed the backbone for evaluating models.

The team then evaluated a wide range of open-source large language models from 135M to 3B parameters (SmolLM, Qwen2.5, Llama-3.2, GPT-2-XL, phi-2), testing different prompting strategies and generation parameters. Several text-similarity metrics were tried (BLEU, ROUGE, BERTScore, Bi-/Cross-Encoder), but Cross-Encoder similarity emerged as the most informative for ranking models. Custom Python classes were built to automate prompt creation, model loading, report generation, metric computation, grid searches, and later fine-tuning.

Deployment took two forms: a Google Colab demo notebook and a lightweight Streamlit web application hosted on Hugging Face Spaces. These allow users to choose a model, enter the key details of an incident, adjust prompts and generation parameters, and download the resulting report.

Model evaluation revealed that among small models, Qwen2.5-0.5B consistently achieved the best similarity scores (around 0.83–0.86), while among medium models Llama-3.2-3B reached roughly 0.89–0.90 on pharmaceutical reports and up to 0.935 on traffic accidents. Prompting with an example from the relevant domain (methods C and D) further improved results.

In a final phase, the team explored specialization by fine-tuning a small model, SmolLM2-360M-Instruct, on 800 and then 5,000 synthetic traffic accident cases using Supervised Fine-Tuning with QLoRA and PEFT. Remarkably, training on just 800 cases lifted the model's performance to about 0.88 similarity—close to much larger 3B-parameter models – while requiring only a fraction of the resources. Increasing the dataset size to 5,000 did not yield additional gains. An even smaller model (135M parameters) failed to achieve acceptable performance, and experiments with knowledge distillation also brought no improvement because the reporting task was too deterministic to benefit from the teacher model's "soft" knowledge.

In conclusion, Llama-3.2-3B-Instruct is the best choice when maximum performance is required out-of-the-box, achieving ~0.93 similarity on traffic accident reports. However, the fine-tuned SmolLM2-360M-

[1] The summary was AI generated. Reviewed and amended before inclusion in this report.

Instruct offers an excellent performance-to-size trade-off: roughly ten times smaller than Llama-3B but only about five percentage points lower in similarity. Qwen2.5-0.5B remains a solid lightweight option, especially if multilingual capabilities are needed. This work shows that with careful prompt engineering and small targeted fine-tuning, compact models can rival much larger ones for structured report generation.

I. INTRODUCTION

In many professional areas, documenting incidents is not only a best practice but often a legal requirement. Two examples can be mentioned:

- In the pharmaceutical industry whenever an unexpected event or a deviation occurs during a process such as manufacturing or quality testing, a report must be filed within the quality system documentation of the company.
- Road traffic is subject to incidents and accidents, and the police forces are required to describe the circumstances, particularly in case of material damage or casualties.

Therefore, these reports must be concise, clear and fact-based. In most situations, the text covers the following aspects:

- ***What*** has happened? A description of the incident.
- ***When*** did it happen? Time of occurrence or discovering the incident.
- ***Where*** did it happen? Location of the incident.
- ***Who*** was involved? Parties concerned by the incident (and their role).
- ***How*** did it happen? The immediate cause of the incident.
- ***Why*** did it happen? The root cause (if known) of the incident.
- What are the ***contingency actions***? Immediate consequences and actions to limit the impact of the incident.

In this project, the objective is to develop an NLP AI agent to automate the generation of such reports, based on information provided by the user through a standardized questionnaire. The user workload is minimal as the questionnaire only requires entering free text key information (key words or short sentences). The AI agent then generates a well-structured text, including a title and report body, that plainly describes the above aspects.

The project has the following deliverables:

- A GitHub repository containing all the code files and environment requirements to execute the agent. It can be found here: https://github.com/zBotta/reportingAgent. For information, this project has been developed using Google Colab resources.
- A deployed application which can be found in two ways:
    - either running the ***demo Colab notebook*** available on the GitHub repository (app/reportingAgent_demo.ipynb). The user must clone the repository and execute the demo using the requirements.txt environment.

- or using the ***online deployed Streamlit version*** here: ReportAgent - a Hugging Face Space by zBotta and SamdGuizani. NB: if the app is sleeping due to inactivity in Hugging Face Spaces, you would need to restart the space it and wait for building before using it.

II. PROJECT SCOPE AND PHASES

The project covers the following scope:

- ***Testing various open-source LLM models of small to medium size*** (~100 million to ~3 billion parameters). To do so, various prompting strategies have been developed and combined with different generation parameters in order to evaluate their impact on the quality of the reports. The quality of the AI generated texts was measured through text similarity metrics comparing them to reference reports.
- ***Model specialization***, attempting to use an existing small size model and training it on a small dataset, then evaluating if its performance improves. The motivation for specialization is to achieve performance that is close to larger models with the benefit of using a small model, requiring less resource footprint (computation, memory, portability on small devices…).
- ***Deployment of the reporting agent*** in the form of Colab notebook and an online web application.

The project has been divided into 4 phases (Figure 1):

- ***Phase 1 – Data collection***
- ***Phase 2 – Model selection and optimization of text generation parameters***
- ***Phase 3 – Deployment***
- ***Phase 4 – Model specialization*** (not represented but considered as an alternative approach to deploy new NLP models, closely matching the user needs and requiring less resources)

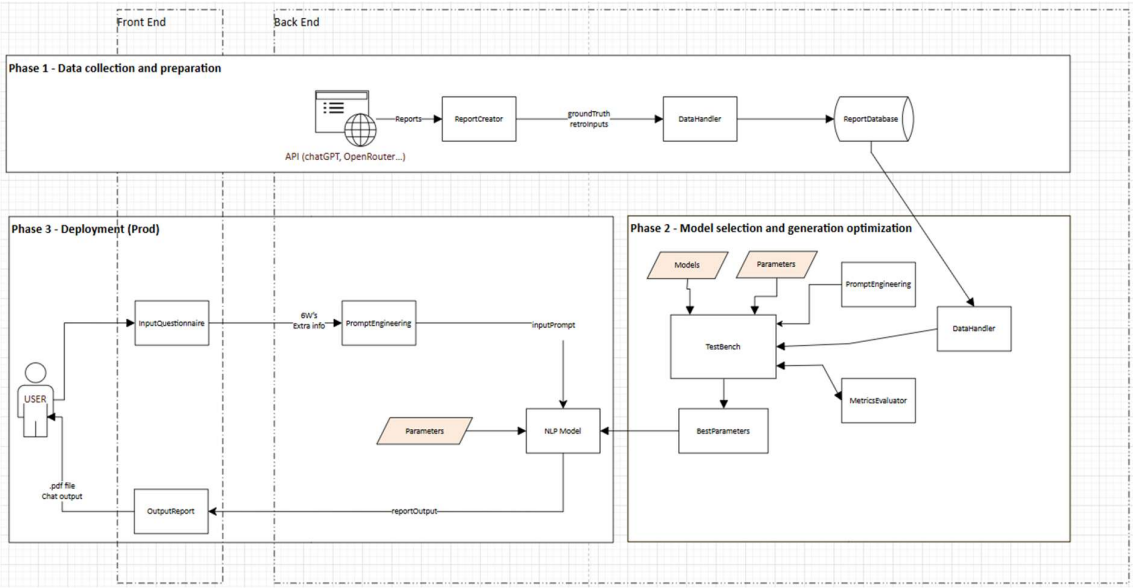


Figure 1 – Outline of project phases and components

The following sections provide details on the tasks and development related to each phase.

A. PHASE 1 – DATASET COLLECTION

A key challenge to develop this AI reporting agent is the availability of datasets. Indeed, testing models require access to data where incident reports are available as well as their underlying key information (what, when, where, who, how, why and contingency actions).

In absence of publicly available datasets, a decision was made to revert to synthetic datasets produced through GenAI tools such as Chat-GPT. At first, experiments were carried out through manually prompting Chat-GPT to generate fictional incident reports, in the context of pharmaceutical industry deviations or in the context of road incidents, as well as extracting the key information. Then the process was automated through API calls to the Chat-GPT 4.1-mini model in order to generate enough reports. In total, 107 “pharmaceutical deviation” and 102 “road incident” synthetic reports have been generated. The reports have been reviewed to check the consistency between the key information and the generated text.

The datasets being simple, they were aggregated in Excel worksheets (each row corresponding to a case, including the following columns: report name, what, when, where, who, how, why, contingency actions and event description text). Figure 2 illustrates an example of a generated case in the context of pharmaceutical deviation reporting.

index	report_name	what	when	where	who	how	why	contingency_actions	event_description	NbChr	Comments
61	Cross-Contamination Incident in Powder Transfer	cross-contamination during powder transfer between batches 8X22 and 8X23	April 27, 2025 at 15:10 (discovery time)	Powder Transfer Station, Manufacturing Line 4	Laura Smith, Operator - reported foreign particles during process; Samuel Green, QA Investigator - launched investigation	inadequate cleaning between batch transfers	human error and procedural non-compliance	production halted, affected batch quarantined, cleaning procedures reviewed, retraining conducted, investigation started (Ref: DEV-2025-1059)	On April 27, 2025 at 15:10, operator Laura Smith observed foreign particles during powder transfer between batches 8X22 and 8X23 at Manufacturing Line 4. The incident was attributed to inadequate cleaning between transfers. QA Investigator Samuel Green was notified and halted production immediately. Affected batches were quarantined and cleaning protocols reviewed. Retraining for operators was scheduled, and a deviation investigation DEV-2025-1059 was initiated.	466	2025-07-19_Automatically generated using API (LLM = gpt-4.1-mini) - Batch 1 (18 reports)

Figure 2 – An example of a “pharmaceutical deviation” synthetic case (red: key information, green: generated text)

B. PHASE 2 – MODEL SELECTION AND OPTIMIZATION OF TEXT GENERATION

The goal of phase 2 is to test several models by giving the cases' key information and generating reports, while varying the prompts and generation parameters. For each case, text similarity metrics are used to compare the generated reports with the report generated in phase 1 (which thereby constitutes the "***ground truth***"). To select the best models and their generation parameters a grid search was carried out.

