Making an LLM an NLP expert via fine tuning

Aashrith Madasu asm6590@psu.edu
Darshan Chudiwal, dsc5636@psu.edu,
Yash Priyadarshi yvp5218@psu.edu

1. Problem Definition

Large Language Models like GPT-2, GPT-3 and their open source variant have demonstrated have showcased great capabilities in generating coherent text, language translation and code drafting. Although these models possess a variety of skills and knowledge across different domains, they are do not specialize in a single focus area. When these models are presented with a novel research challenge, they lack the expertise and in-depth knowledge required to outline the thorough, step-by-step research methodologies. Whether it is academic or industrial NLP arena, this gap means that even with access to the most powerful generative tools, researchers still have to rely on their own experience, literature surveys and manual synthesis for moving ahead.

This project addresses this gap by fine-tuning an open source casual language model so that it becomes and expert advisor for NLP research problems. Our fine-tuned model will be trained on a carefully selected research focused dataset in a specific NLP subdomain (**LLM Prompt Optimisation Engineering**). Our main goal is to take a general purpose LLM and turn it to a focussed assistant which can understand research questions, recall relevant methods and give a clear step by step plan for solving them.

Our two stage workflow consists of data preparation and model fine tuning. For data preparation, we are selecting and preprocessing a task specific corpus (above highlighted tag) which expresses a mapping of problem and method in already published research articles. The individual training examples pairs an apt statement of a problem and how to approach it (using Llama). With respect to fine tuning, we are picking up google/gemma-3-1b-it, meta-llama/Llama-3.2-1B and adapting the model's parameters using supervised causal language modelling which will make the model able to specify the descriptive approach which follow a problem statement.

This work lays the foundation for future tools that combine on demand retrieval of external research, automatic citation integration, and interactive refinement making it faster and easier to develop and iterate on new NLP methods.

2. Dataset Overview

For our chosen task, we could not find a readily available dataset, hence we decided to generate the dataset ourselves. The dataset focusses exclusively on "**LLM Prompt Optimisation Engineering**" papers and its construction takes place in two phases. Let us talk more on each phase.

Starting with phase one, here we are querying the ArXiv API for retrieving papers with the tag highlighted above. We are limiting the results to a maximum of 100,000 fetching each paper's metadata. The query gives us a total of 3,200 results. Each row entry has the following fields: {tag, title, authors, published, summary, link, pdf_link, id}. We are treating the abstract of the paper as the summary, and rest of the things have been taken as it is from the API. This is then stored as a CSV file.

In phase two, we are reading the CSV file created in phase one. We are then fetching the abstract of each paper and constructing a prompt. The structured prompt includes an example which shows how to extract problem and approach from a sample abstract (**few shot learning**). Using Meta's Llama-3.2-3B Instruct, via Hugging Face we are reading the prompt and predicting the issue addressed by a paper (problem) and how to resolve it (approach), all being done based on the abstract. Now because of prompt leakage we are getting extra data. We are filtering out only the problem and approach and adding a them as new columns in the original dataset and creating a new csv file.

Collectively the final dataset has the following columns; {paper_id, tag, title, authors, published, summary, link, pdf_link, id, problem_solution, problem, approach}. This dataset provides a base for finetuning the LLM.

2.1. Column Description

Let us also have a discussion on the columns present in the dataset. The first column is an unnamed column which is used for index identification, next we have paper_id which is the integer index which got sequentially assigned during data collection. Next, we have the tag followed by the full title of the paper. Next is the comma separated list of authors. Next is the publication date followed by the summary of the paper

(abstract). Next, we have the URL to the paper's ArXiv abstract page followed by the direct PDF download URL. Next is the native ArXiv id. Next is problem_solution which is the JSON returned by Llama-3.2-3B Instruct plus some gibberish text. The last 2 columns are problem and approach which have been filtered out from problem_solution column.

2.2. Dataset Split

The dataset has been split into training and validation sets using a 90-10 ratio, where 90% of the data goes for training the model and the remaining 10% is reserved for validation. The data contains 3,200 rows, with 2,880 rows allocated for training and 320 rows for validation. This ensures that the model is exposed to a great number of examples during training before it is evaluated for its performance.