1. List of tested models

The LLM's investigated are small to medium size. They are open-source models available from the Hugging Face Transformers library. Following is the list of tested models, ordered by number of parameters:

- ***SmolLM2-135M*** = HuggingFaceTB-SmolLM2-135M-Instruct
- ***SmolLM2-360M*** = HuggingFaceTB-SmolLM2-360M-Instruct
- ***Qwen2.5-0.5B*** = Qwen-Qwen2.5-0.5B-Instruct
- ***Llama-3.2-1B*** = meta-llama-Llama-3.2-1B-Instruct
- ***gpt2-xl*** (1.5 billion parameters) = openai-community-gpt2-xl
- ***phi-2*** (2.7 billion parameters) = microsoft-phi-2
- ***Llama-3.2-3B*** = meta-llama-Llama-3.2-3B-Instruct
- ***SmolLM3-3B*** = HuggingFaceTB-SmolLM3-3B

The selection of the medium size model covers a ***range of sizes (from 1B to 3B)***, specialized ***Instruct or non-Instruct*** versions and ***different model architectures*** (single-headed "old" models like GPT2 vs more recent ones like Llama-3.2).

For the small model options, selection was made according to two criteria: ***Instruct*** versions were chosen because they perform better on the reporting task and ***different sizes below 1B parameters*** (ranging from 135M to 500 M).

All the selected models can easily handle reports below a ***few hundred tokens***. The "ground truth" reports are ranging from 250 to 600 characters (est. 80-120 tokens), so they are fit for the task.

2. Grid search parameters

For each tested model, the parameters explored are:

- ***Prompt***: 3 prompting methods (A, B, C and D) have been created (see details in section "Developed tools"). *The 4 methods were explored during the grid search*.

- ***Temperature***: affects the variability and randomness of generated text. Lower values (close to 0) force a more deterministic generation while higher values (1.0 or above) introduce more creativity. *In the grid search, temperature range was set to 0.3 to 1.3*.
- ***Top p***: text generation is an iterative process. At each iteration, the model uses the already generated token sequence to compute the next token probability distribution on the entire vocabulary of the model. The next token is chosen from this distribution. top_p parameter restricts the choice to the tokens having a cumulative probability matching the top_p value. *During the grid search, top_p value was varied between 0.3 and 0.9* (0.3 indicates less tokens to choose from, 0.9 indicates more tokens to choose from)
- ***Top k***: like top_p, top_k is a way to restrict the set of tokens to choose from at each text generation iteration. Its use was found to be redundant with top_p. Hence, *in most of the grid search experiments top_k was held constant at 50*.
- ***Max new tokens***: used to control the length of the generated text sequence. The vast majority of the "ground truth" reports had less than 700 characters, therefore *max_new_tokens value was fixed to 300*.
- ***Sampling***: a Boolean parameter, if False, it forces the generation process to always pick the most likely token at each generation iteration. As the goal is to explore the randomness and diversity of generated text, *do_sample parameters was always set to True*.
- ***Repetition penalty***: a numeric parameter that penalizes repeated tokens in the generated text. In our case, *repetition_penalty was set to 1.0* (i.e. no penalty was applied for repeated token).

3. Text similarity metrics

The similarity between 2 text sequences can be addressed at 2 levels: using ***surface-level*** metrics or using ***semantic-level*** metrics.

a. Surface-level metrics

The comparison is based on evaluating the overlap of n-grams (i.e. groups of n consecutive tokens) between the 2 sequences. 2 scores are often mentioned:

- ***BLEU*** (bilingual evaluation understudy) [[1]] is a precision-oriented score calculated based on the overlapping of 1-gram, 2-grams, 3-grams and 4-grams. BLEU has been used mostly to evaluate the performance of translation tasks. For the reporting application in scope, BLEU score might not be highly relevant but will nevertheless be computed for reference.
- ***ROUGE*** (Recall-Oriented Understudy for Gisting Evaluation) [[2]] is a set of recall-oriented scores that also uses the n-gram overlap. However, they focus on 1-gram, 2-grams and Longest Common Sequence (LCS). ROUGE scores have been used to evaluate summarization tasks as they reflect some of the sentence semantic meaning. Hence, in the scope of the reporting agent, ROUGE scores could be valuable to assess the performance of the models.

b. *Semantic-level metrics*

The semantic-level comparison is based on the embedding vectors which represent the meaning of the text. Depending on the scores, the embeddings considered can either be the embeddings of the texts as a whole or the embeddings of each token in the sequences. Two semantic-level scoring methods have been used in this project:

- ***BERTScore*** [[3]][[4]][[5]] uses the BERT model self-attention mechanism to calculate a scoring reflecting the global alignment between two texts. It compares token-by-token embeddings within the context of the text sequences. BERTScore outputs *3 values: precision, recall and f1*. Figure 3 hereafter shows how the BERTScore recall score is calculated.

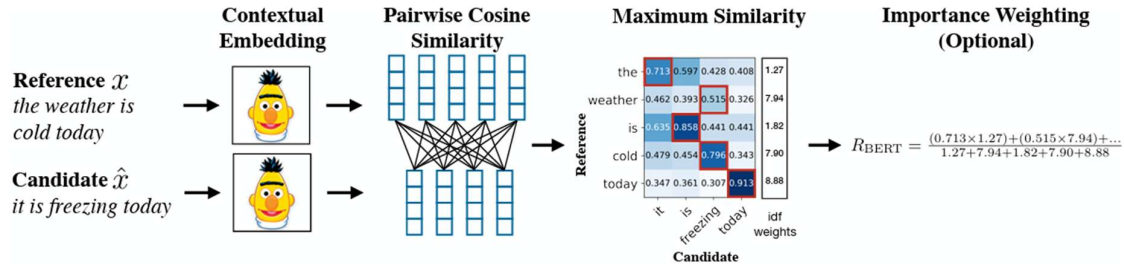


Figure 3 – Illustration of the computation of the recall metric RBERT (from [[4]])

- ***Bi-Encoder and Cross-Encoder Similarity*** [[6]][[7]]. These scores also evaluate the semantic alignment between 2 text sequences and rely on embedding representation of the whole text sequences. Two scores are often reported (Figure 4):
  - ***Bi-Encoder*** converts independently each sequence to an embedding representation then calculates the cosine-similarity (ranges between 0 and 1, values closer to 1 indicate higher alignment).
  - ***Cross-Encoder*** simultaneously passes both sequences to the Transformer model and computes a value indicating higher similarity when it is closer to 1.

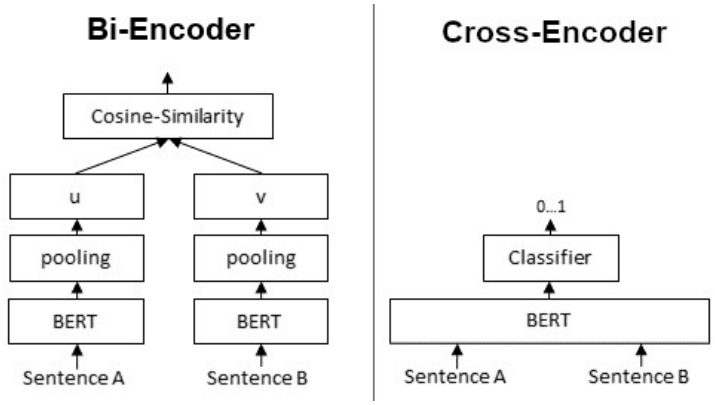


Figure 4 – Comparison of Bi-Encoder and Cross-Encoder Similarity scoring (from [[6]])

4. Developed tools

During phase 2, several objects have been designed in order to automate the testing of models and the optimization of generation parameters. Some of them were also used within the deployed application (see section “ Phase 3 – Deployment”). Hereafter is a list of the developed components and their main functionalities (see Figure 1 for better understanding the architecture):

- ***promptGenerator*** is a Python class aiming to create prompts using 4 methods (A, B,C and D). Method A is the simplest (asking to generate a report based on the key information provided). Method B adds guidelines to ensure the generated text focuses on accuracy and consistency with the key information. Method C adds an example of a specific case in the pharma industry with its key information and its “ground truth” report. Method D has the same instructions and structure as Method C, but the given example is a traffic accident case instead of a pharma industry case.
- ***modelLoader*** is a Python class that manages the import of Transformers models from Hugging Face. Its role is to instantiate the model and the tokenizer.
- ***reportGenerator*** is a Python class that generates a report given a prompt, a model and a tokenizer. Optionally, generation parameters (temperature, top_p…) are passed to the generate_report() method, which allows to get various generated text from the same model and input prompt. Also, this class enforces the use of a specific output structure to ensure consistency and facilitate further processing (in this project, a Pydantic class is used, enforcing a JSON output with 2 fields: title and report)
- ***metricsEvaluator*** is a Python class that computes the metrics described in section “Text similarity metrics” taking as input the generated text (prediction) vs. the “ground truth” text (reference).
- ***testBench*** is a Python class used to automate the grid search on a given Transformers model. It automatically changes the prompt and generation parameters, computes the metrics and saves the generated results for each case in the input dataset.
- ***dataHandler*** is a Python class that manages the input and output data from all other classes.
- ***trainerModel*** is a Python class that manages the training process of a base model. The datasets have been created with chatGPT as ground truth. In this class, three types of training are available based on a SFT (Supervised Fine-Tuning) approach. The three use a different loss function: ***Cross-entropy*** (CE) only, ***Knowledge Distillation*** (KD) which combines CE with Kullback-Liebler (KL) divergence and an ***advanced KD version*** that includes a non-repetition penalty inside the loss function (Unlikelihood Loss – UL).

C. PHASE 3 – DEPLOYMENT

1. Demo Colab notebook

The Reporting Agent functionalities can be experimented using the notebook ***reportingAgent_demo.ipynb***. This demo walks the user through the steps to choose a model and load it, input the event key information and generate a report. The user can also experiment with changing the prompt method and generation parameters.

The demo notebook has been developed using Google Colab therefore it is **recommended to execute it in Google Colab environment**. The steps are:

- Cloning the repository https://github.com/zBotta/reportingAgent (to Google Drive)
- Creating and activating a Python 3.11 environment using requirements.txt
- Executing the notebook reportingAgent_demo.ipynb cell by cell and following the instructions

2. Streamlit application

A simple Streamlit Interface has been developed with Python to show the generated reports in a web application (ReportAgent - a Hugging Face Space by DSTI). It is hosted in Hugging Face Spaces and the deployment is done with a dockerfile. The dockerfile automatically clones the project's Github repository, installs the Python "requirements.txt" file and launches the Streamlit application on port 8501.

The size of the models deployed are as small as possible as the offered free resources in Hugging Face Spaces are limited (16 GB CPU RAM, no GPU). Despite these limitations, the latency and the loading time of the app are acceptable having in mind that it is not meant to be a production application.

The web application allows to select between the following models:

- ***Qwen2.5-0.5B-Instruct***
- ***DSTI/smollm2-accident-reporter-360m-800***: a specialized SmolLM2-360M-Instruct model, result of this project.

After loading the model, it prompts the user to input the 5W1H information and then a report can be generated. If the user is satisfied with the result, the report is downloadable as a pdf file.

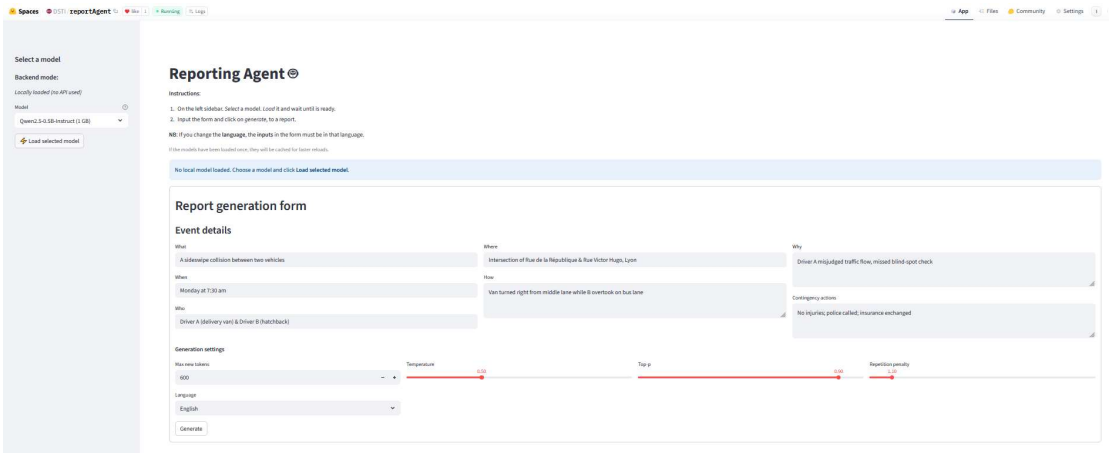


Figure 5 – Snapshot of the Streamlit application hosted in HF Spaces

D. PHASE 4 – MODEL SPECIALIZATION

During Phase 4, a small model size, with an average reporting performance has been specialized by retraining it on a synthetic dataset related to traffic accident reports. The goal is to explore the opportunity of developing a small size model that is specifically targeting the task of the reporting application. The benefit of using a specialized small size model is that it would require less resources in a routine-use context.

III. EXPERIMENTAL RESULTS

A. PHARMA REPORTS

1. Model selection and text generation parameters fine-tuning

A grid search was applied to each of the models mentioned in section “List of tested models”. The input dataset consists of 80 “pharmaceutical deviations” cases among the generated datasets according to section “Phase 1 – Dataset collection”. The prompting and generation parameters were explored as described in section “Grid search parameters”.

a. Text similarity metrics correlation study

All the text similarity metrics were computed for SmolLM3-3B models under all the combinations of 3 prompt methods (A, B, C), 4 levels of temperature and 3 levels of top_p, totaling 2880 generated reports. Correlation among the metrics was evaluated through a Correlation Matrix heatmap and a Principal Components Analysis (PCA).

On Figure 6, the correlation matrix clearly shows that the 3 BERTScore metrics are highly correlated (green rectangle). A high level of correlation is also observed between BERTScore values and Bi-Encoder/Cross-Encoder Similarity scores (purple rectangle), and at a lesser degree, with ROUGE metrics (blue rectangle). However, a much weaker correlation is observed between the BLEU score values vs. all other metrics (yellow and red rectangles). This is particularly true for the precision scores on 2-, 3- and 4-grams (red rectangle)

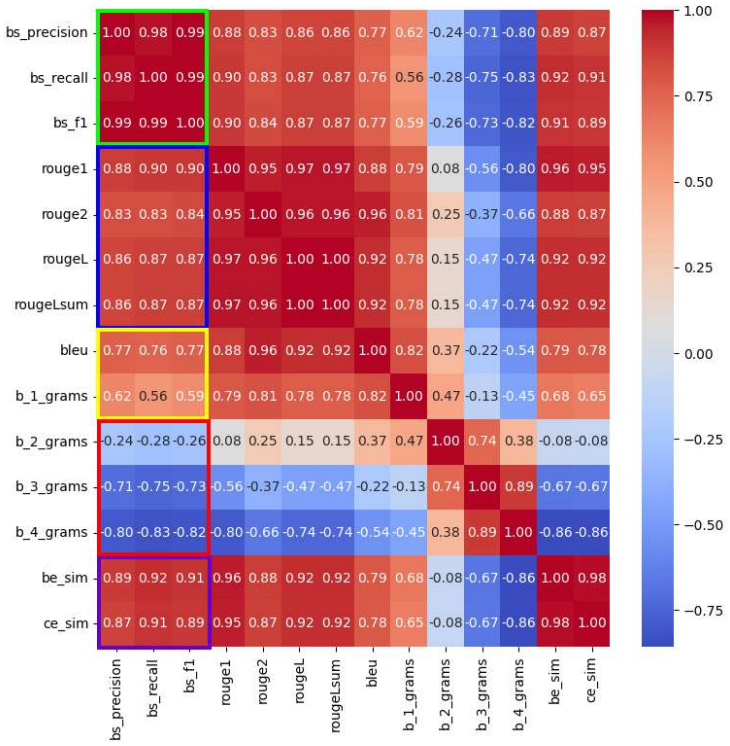


Figure 6 – Correlation heat map of text similarity metrics

The PCA loading plot (Figure 7) confirms the correlation patterns and the clustering of the metrics (same color coding has been used as in Figure 6). This graphical representation of text similarity metrics within the space defined by the 2 first components indicates that the 1st principal component is mostly carrying information on semantic-level similarity, while the 2nd principal describes the surface-level similarity.

Consequently, from this metrics correlation study, a decision has been made to ***concentrate on Cross-Encoder Similarity score when comparing the results of the grid search*** on the models selected for testing (see sections “Grid search results on small size models” and “Grid search results on medium size models”).

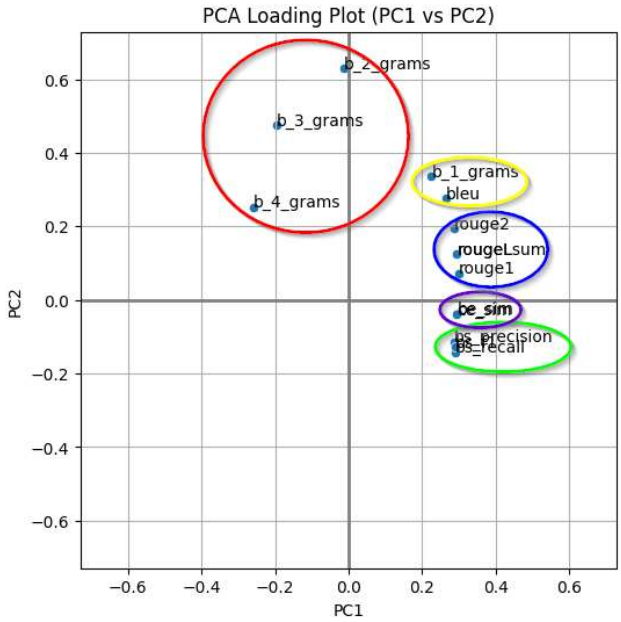


Figure 7 – PCA loading plot of text similarity metrics

b. Grid search results on small size models

Each model's mean Cross-Encoder similarity score has been graphically evaluated against the tested grid search parameters (Figure 8). Generally, the similarity level increases when using a more precise prompt and/or including an example of the desired output. As expected, when models are given more freedom in the generation process (higher temperature and/or higher top_p), the generated text tends to be less similar to the reference text.

Among the small size tested models, ***Qwen2.5-0.5B is the best performing***, applying prompt method C. It shows a very steady performance across the explored temperature and top_p ranges.

The second-best model option is Llama-3.2-1B, applying prompt method C, if temperature is limited to less than 0.7 or top_p to less than 0.6. However, a higher variability in Cross-Encoder Similarity scores is observed, indicating that this model is likely to generate text that would significantly differ from the reference reports' structure and/or semantic meaning.

Table 1 summarizes the optimal generation parameters for these 2 models.