2.3. Data Preprocessing

We began by splitting the CSV file from phase 2 into train.csv and valiation.csv in 90-10 ratio. We then prepended "Question: " to the problem field and "\nAnswer: " to the approach field and created a new raw string. Consider the example below for better visualisation –

> Question: How can we adapt prompt tuning for a low-resource domain?
>
> Answer: We first gather in-domain unlabeled data, then…

Next, we are calling in the model's tokenizer on this combined text while specifying truncation as True, and the max_length as 512, to make sure that no sequence exceeds 512 tokens. We are then creating the training labels by copying the resulting input_ids (for the model to learn to predict every token, prompt and answer) and drop the original CSV columns (Unnamed: 0, problem, approach) to leave only the model inputs (input_ids, attention_mask) and labels.

Finally during training, we are wrapping each split in a PyTorch DataLoader which uses a custom collator to batch those tensors to uniform inputs, which makes things ready for our casual language model.

2.4. Dataset Challenges

Let us also discuss some of the challenges we faced while preparing the dataset. Since we could not find a readily available dataset for our task and we are limited to focussing on a single area of NLP, we could only come up with a limited number of examples for our dataset. The model occasionally emits malformed JSON or incomplete fields. For resolving this, we stored the raw JSON, then processed the column to filter out just the problem and approach sections of the Llama-3.2-3B Instruct output. Our initial flow was to parse the PDFs directly, however it was very difficult to parse through, so we shifted to making use of the ArXiv API to fetch the data which is there in dataset.csv (CSV file from phase one).

3. Approach

For converting a general purpose Large Language Model into an NLP research advisor, we are fine tuning it on pairs of research problem prompt and the approach generated by Llama-3.2-3B Instruct.

3.1. Model Architecture

Let us take a look at the architecture of the models we considered as candidates for this work, which are gemma-3-1b-it and Llama-3.2-1B.

The google/gemma-3-1b-it is an instruction variant in the gemma family. It's architecture type is a dense decoder-only model. The model has 1 billion parameters and it is instruction tuned (meaning the model is fine-tuned to follow user-instructions and perform tasks with natural language instructions). It's input modality is text only with a context window of 32,000 tokens for both the input and the output.

The meta-llama/Llama-3.2-1B is a new release in the llama family of models. It's architecture type is an auto-regressive decoder model. The model has 1.23 billion parameters in total. It reduces memory usage and improves inference speed by grouping query heads. The model has a context window of a staggering 128,000 tokens. It supports 8+ languages as well.

3.2. Training and Evaluation

We are fine-tuning the model over 3 epochs and explored 3 approaches in fine-tuning the models which are SFT, LoRA, and QLoRA. The hyperparameters for fine-tuning the model via SFT are as follows:

Learning rate (INITIAL_LR)	5e-5
Batch size (BATCH_SIZE)	8
Epochs (EPOCHS)	3
LR scheduler	Linear decay from 5e-5 to 0 over len(dataloader)×EPOCHS steps, with 0 warmup steps

Next, we fine-tuned the model using LoRA via PEFT which involves injecting smaller “adapter” matrices into attention layers; and only these LoRA weights are trained, leaving the bulk of the model frozen. Here we have used the following hyperparameters:

Rank (r)	8
Alpha (lora_alpha)	32
Dropout (lora_dropout)	0.1
Target modules	attention projection layers (c_attn, q_proj, v_proj)
Bias	none

Finally, we fine-tuned the model using QLoRA which combines LoRA with 4-bit quantization of the base model to store weights in 4 bits and then train only the LoRA adapters in full precision. Here we have the following hyperparameters:

4-bit type	NF4
Double quantization	enabled
Compute dtype	float16
Rank (r)	8
Alpha (lora_alpha)	32
Dropout (lora_dropout)	0.1

We used AdamW for all the trainable parameters. Also for model evaluation we are using Bleu Score, Rouge-L, and Meteor. For fine-tuning, we used Nvidia RTX A6000 - 50GB and loaded the model in it.

```
|   0  NVIDIA RTX A6000               On  |   00000000:1B:00.0 Off |                  Off |
| 66%   85C    P2             296W /  300W |   26293MiB /  49140MiB |     100%      Default |
|                                          |                        |                  N/A |
```

Figure 1: Hardware specifications for this work

4. Experiments and Analysis

As discussed in section 3.2, we explored 3 approaches in fine-tuning the model which are SFT, LoRA, and QLoRA, and found SFT to be the most time consuming followed by LoRA and QLoRA.