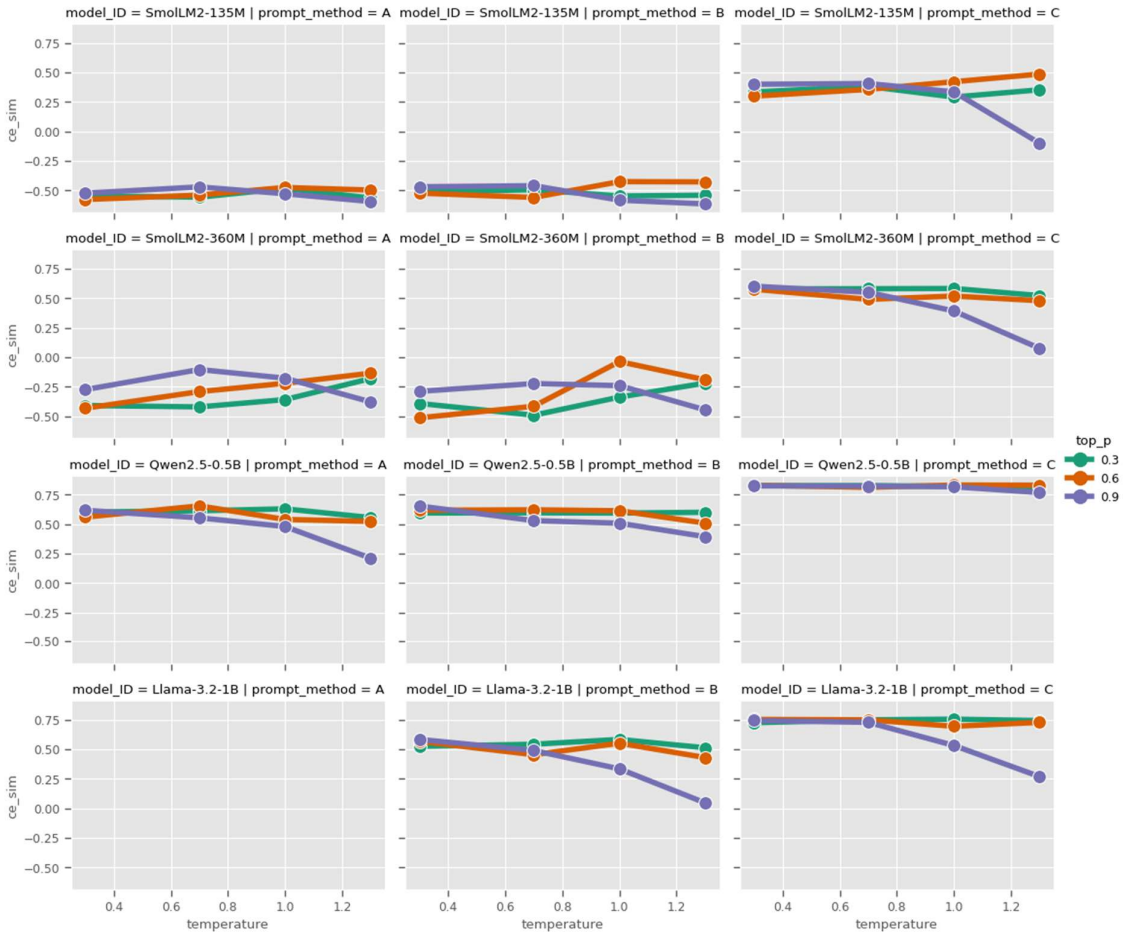


Figure 8 – Grid search results on small size models: Cross-Encoder Score (y-axis) by model_ID (rows) vs. temperature (x-axis), top_p (color) and prompt method (columns)

Model	Optimal generation parameters	Cross-Encoder Similarity
Qwen2.5-0.5B-Instruct	prompt_method = C temperature = 1.0 top_p = 0.6 top_k = 50 k_new_tokens = 300 do_sample = True repetition_penalty = 1.0	Mean = 0.832 St. dev. = 0.078
Llama-3.2-1B-Instruct	prompt_method = C temperature = 1.0 top_p = 0.3 top_k = 50 k_new_tokens = 300 do_sample = True repetition_penalty = 1.0	Mean = 0.755 St. dev. = 0.284

Table 1 – Best small size models with their optimal generation parameters

c. *Grid search results on medium size models*

Comparable trends are observed with medium size models (Figure 9). High temperature combined with high top_p tends to generate text that is less comparable to the reference. Using a more precise prompt improves the similarity score, except surprisingly for SmolLM3-3B models (which yields better results with prompt method B).

However, it is worth highlighting that larger size models are not necessarily guaranteeing better results. gpt2-xl, which has 1.5 billion parameters, gives rather poor similarity scores and phi-2 (2.7 billion parameters) has a comparable performance to Llama-3.2-1B. This suggests that older generation models with large number of parameters can be outperformed by smaller size but more recent models: gpt2-xl was released in Nov-2019, phi-2 in Dec-2023 and Llama-3.2-1B in Sep-2024.

Among the medium size tested models, ***Llama-3.2-3B is the best performing***. With prompt method C, it has a steady performance across the explored temperature and top_p ranges. However, for the purpose of this project, it only shortly outperforms Qwen2.5-0.5B.

SmolLM3 could also be used but restricting it to prompt method B and using top_p not exceeding 0.6.

Table 2 summarizes the optimal generation parameters for these 2 models.

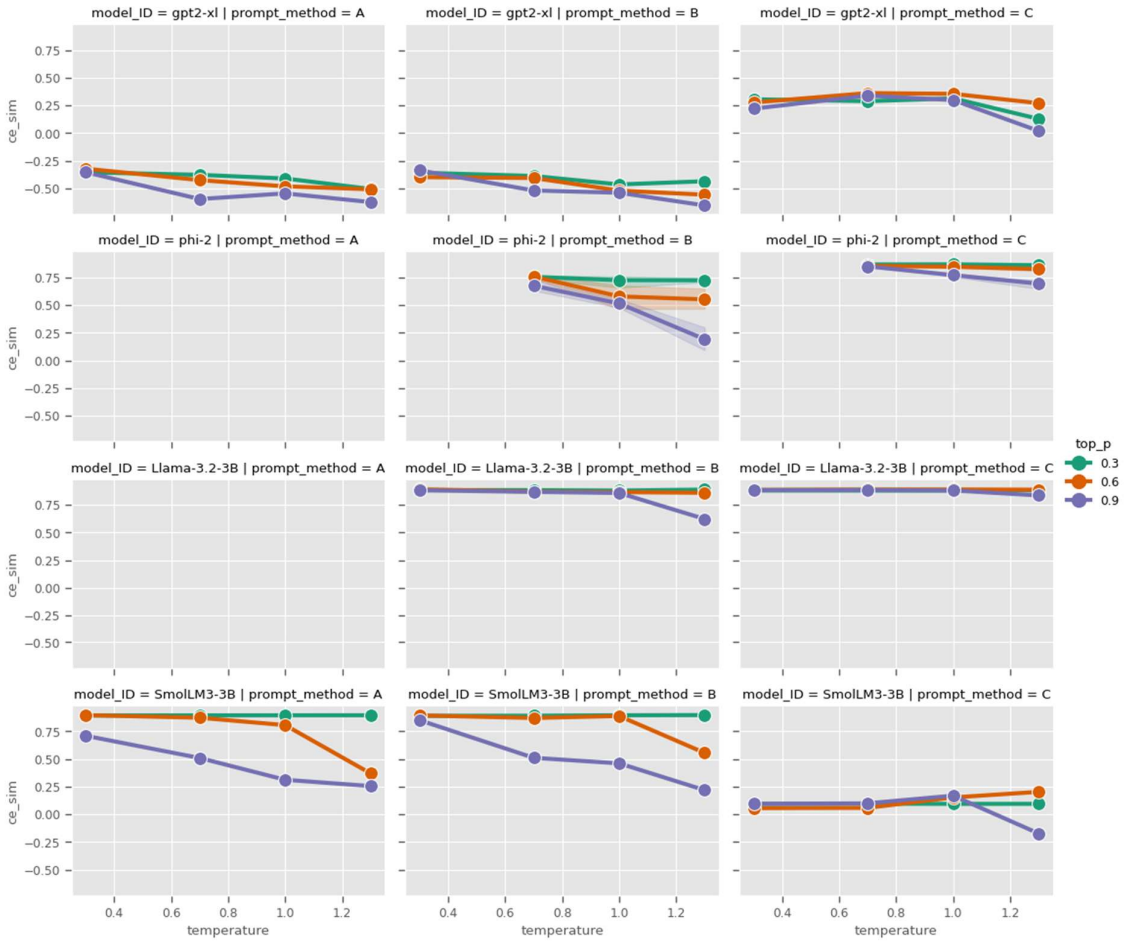


Figure 9 – Grid search results on medium size models: Cross-Encoder Score (y-axis) by model_ID (rows) vs. temperature (x-axis), top_p (color) and prompt method (columns)

Model	Optimal generation parameters	Cross-Encoder Similarity
Llama-3.2-3B-Instruct	prompt_method = C temperature = 1.0 top_p = 0.6 top_k = 50 k_new_tokens = 300 do_sample = True repetition_penalty = 1.0	Mean = 0.889 St. dev. = 0.062
SmolLM3-3B	prompt_method = B temperature = 1.0 top_p = 0.3 top_k = 50 k_new_tokens = 300 do_sample = True repetition_penalty = 1.0	Mean = 0.895 St. dev. = 0.065

Table 2 – Best medium size models with their optimal generation parameters

d. Generation results on test dataset

The remainder of the “pharmaceutical deviation” cases generated as described in section “Phase 1 – Dataset collection” has been used to test the performance of the chosen models on cases that have not been part of the grid search. The optimal generation parameters have been applied as presented in Table 1 and Table 2. The table hereafter summarizes the results obtained on the test set and confirms that:

- Among the 2 small size models, Qwen2.5-0.5B should be preferred. It yields a higher average and lower standard deviation Cross-Encoder Similarity score compared to Llama-3.2-1B.
- For the medium size models, both Llama-3.2-3B and SmolLM3-3B perform well and yield better metrics compared to the small models.