Model	Per epoch time
QLORA	2:47 minutes for each epoch
LORA	6:37 minutes for each epoch
SFT	9:42 minutes for each epoch

This was expected as SFT involves fine-tuning all the parameters. Further, the SFT tuned LlaMA model performs the best (as discussed later in section 5). Thus, we perform the next few experiments with the fine-tuned LlaMA model.

4.1. Inference Time Ablations

At inference time, there are multiple parameters provided by the hugging face library where we can allow an LLM to be more expressive. Decoding hyperparameters play an important role in controlling the trade-off between response quality, diversity, and latency. We are mainly focusing on three widely used settings in the Hugging Face Transformers library which are top-k sampling, nucleus (top-p) sampling, and beam search.

By systematically varying each parameter, we are aiming to quantify how expressive the model can be under different decoding constraints, and how this expressiveness affects both automatic metrics (e.g., BLEU, ROUGE) and human judgments (e.g., coherence, creativity).

4.1.1. Top-k Sampling

In top-k sampling we restrict the model’s next-token distribution to the k tokens with the highest probabilities, renormalize, and sample from this truncated set.

> *Hypothesis:* Lower k (e.g., 5 or 10) yields more focused—but potentially repetitive—outputs; higher k (e.g., 50 or 100) increases diversity at the risk of incoherence.
>
> *Settings:* k ∈ {5, 10, 50, 100}

Figure 2: Evaluation metrics vs top_k

A small k (5) gives the highest BLEU and METEOR, but ROUGE-L peaks at k=50. Again, variations are modest (~0.003–0.005), so tuning k yields only incremental improvements.

4.1.2. Nucleus (Top-p) Sampling

Nucleus sampling selects the smallest set of tokens whose cumulative probability exceeds p, then samples from that set. This adaptive truncation often balances diversity and fluency more gracefully than fixed-k.

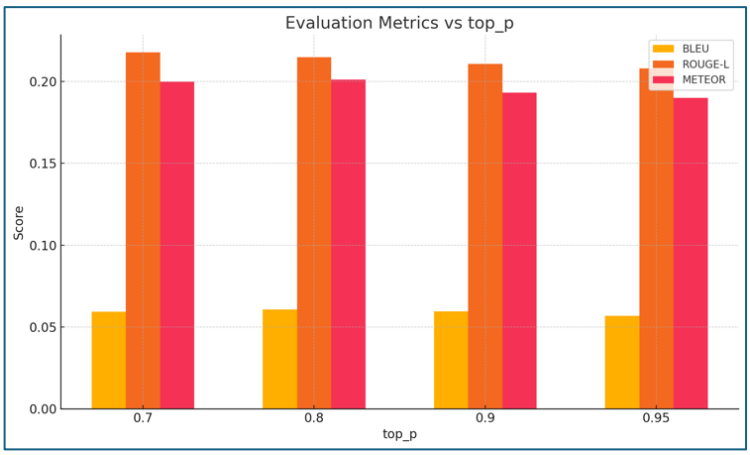

Figure 3: Evaluation metrics vs top_p

Hypothesis: Smaller p (e.g., 0.7) produces safe, conservative text; larger p (e.g., 0.95) allows for more creative but potentially off-topic continuations.

Settings: p ∈ {0.7, 0.8, 0.9, 0.95}

Nucleus sampling shows the largest lift overall—BLEU jumps into the 0.06 range, ROUGE-L up to 0.218, and METEOR to 0.201. The best results are observed around p=0.7–0.8; beyond that (p≥0.9) the scores steadily decline.

4.1.3. Beam Search

Beam search explores the top b candidate sequences at each decoding step, retaining the most likely b partial sequences until termination. This deterministic strategy tends to optimize likelihood but can produce generic outputs.

Hypothesis: Small beam widths (b=1 or 2, the latter being equivalent to greedy + one alternative) are fast but may miss higher-scoring sequences; larger beams (b=5 or b=10) improve quality at the cost of speed and repetition.

Settings: beam_size ∈ {1, 2, 5, 10}

Using more beams yields incremental gains in ROUGE-L and BLEU, but the improvements are quite small, and METEOR actually dips after 2 beams.

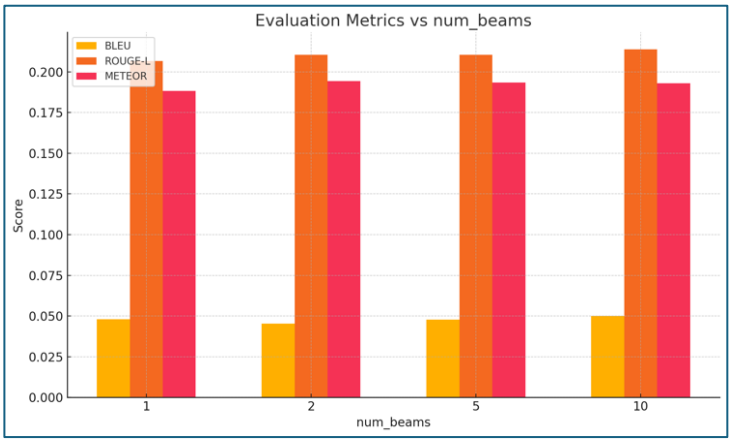

Figure 4: Evaluation metrics vs num_beams

5. Results

First, we take a look at how the models look during the training. Then, we take a look at the metrics that will be used to benchmark the models.

5.1. Training Loss

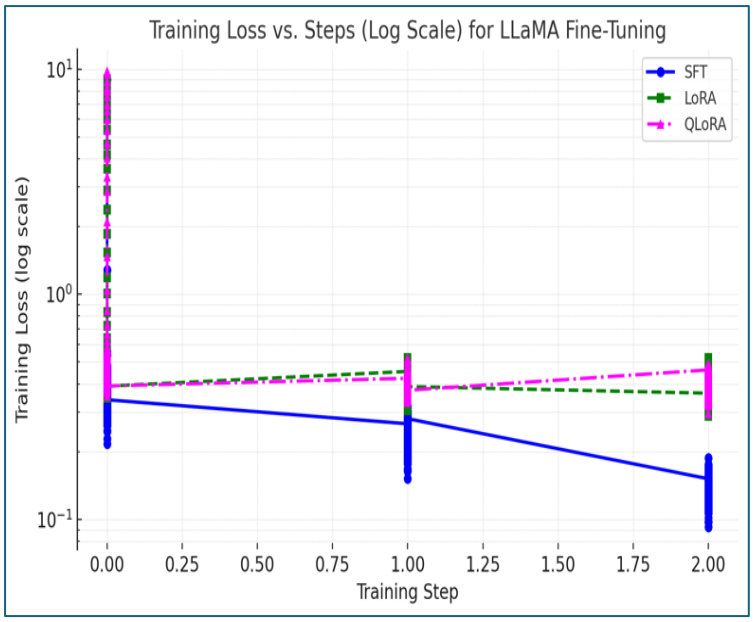

Figure 5: Training Loss vs Steps for LLaMA with SFT, LoRA and QLoRA

All three curves start with a high loss value at "step 0", this is really just the un-fine-tuned LLaMA model's cross-entropy on your training batch.

SFT (blue) plunges the fastest: from ~0.54 at step 0 down to ~0.26 at step 1, then to ~0.15 at step 2.

LoRA (green dashed) starts lower than QLoRA at step 0 (~0.43), then falls to ~0.35 at step 1 and ~0.32 at step 2. Because LoRA only learns low–rank "injection" matrices rather than the full parameter set, it converges more gradually than SFT.

QLoRA (magenta dash-dot) actually begins highest of the three (~0.49), dips slightly to ~0.45 at step 1, then creeps up a bit to ~0.52 by step 2. QLoRA's extra quantization overhead can make the initial optimization trickier; it often takes more steps (or a smaller learning rate) to beat the other methods.

5.2. BLEU Score

Bilingual Evaluation Understudy or BLEU is an evaluation metric used for checking the quality of a machine generated text. It is most commonly used for machine translation by comparison of generated text against one or more reference text written by humans. It focuses on the number of n-grams in the candidate/generated text which match with the reference text, making it a precision oriented metric.