Model	TEST SET Cross-Encoder Similarity
Qwen2.5-0.5B-Instruct	Mean = 0.855 St. dev. = 0.079
Llama-3.2-1B-Instruct	Mean = 0.774 St. dev. = 0.248
Llama-3.2-3B-Instruct	Mean = 0.900 St. dev. = 0.064
SmolLM3-3B	Mean = 0.899 St. dev. = 0.056

Table 3 – Cross-Encoder Similarity score on TEST SET, with optimal generation parameters

B. TRAFFIC ACCIDENT REPORTS

1. Model selection and text generation parameters fine-tuning

The previous study on the pharma industry cases allowed to evaluate the performance of the models by size and for different generation parameter combinations. For the sake of saving computational resources, for traffic accident cases, only the best small and medium size model candidates are evaluated:

- Best small size model***: Qwen2.5-0.5B-Instruct***
- Best medium size model: ***Llama-3.2-3B-Instruct***

A grid search was applied to each model on 80 traffic accident reports that were generated according to section “Phase 1 – Dataset collection”. Note that prompting method A was not considered in the scope of this grid search (method A has consistently been outperformed by methods B and C on the pharma industry cases). A new prompting method D has also been introduced (it is equivalent to prompting method C, just changing the provided example from a pharma industry case to a traffic accident case)

The results of the grid search are graphically presented on Figure 10 and the best generation parameters are reported in Table 4. They are highly comparable to results obtained during the grid search on pharma industry cases (refer to Figure 8, Figure 9, Table 1 and Table 2). Comparing prompt method C vs. D, it can be concluded that changing the example provided in the prompt had a positive influence on the Cross-Encoder Similarity score.

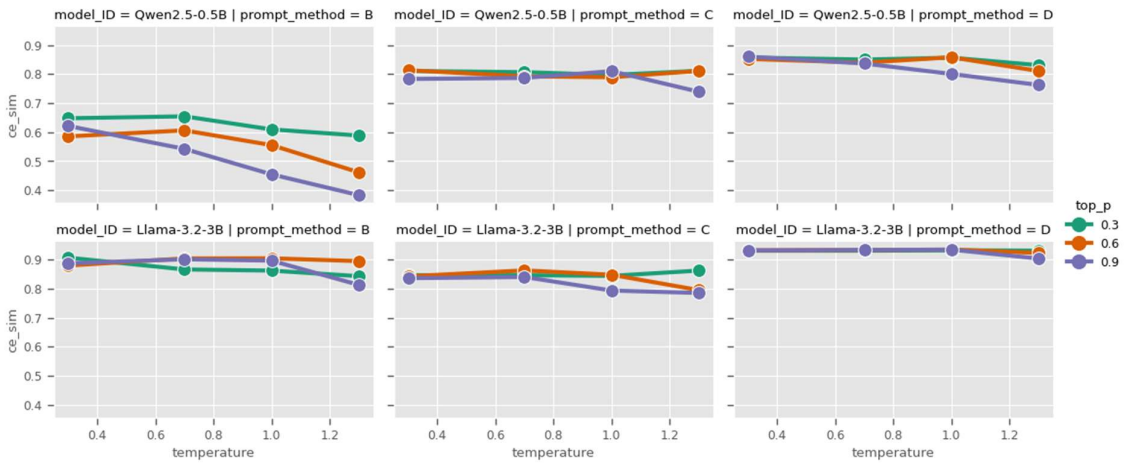


Figure 10 – Generation parameters grid search applied to Qwen2.5-0.5B and Llama-3.2-3B for traffic accident context

Model	Optimal generation parameters	Cross-Encoder Similarity
Qwen2.5-0.5B	prompt_method = D temperature = 1.0 top_p = 0.6 top_k = 50 k_new_tokens = 300 do_sample = True repetition_penalty = 1.0	Mean = 0.858 St. dev. = 0.109
Llama-3.2-3B-Instruct	prompt_method = D temperature = 1.0 top_p = 0.6 top_k = 50 k_new_tokens = 300 do_sample = True repetition_penalty = 1.0	Mean = 0.935 St. dev. = 0.049

Table 4 – Optimal generation parameters with Qwen2.5-0.5B and Llama-3.2-3B for traffic accident context

The results on the grid search in traffic accident context being very similar to the pharma industry context, no confirmatory experiment was performed on a test dataset in the context of traffic accidents. Instead, it was decided to direct the effort and computational resources toward the specialization of a small size model in the context of traffic accidents reporting (detailed in section “Model specialization”).

2. Model specialization: SFT training

We have decided to specialize a model through training with a Supervised Fine-Tuning (SFT) approach. In order to do it we had to generate a large synthetic data set with only traffic accident reports. The SFT training uses the base SFTTrainer class that uses a Cross-entropy formula for calculating the loss.

a. Discussion

For the specialization task, and after analyzing the experimental results on the "pharma industry" cases, we have decided to pick between two small models:

- SmolLM2-360M Instruct
- Qwen2.5-0.5B Instruct.

The constraints and objectives for this specialization task are defined below.

i. Objectives

For the specialization task we have to:

- ***Be able to create a model that maximally adheres to our ground truth report dataset.*** The model must show an improvement, i.e. better similarity scores to the ground truth after training. If the training shows improvement, we can justify a model specialization to a given task (generate reports with a simple structure).
- ***Create the tiniest model that specializes in generating one-paragraph reports*** (like our dataset). The idea is to create a specialized agent with the lowest possible (<1B) size that performs as good as the 3B models.

NB: we have noticed that, in normal circumstances, generating a one-paragraph report could be easily done through prompt engineering. However, since in this study we want to experiment on the training process, we have adapted the training task to adhere to a simple one-paragraph report to be able to evaluate the performance of our training.

ii. Constraints

The constraints of our specialization task are:

- The size of the dataset is 800 cases
- The dataset consists of one-paragraph short reports
- Reports are generated in English (no need for multilingual model)

iii. Comparing model cards

Table 5 presents ***Qwen2.5-0.5B-Instruct*** vs ***SmolLM2-360M-Instruct*** side-by-side model cards and compares their performance on popular small-model benchmarks (source SmolLM2-360M-Instruct model card on Hugging Face [[8]]).

NB: SmolLM2 model card reports both models under ***lighteval***, so it's apples-to-apples.

Benchmark	What it measures	Qwen2.5-0.5B-Instruct	SmolLM2-360M-Instruct
IFEval (avg prompt/inst)	Pure **instruction-following** & constraint adherence	**31.6**	**41.0**
MT-Bench	Multi-turn helpfulness/quality (LLM-as-judge; 0–10)	**4.16**	**3.66**
GSM8K (5-shot)	Grade-school math word-problem accuracy	**26.8**	**7.43**
BBH (3-shot)	Reasoning on “Big-Bench Hard” tasks	**30.7**	**27.3**
HellaSwag	Commonsense completion	**48.0**	**52.1**
ARC (Avg)	Science-QA reasoning (ARC-C/E)	**37.3**	**43.7**
PIQA	Physical commonsense	**67.2**	**70.8**

Table 5 - Qwen2.5-0.5B-Instruct vs SmolLM2-360M-Instruct model cards

In summary, if we are looking for:

- ***Instruction adherence, IFEval*** slightly favors ***SmolLM2-360M***, which can help when we enforce ***single paragraph outputs***
- ***Overall answer quality & reasoning, MT-Bench, GSM8K and BBH*** favor ***Qwen2.5-0.5B-Instruct***, which often yields cleaner prose and better causal phrasing.

Regarding our specialization objectives, the ***IFEval*** is the one that interests us. A high value of this score means that the model will better follow our instructions during training.

b. Model selection for specialization

After analyzing the information presented in the discussion, ***SmolLM2-360M-Instruct*** was chosen for specialization for the following arguments:

- ***It has a tighter format obedience.*** It typically scores higher on instruction-following tests (like IFEval), which maps directly to "***one paragraph***" instructions.
- ***It is lighter and faster.*** ~360M params vs ~500M → smaller VRAM/RAM footprint and quicker ***training/inference, with enough capacity for a short, standard report style.***
- ***It has a bigger improvement gap.*** Qwen2.5-0.5B-Instruct performed well on cross-encoder similarity ***85%***, against the ***77%*** for SmolLM2-360M-Instruct.
- ***It is limited to English*****.** Our data set is only in English, so there is no need for a multilingual model like Qwen2.5-0.5B-Instruct.

c. Creating training data sets

A synthetic data set with 800 traffic accident reports was generated from the OpenAI API. Later, for the sake of improving the training we have created a 5,000 traffic accident report set with the same method.

The data set format is in JSON-lines and parquet. The data set has two columns:

- ***Input***: where we can find the 5W1H information and the contingency action.
- ***Target***: where we find the ground truth or reference report generated with that input.

The 800 rows data set was split as follow:

- ***Train***: 630 rows
- ***Evaluation***: 70 rows
- ***Test***: 100 rows

The 5,000 data set was split as follow:

- ***Train***: 4500rows
- ***Evaluation***: 500 rows
- ***Test***: 100 rows

d. Training process

Regarding the training method we have selected a Supervised Fine-Tuning (SFT) approach available in the Transformer Reinforcement Learning (TRL) library. SFT is the simplest and most used method to adapt a language model to a target dataset. The model is trained in a fully supervised fashion using pairs of input and output sequences. The goal is to minimize the negative log-likelihood (NLL) of the target sequence, conditioning on the input [[9]].

The loss used in SFT is the ***token-level cross-entropy (CE) loss***, defined as:

$$L_{SFT}(\theta) = -\sum log\ p_\theta(y_t \mid y_{<t})$$

where y_t is the target token at timestep t, and the model is trained to predict the next token given the previous ones. In practice, padding tokens are masked out during loss computation [[9]].

i. Optimizing GPU resources

For optimizing the GPU resources and the training runtime, we have decided to use:

- ***Quantization***: the bit size of the model weights and biases is reduced to a 4 bit size with QLoRa (included in the BitsAndBytes module in the transformers library). The only drawback is losing precision on the weights, but the QLoRa method has been proven to give good scores when compared to other bigger models [[10]]. ***This implies a lower VRAM/RAM footprint.***
- ***Parameter-Efficient Fine-Tuning*** (PEFT) [[11]] was introduced by Houlsby et al. (2019). It leaves the weights of the base model unchanged and adds an adapter into each transformer block (see Figure 11). The adapter permits to reduce the trainable parameters and achieve a performance close to a full fine-tuning [[12]]. Integrated in SFT library, PEFT methods, like LoRa, permit to fine-tune large pre-trained models by tuning only the weights and biases affecting a specific task. For instance, if our task is a Causal Language Model (i.e. predict the next token based on the previously generated tokens), the PEFT will focus on optimizing the neurons involved in the Causal LM task ignoring the others. ***This reduces training computational time.***

By using these methods, on a T4 Google Colab virtual machine, the model takes less than 30 minutes to be trained with 13 epochs and be evaluated with 700 cases.

NB: Thanks to PEFT we train only 1.2% of the weights: *trainable params: 4,341,760 || all params: 366,162,880 || trainable%: 1.1857*.

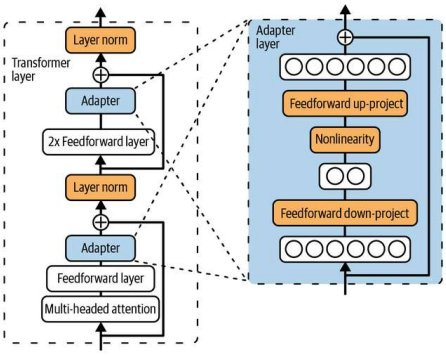


Figure 11 – Houslby et al (2019) first introduced an adapter layer into each transformer layer in BERT model [[12]].

ii. Results on first attempts

For the first training trials, 2 epochs have been used. No improvement in the model performance was observed (see 28Table 6). Thus, the dataset size was increased to 5,000 cases and the training process was fine-tuned.

temperature	top_p	count	mean	std
0.3	0.6	80	0.774	0.247
	0.9	80	0.769	0.305
1	0.6	80	0.762	0.250
1.3	0.3	80	0.760	0.287
0.3	0.3	80	0.744	0.338
0.7	0.3	80	0.742	0.344
1	0.3	80	0.737	0.337
0.7	0.6	80	0.719	0.364
	0.9	80	0.716	0.330
1.3	0.6	80	0.644	0.379
1	0.9	80	0.602	0.439
1.3	0.9	80	0.236	0.590

temperature	top_p	count	mean	std
0.7	0.6	80	0.795	0.254
0.3	0.6	80	0.791	0.244
0.7	0.3	80	0.773	0.275
	0.9	80	0.771	0.223
1.3	0.3	80	0.766	0.283
0.3	0.3	80	0.759	0.285
	0.9	80	0.708	0.404
1	0.3	80	0.706	0.417
	0.6	80	0.677	0.437
1.3	0.6	80	0.635	0.411
1	0.9	80	0.503	0.512
1.3	0.9	80	0.085	0.614

Table 6 – Cross encoder similarity stats on Prompt D: **without specialization** *(left) vs* **first attempt of specialization** *(right).*

iii. Training setup

Below we present the main setup for the SFTTrainer class and the SFTConfig (training arguments).

➢ SFTConfig object

The same training arguments are used for both the 800 and 5,000 targets data sets. The arguments are the following ones:

- ***Number of epochs***. With large number of epochs (30) and an early stopping we would stop when the training diverges and save the best point when this occurs.

- ***Learning rate (2e-4)***. With a cosine function on the learning rate and a warmup ratio of 5%, the training was stable.
- ***Evaluation strategy***. The loss was calculated at each end of epoch.
- ***Batch size and gradient accumulation steps***. These two parameters affect the size of the VRAM used during the training process. We have set a ***batch size*** of 4 (how many samples each GPU processes ***at once*****)** and 16 ***gradient accumulation steps*** (how many micro-batches we run ***before*** doing one optimizer update) aiming to not overflow the VRAM of the Colab T4 machine.
- ***Optimizer***. The optimizer used is a paged Adam 8 bit optimizer that suits the QloRa training approach

➢ LoRa Config object

LoRa is used to make fine-tuning efficient (few trainable params, tiny optimizer state, less VRAM) while still letting the model learn task-specific behavior. For a target weight $W \epsilon R^{dxd}$ LoRA adds a small trainable update $\Delta W = B.A$, where $A \epsilon R^{rxd}$ $B \epsilon R^{dxr}$ ***r≪d.*** During training, W is frozen and ***only A, B are trained***; at inference, W, A and B are merged back $W + \alpha \dot{A}.B$ into the model or keep the adapter as a sidecar.

In summary, what LoRa (fp16, fp32) does is reduce the number of tunable parameters. Combined with Quantization, it produces ***QLoRA***, which ***quantizes the frozen base*** (to 4-bit in our case). QLora combines ***low VRAM*** (quantization) with ***efficient adaptation*** (LoRA).

```
lora_cfg = LoraConfig(
    r=8, lora_alpha=16, lora_dropout=0.05,
    target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj"],
    task_type="CAUSAL_LM"
)
```

- *r (rank)*: is the size of the low-rank update matrices. Bigger sizes would consume more VRAM in order to model more complex changes. For our purpose, as it is a narrow task (creating one paragraph reports), we are taking r=8.
- *lora_alpha*: it is a gain that controls how much new changes in the LoRa adapter will affect the weights. It is a way to control how sensitive training is for affecting weights.
- *lora_dropout:* dropout applied to the adapter input ***during training only*** (no effect at inference). Regularizes the LoRa adapter to reduce overfitting, especially with small datasets.
- *task_type*: the task that we are optimizing. In our case it is **"*CAUSAL_LM*"** as we are training a *decoder-only model* doing next-token generation from a prompt.
- *target_modules:* which part of the LM architecture are going to be modified during training. These targets are the ***high-impact linear layers*** in LLaMA/Mistral/SmolLM-style decoders. Putting LoRA on them gives most steering power per parameter. What each one is doing:

- ***q_proj, k_proj, v_proj, o_proj (attention mechanism)***:
    - Project hidden states into queries/keys/values and back out.
    - Tweaking these changes who attends to whom, how strongly, and how heads mix info.
    - Big lever for instruction following, factual grounding, and long-range coherence.
- ***gate_proj, up_proj, down_proj (in the MLP / SwiGLU block)***
    - *up_proj*: produces the value stream. It learns a rich candidate transformation for each token.
    - *gate_proj*: produces the gated stream and is gated by SiLU (the "Swi" in SwiGLU). It learns where/when to let those features flow (content-dependent control).
    - *down_proj*: brings the inner representation back to model dimension so they can be added to the residual stream.
    - These control ***token-wise transformation and style/format choices (MLP)***.

In practice, pairing ***MLP targets*** (gate_proj, up_proj, down_proj) with the ***attention targets*** (q_proj/k_proj/v_proj/o_proj) yields the best adaptation per parameter for instruction/style tasks.

➢ SFTTrainer object

The SFTTrainer takes the model, the training arguments (SFTConfig object), the training and evaluation datasets, a collator to focus only on the one paragraph report output during the training process and an early stopping callback.

```
trainer = SFTTrainer(
    model=model,
    args=sft_cfg,
    peft_config=lora_cfg,
    train_dataset=ds_tok["train"],
    eval_dataset=ds_tok["eval"],
    data_collator=collator,
    callbacks=[EarlyStoppingCallback(
        early_stopping_patience=2,
        early_stopping_threshold=1e-3
    )],
)
```

We decided to pass a *EarlyStop* callback to the SFTTrainer function with the following parameters:

- *patience* = 2. This permits two consecutives values that do not improve the optimization function during training.
- *parly_stop_threshold* = 1e-3. This allows taking a threshold under which there is no improvement of the optimization function. The value is the difference between the current value in the training minus the previous value of the optimization function (gradient).

For instance, for these parameters, if the gradient is not above the threshold (1e-3) during at least two optimization steps, the training is stopped earlier and the best value is taken. This is done to avoid overfitting. This permits us to train with a big *epoch* value, because the training will stop before starting to overfit.

iv. Results on 800 targets

The result of training ***SmolLM2-360M-Instruct*** model is shown below. The training and evaluation losses show no overfitting. In fact, looking at the table, when the eval_loss started to slightly increase, the training stopped, and the best candidate was kept.

Epoch	Training Loss	Validation Loss
1	1.360200	1.347312
2	1.196200	1.192014
3	1.089900	1.076866
4	0.997900	1.012634
5	0.951800	0.972929
6	0.887900	0.944924
7	0.839300	0.922092
8	0.825800	0.903987
9	0.780800	0.888010
10	0.732200	0.879897
11	0.714900	0.874639
12	0.679600	0.874462
13	0.653600	0.876633

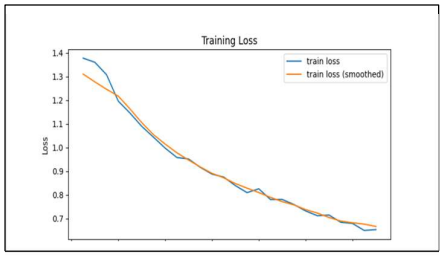


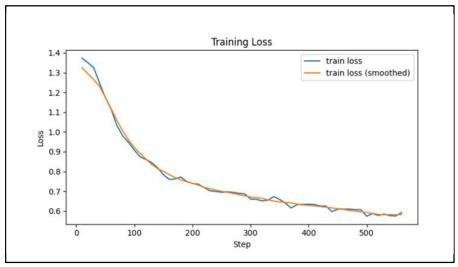
Figure 12 - SmolLM2-360M-Instruct: Training and evaluation losses on the 800-row dataset.

Best evaluation loss is 0.87 at step 120 (perplexity ~ 2.40). ***Final train loss is 0.65*** at step 130. Although we gave 30 epochs the training stopped at epoch 13.

v. Results on 5,000 targets

With 5,000 rows the number of epochs needed to find the optimal is reduced from 13 to 8.

Epoch	Training Loss	Validation Loss
1	1.032200	0.994119
2	0.819000	0.804575
3	0.736300	0.741896
4	0.689500	0.715483
5	0.658600	0.702093
6	0.625500	0.695264
7	0.606500	0.697379
8	0.592200	0.698419



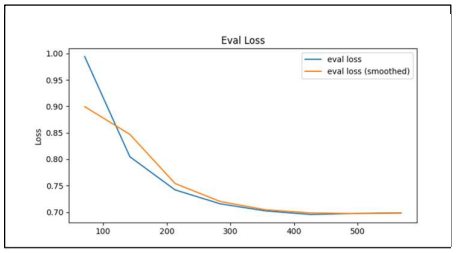


Figure 13. SmolLM2-360M-Instruct: Training and evaluation losses on the 5,000 row dataset.