How this metric exactly works is, it calculates the proportion of n-grams in the generated text which is also present in the reference text. Usually, BLEU combines results for 1-gram up to 4-gram matches, and to prevent the model from generating very small outputs, BLEU applies brevity penalties if the candidate text is smaller than the reference text. Mathematically speaking, BLEU is represented as shown below –

$$BLEU = BP \times exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

Here, p_n is the precision for n-grams or simply speaking, it is the proportion of the candidate bigrams which appear in the reference. w_n is the weight of each n-gram order, and BP is the brevity penalty. BP is calculated as:

$$BP = \begin{cases} 1, & c > r \\ e^{1-r/c}, & c \leq r \end{cases}$$

where c is the length of the candidate text, and r is the length of reference.

5.3. ROUGE-L

Recall Oriented Understudy for Gisting Evaluation or ROUGE is another metric for evaluating text generation specially in case of text summarization. ROUGE-L particularly measures the longest common subsequence (LCS) between the candidate and the reference text. ROUGE-L is more recall oriented than BLEU and focusses on how much of the reference is covered by the generated text.

ROUGE-L identifies the LCS of words which appear in both the texts maintaining the order, after which it computes the precision and recall. Precision is the amount of candidate which is present in reference text, and recall is just the opposite. The final score (f1-score) is usually the harmonic mean of precision and recall. Let us also look at all three of them mathematically:

$$R_{LCS} = \frac{LCS\ length}{number\ of\ words\ in\ refernce}$$

$$P_{LCS} = \frac{LCS\ length}{number\ of\ words\ in\ candidate}$$

$$ROUGE - L = \frac{(1+\beta^2)\times R_{LCS}\times P_{LCS}}{R_{LCS}+\beta^2.P_{LCS}}$$

where, R is recall, P is precision and β controls the relative importance of recall versus precision.

5.4. METEOR

Metric for Evaluation of Translation with Explicit Ordering or METEOR is a metric which considers not only the exact word matches but also stemming (word root) and synonyms. It combines both precision and recall, with a stronger emphasis on recall. It also includes penalties for fragmented or poorly ordered matches.

METEOR aligns the words between the candidate and reference which make it capable of seeing exact matches, stem matches and synonym matches. It calculates unigram precision and recall, and combines them using a harmonic mean (f-score) while giving higher weightage to recall. Lastly, it applies a fragmentation penalty if the matched words are not there in a contiguous sequence, penalizing the disordered or scattered matches. Let us also look mathematically at them:

$$P = \frac{number\ of\ matched\ unigrams}{number\ of\ unigrams\ in\ candidate}$$

$$R = \frac{number\ of\ matched\ unigrams}{number\ of\ unigrams\ in\ reference}$$

$$F_{mean} = \frac{10\ .P\ .R}{R + 9P}\ (default\ weights)$$

$$Penalty = \gamma\left(\frac{chunks}{matches}\right)^{\beta}$$

$$METEOR = F_{mean}\ .(1 - Penalty)$$

Where chunks are the number of matched word groups that are contiguous in order, γ and β are hyperparameters which control the penalty strength.

We get the following as a result when fine-tuning the gemma model:

```
SFT:
{"bleu": 0.056685505260577425,
"rouge-L": 0.19610763638359774,
"meteor": 0.1836898504626767}

LoRA:
{"bleu": 0.008169024296192044,
"rouge-L": 0.12541329281173502,
"meteor": 0.1256989702418963}

QLoRA:
{"bleu": 0.007569245958945915,
"rouge-L": 0.12036331784575866,
"meteor": 0.11083528786296434}
```

Figure 6: Results for Gemma: bleu, rouge-L and meteor scores for the models fine-tuned by SFT, LoRA, and QLoRA.

```
SFT:
{"bleu": 0.058441161243763896,
"rouge-L": 0.21878601043789453,
"meteor": 0.20190224909161486}

LoRA:
{"bleu": 0.01569796368126001,
"rouge-L": 0.18180380019805029,
"meteor": 0.16404506524811838}

QLoRA:
{"bleu": 0.015757359188831702,
"rouge-L": 0.180910000172323,
"meteor": 0.16753561720940716}
```

Figure 7: Results for LLaMA: bleu, rouge-L and meteor scores for the models fine-tuned by SFT, LoRA, and QLoRA.