Best evaluation loss is 0.695 at step 426 (perplexity ~ 2.00). ***Final train loss is 0.592*** at step 560

e. Model performance comparison

We have compared the SmolLM2-360M-Instruct model before and after training. For comparing the effect of specialization, we have done a grid search on the same spectrum as in Phase2.

i. No training on SmolLM2-360M-Instruct

Below, we have the results before training for prompts C (with a pharma example) and D (with a traffic accident example). We observe that the mean and standard deviation does not vary much between the prompt types and the best mean is around 76% on the cross-encoding similarity score.

temperature	top_p	mean	std
0.3	0.9	0.762	0.254
0.7	0.3	0.734	0.335
1	0.6	0.732	0.293
0.3	0.3	0.724	0.353
1	0.3	0.723	0.387
0.7	0.6	0.711	0.354
0.3	0.6	0.689	0.393
0.7	0.9	0.647	0.377
1.3	0.3	0.646	0.431
	0.6	0.588	0.375
1	0.9	0.508	0.493
1.3	0.9	0.083	0.626

temperature	top_p	mean	std
0.3	0.6	0.774	0.247
	0.9	0.769	0.305
1	0.6	0.762	0.250
1.3	0.3	0.760	0.287
0.3	0.3	0.744	0.338
0.7	0.3	0.742	0.344
1	0.3	0.737	0.337
0.7	0.6	0.719	0.364
	0.9	0.716	0.330
1.3	0.6	0.644	0.379
1	0.9	0.602	0.439
1.3	0.9	0.236	0.590

Table 7 – Cross encoder similarity stats on SmolLM2-360M-Instruct **without specialization** *for* **prompt C** *(left) and* **prompt D** *(right).*

ii. 800 cases trained model

Regarding the training on 800 cases, the results show a good adherence to a one-paragraph report. The performances, around ***88%*** on cross encoder similarity score, are comparable to the medium size models we have studied on section *Model selection and text generation parameters fine-tuning*.

Comparing the results of the training (Table 8) against the base model (not trained –Table 6 – Cross encoder similarity stats on Prompt D: ***without specialization*** (left) vs ***first attempt of specialization*** (right). Table 7). We observe:

- an ***improvement*** of ***11%*** on the mean ***cross encoder similarity score*** and a significant reduction of the ***standard deviation***, from ***~0.30*** to ***0.07.***
- a very minor difference between using prompt C (a pharma example is given) and prompt D (traffic accident example is given). Thus, ***the training shows generalization to several types of examples***.

temperature	top_p	mean	std
0.7	0.6	0.8812	0.0713
0.3	0.6	0.8801	0.0770
0.7	0.9	0.8781	0.0738
1	0.6	0.8761	0.0706
0.3	0.3	0.8753	0.0732
1.3	0.3	0.8728	0.0740
0.7	0.3	0.8718	0.0758
1	0.3	0.8677	0.0855
0.3	0.9	0.8575	0.1976
1.3	0.6	0.8458	0.1685
1	0.9	0.7756	0.3621
1.3	0.9	0.5035	0.5604

temperature	top_p	mean	std
1.3	0.3	0.8855	0.0672
0.7	0.3	0.8794	0.0663
	0.6	0.8766	0.0662
0.3	0.6	0.8762	0.0637
	0.3	0.8761	0.0753
1	0.3	0.8759	0.0716
0.7	0.9	0.8686	0.0694
0.3	0.9	0.8664	0.0771
1	0.6	0.8616	0.0944
1.3	0.6	0.8584	0.0850
1	0.9	0.8074	0.3158
1.3	0.9	0.6055	0.4712

Table 8 – Cross encoder similarity stats on SmolLM2-360M-Instruct **with specialization on a 800-row dataset** *for* **prompt C** *(left) and* **prompt D** *(right).*

iii. 5,000 cases trained model

Compared to the 800-row training, enlarging the data set to 5,000 cases ***does not improve performance***. The results are shown below, note that the means of the scores are within the range of the standard deviations when compared to the 800 row training results. This means that no improvement is shown and that, for the worst generation parameters (last rows in Table 9) the standard deviation increases significantly.

Thus, we can deduce that ***the 800-row data set is sufficient for training a small model of less than 1B*** parameters like SmolLM2-360M-Instruct.

temperature	top_p	mean	std
1.3	0.3	0.8855	0.0672
0.7	0.3	0.8794	0.0663
	0.6	0.8766	0.0662
0.3	0.6	0.8762	0.0637
	0.3	0.8761	0.0753
1	0.3	0.8759	0.0716
0.7	0.9	0.8686	0.0694
0.3	0.9	0.8664	0.0771
1	0.6	0.8616	0.0944
1.3	0.6	0.8584	0.0850
1	0.9	0.8074	0.3158
1.3	0.9	0.6055	0.4712

temperature	top_p	mean	std
0.7	0.6	0.8844	0.0765
1	0.3	0.8806	0.0732
0.3	0.3	0.8756	0.1020
1.3	0.3	0.8742	0.1009
0.3	0.6	0.8720	0.0787
0.7	0.3	0.8663	0.1050
0.3	0.9	0.8643	0.0935
1	0.6	0.8597	0.1036
0.7	0.9	0.8229	0.2218
1.3	0.6	0.8033	0.2304
1	0.9	0.6647	0.4533
1.3	0.9	0.4892	0.5337

Table 9 – Cross encoder similarity stats on SmolLM2-360M-Instruct **with specialization on a 5,000-row dataset** *for* **prompt C** *(left) and* **prompt D** *(right).*

f. Model and data set availability

The trained model and the data set produced can be found in the Hugging Face DSTI repo (https://huggingface.co/DSTI). Also, the demo application is available in DSTI's Hugging Face Spaces (https://huggingface.co/spaces/DSTI/reportAgent).

3. KDModel specialization: distillation

a. *Motivation*

The motivations for creating this experiment are:

- Create the tiniest model that could perform as good as the trained SmolLM2-360M-Instruct
- Experiment with a distillation process in LMs
- Comparing the use of KD-CE vs CE-only on the SFT loss function for a small model and for the context of generating one-paragraph reports.

b. *Objectives*

The objective is to create a distilled model from a bigger model of the same family: ***SmolLM2-1.7B-Instruct.***

This bigger model will be distilled into a trained ***SmolLM2-360M-Instruct*** and ***SmolLM2-135M-Instruct*** model. To be able to compare KD-CE (KD logit distillation and CE) and base-SFT (CE only), a CE-only SFT training is carried out on the SmolLM2-360M-Instruct and SmolLM2-135M-Instruct models.

c. *Comparing KD-CE vs CE-only*

i. The distillation process [[13]]

Knowledge distillation is a general-purpose method for training a smaller ***student*** model to mimic the behavior of a slower, larger, but better-performing ***teacher***. Originally introduced in 2006 in the context of ensemble models, it was later popularized in 2015 that generalized the method to deep neural networks and applied it to image classification and automatic speech recognition

By training the student to mimic these probabilities, the goal is to distill some of this “dark knowledge” that the teacher has learned – that is, knowledge that is not available from the labels alone. When that happens, the teacher doesn’t provide much additional information beyond the ground truth labels, so instead we “soften” the probabilities by scaling the logits with a temperature hyperparameter *T* before applying the softmax:

$$p_i(x) = \frac{exp(z_i(x)/T)}{\sum_j exp(z_i(x)/T)}$$

Since the student also produces softened probabilities of its own $q_i(x)$, we can use the Kullback-Leibler (KL) divergence to measure the difference between the two probability distributions:

$$D_{KL}(p, q) = \sum_i p_i(x) log \frac{p_i(x)}{q_i(x)}$$

With the KL divergence we can calculate how much is lost when we approximate the probability distribution of the teacher with the student. This allows us to define a ***knowledge distillation loss***:

$$L_{KD} = T^2 D_{KL}$$

Where T^2 is a normalization factor to account for the fact that the magnitude of the gradients produced by soft labels scales as $\frac{1}{T^2}$. For classification tasks, the student loss is then a weighted average of the distillation loss with the usual cross-entropy loss of the ground truth labels:

$$L_{\text{student}} = \alpha L_{CE} + (1 - \alpha) L_{KD}$$

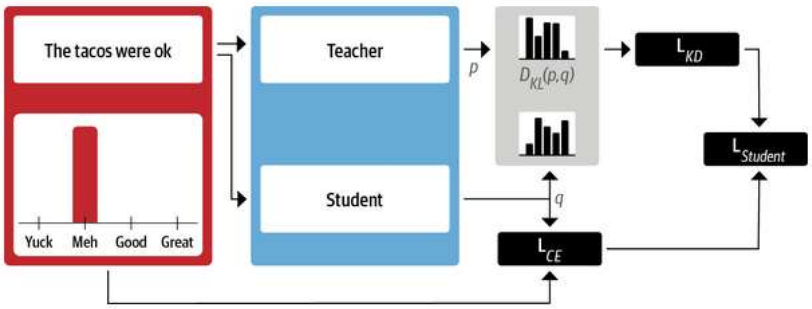


Figure 14 – A workflow of the distillation process. A hybrid loss function containing CE (cross-entropy) and KL divergence to learn from the teacher.

The loss expressed as L_{student}, is what we have called ***KD-CE (Knowledge Distillation and Cross Entropy)***, i.e. the distillation process.

ii. Results on CE-only loss function

- 800 row data set

We have taken the 800 rows dataset (*zBotta/traffic-accidents-reports-800*) and trained a ***SmolLM2-135M-Instruct*** base model. We can observe that, even if the training curves look healthy, the cross-encoder similarity score is poor (around ***27%*** for the best generation parameters).

With 30 epochs given, after epoch 17, the training is stopped with ***best evaluation loss: 1.017*** at step 150 (perplexity ~ 2.76) and as ***final train loss: 0.77*** at step 170.

Epoch	Training Loss	Validation Loss
1	1.700600	1.668881
2	1.505800	1.473667
3	1.371300	1.340912
4	1.248400	1.256381
5	1.183000	1.193175
6	1.106400	1.144049
7	1.045400	1.106472
8	1.031200	1.083902
9	0.980200	1.061038
10	0.935300	1.046512
11	0.917900	1.036460
12	0.888000	1.028848
13	0.860800	1.020548
14	0.835900	1.018986
15	0.811800	1.016806
16	0.787400	1.018156
17	0.771900	1.018733

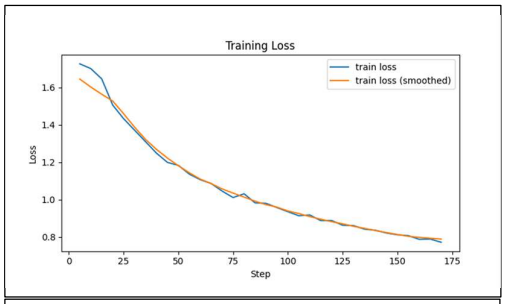


Figure 15 - SmolLM-135M-Instruct: Training and evaluation losses on the 800-row dataset.

temperature	top_p	mean	std
1.3	**0.3**	0.273	0.387
0.3	**0.9**	0.272	0.403
1	**0.6**	0.254	0.464
0.7	**0.9**	0.240	0.464
0.3	**0.3**	0.204	0.397
0.7	**0.3**	0.193	0.379
0.3	**0.6**	0.189	0.428
0.7	**0.6**	0.189	0.440
1	**0.3**	0.172	0.402
1.3	**0.6**	0.165	0.466
1	**0.9**	0.111	0.505
1.3	**0.9**	-0.057	0.520

Table 10 – Cross-encoder sim. scores on SmolLM2-135M-Instruct after training 800 rows (prompt D).

➢ 5k row data set

We have taken the 5k rows dataset (*zBotta/traffic-accidents-reports-5k*) and trained a SmolLM2-135M-Instruct base model. No improvement was observed compared to the 800 rows (results are not repoted as they are redundant).

iii. Results on KD-CE loss function

For the distillation, several ideas have been tested, without achieving better results than the CE only SFT training:

- Giving more influence to D_{KL} term (lower α) or less influence (higher α) in the loss function
- Create a more complex loss function that considers an unlikelihood loss (avoid repetition of tokens and n-grams)
- Play with the size of the data set (800 or 5,000 rows)
- Shorten the length of the prompt to see if it has an impact on the logit adherence of the student (higher supervised fraction on the output)
- Playing with the training parameters (learning rate, epochs, optimizers, …)
- Smoothing the effect of the D_{KL} term, playing with:
    - the KD temperature (smoothing the logits probability curve)
    - the top K tokens (filtering the top tokens considered in the logits probability curve)
    - Adding a warm-up of the influence of the D_{KL} term
    - Forcing the first epochs to use a CE-only loss function

Despite all these attempts, the performance was not improved or decreased. Results are not reported in a detailed manner, but a summary of the distillation experiments is provided below, on both ***SmolLM2-360M-Instruct*** and ***SmolLM2-135M-Instruct*** models.

➢ SmolLM2-135M-Instruct

- The ***distillation did not improve the performance***.
- The ***SmolLM2-135M-Instruct*** model still shows low adherence when generating one-paragraph reports, probably because the model size is not enough for completing the task properly. ***The student can't match the teacher's calibration***.

➢ SmolLM2-360M-Instruct

- The ***distillation did not improve the performance***.
- We only obtained a similar but not better performance than using CE-only (see Table 11). Though, we observe that ***increasing the KD influence reduces the standard deviation on the distillation process*** (Table 11).

temperature	top_p	mean	std
1.3	0.3	0.850	0.103
0.3	0.6	0.846	0.115
	0.9	0.839	0.112
1	0.3	0.838	0.109
0.7	0.6	0.834	0.118
0.3	0.3	0.831	0.117
0.7	0.3	0.821	0.211
	0.9	0.818	0.194
1	0.6	0.817	0.124
1.3	0.6	0.817	0.123
1	0.9	0.664	0.416
1.3	0.9	0.250	0.609

temperature	top_p	mean	std
0.7	0.6	0.865	0.079
1	0.6	0.853	0.091
0.3	0.3	0.851	0.093
	0.6	0.848	0.099
	0.9	0.847	0.096
0.7	0.3	0.846	0.099
1.3	0.3	0.837	0.106
0.7	0.9	0.836	0.103
1	0.3	0.834	0.113
1.3	0.6	0.818	0.193
1	0.9	0.791	0.185
1.3	0.9	0.551	0.520

Table 11 - SmolLM2-360M-Instruct. **Distillation results** *on a lower KD influence (left; α =[0.75 to 0.9]) vs a higher KD influence (right;α =[0.3 to 0.6])*

IV. CONCLUSION

A. BEST MODEL ON PHARMA

- In order to select a model and its generation parameters, several text similarity metrics described in literature have been assessed. It was found that many of them tend to be highly correlated and therefore focusing on one should be sufficient. In our case, cross-encoder similarity score was found to be well-suited.
- To create a well-structured report based on inputting key information related to a deviation in the field of pharmaceutical industry, the best model among small size group was found to be ***Qwen-2.5-0.5B***. It achieved a cross-encoder similarity score in the range of 0.83.
- Larger models were tested, and the best performing was found to be ***Llama-3.2-3B-Instruct***. Its cross-encoder similarity score was in the range of 0.90.

B. BEST MODEL ON TRAFFIC ACCIDENTS

- The best model according to the cross-encoder similarity score is ***Llama-3.2-3B-Instruct*** with a value of ***0.93.***
- The specialized (trained) ***SmolLM2-360M-Instruct*** model gave very good cross-encoder similarity scores with a mean value of 0.83 on the proposed grid search ranges. The best generation parameters gave a mean cross encoder similarity score of ***0.88.***
- The ***specialized SmolLM2-360M-Instruct*** did not outperform the bigger models Llama-3.2-3B-Instruct (0.93), SmolLM3-3B or Qwen2.5-0.5B-Instruct (0.88) but considering its compact size, this specialized model has ***a better performance-to-size ratio***. For instance, compared to Llama3B, the size is reduced by up to 10 times while the performance is only dropping by 5%.
- ***Qwen2.5-0.5B-Instruct*** is performing well and it is one of the lightest models. It could be a good candidate if ***multilingual report generation*** is needed.
- ***SmolLM2-360M-Instruct***, and ***Qwen2.5-0.5B-Instruct*** showed ***robust behavior for prompts with different type of examples*** (prompt C and D)

C. TRAINING PROCESS

- The best smallest model is ***SmolLM2-360M-Instruct*** with a cross-encoder similarity score of ***0.88***, higher than ***Qwen2.5-0.5B-Instruct*** base model and close to 3B models.
- Adding more rows to the dataset (from 800 to 5,000) did not improve the cross-encoder similarity score on the SmolLM2-360M-Instruct model. This leads to conclude that 800-row dataset was enough for specializing the model.
- The ***SmolLM2-135M-Instruct*** did not show good adherence to one paragraph report generation nor with 800 rows neither with 5,000 rows, giving poor cross-encoder similarity

scores around 10%. We can deduce that, for this model, ***CE-only SFT training does not permit the model to adhere to the task*** independently of the size of the data set.

D. DISTILLATION

In general, Knowledge Distillation (KD) did not work when transferring knowledge from a ***SmolLM2-1.7B-Instruct*** to smaller models of the same family (135M and 360M) because:

- The size of the data set is large enough to use a CE-only strategy as the results on the 360M model have shown. The use of ***KD does not add value*** as it is interesting if ***we had a poor data set*** and we wanted to generate extra data with our teacher.
- The task of ***generating one-paragraph reports is a low-entropy task***, meaning that is too ***deterministic***, therefore the logit probabilities of the teacher will be too sharp. KL adds regularization that conflicts with the gold/target style (CE term). Thus, ***the use of KD here is counterproductive, it could be interesting if the task had higher entropy or was more "creative";*** where the teacher could transfer some of its "creativity knowledge" to the student through the logit curves.

E. BEST MODEL

Based on this experimentation, choosing the best model would depend on the application. If the application targets:

- ***Performance***: The highest performance possible needs to be delivered, without training. In that case, we would recommend using ***Llama-3.2-3B-Instruct*** base model.
- ***Resource efficiency***: Execution with low GPU resources, or even on CPU without considering time of inference, should be favored. Then, we would choose the specialized ***SmolLM2-360M-Instruct*** model (***DSTI/smollm2-accident-reporter-360m-800***).

V. REFERENCES

[1] https://en.wikipedia.org/wiki/BLEU

[2] https://en.wikipedia.org/wiki/ROUGE_(metric)

[3] https://github.com/Tiiiger/bert_score

[4] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, Yoav Artzi. BERTSCORE: EVALUATING TEXT GENERATION WITH BERT. In *International Conference on Learning Representations*, 2020. [https://arxiv.org/pdf/1904.09675.pdf]

[5] https://haticeozbolat17.medium.com/text-summarization-how-to-calculate-bertscore-771a51022964

[6] https://sbert.net/examples/cross_encoder/applications/README.html#cross-encoder-vs-bi-encoder

[7] Nils Reimers, Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP*, 2019. [https://arxiv.org/abs/1908.10084]

[8] SmolLM2-360M card in Hugging Face [https://huggingface.co/HuggingFaceTB/SmolLM2-360M-Instruct]

[9] SFT Hugging Face. https://huggingface.co/docs/trl/sft_trainer

[10] QLora: Efficient Finetuning of Quantized LLMs [https://arxiv.org/abs/2305.14314]

[11] PEFT Hugging Face. https://huggingface.co/docs/peft/index

[12] AI Engineering – Chip Huyen. Ch. 7 – Finetuning [https://learning.oreilly.com/library/view/ai-engineering/9781098166298/]

[13] Natural language Processing with Transformers, *ch. 8: Making Transformers efficient for Production* – Lewis Tunstall et. al *[https://learning.oreilly.com/library/view/natural-language-processing/9781098136789]*