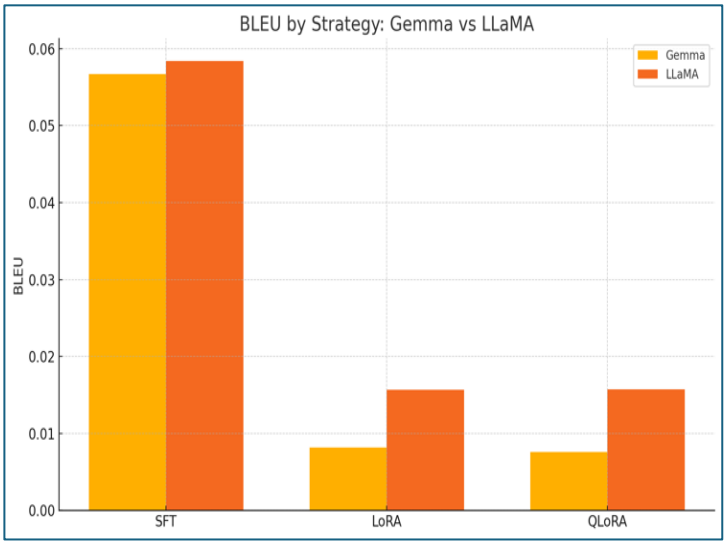

Figure 8: BLEU scores by strategy: Gemma vs Llama

While LoRA and QLoRA have similar scores, SFT performs the best. This was not surprising as SFT involves full fine-tuning of the model.

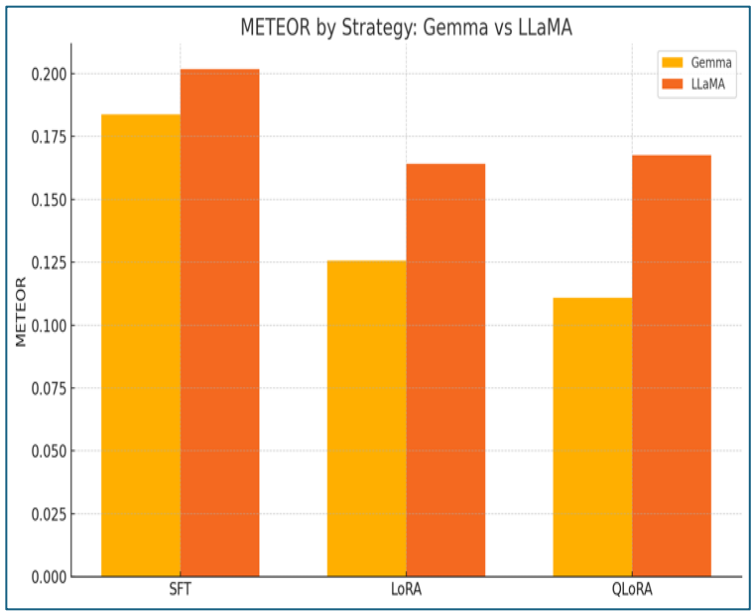

Figure 9: METEOR scores by strategy: Gemma vs Llama

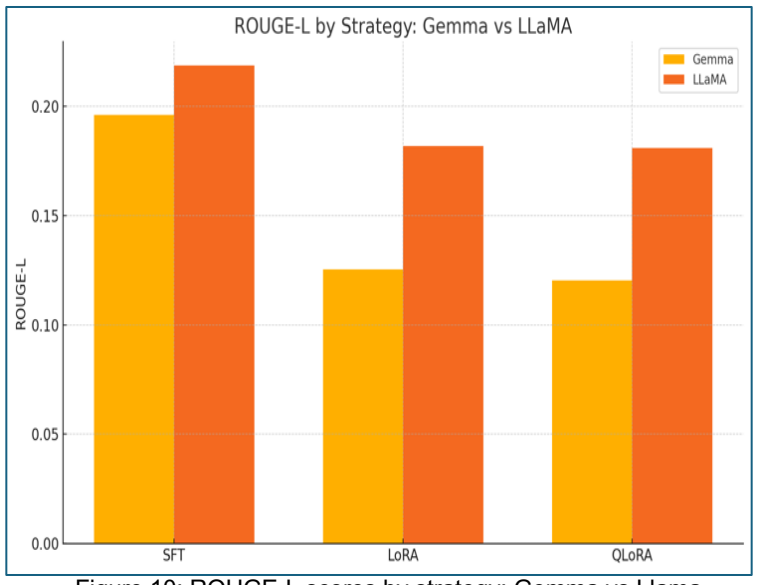

Figure 10: ROUGE-L scores by strategy: Gemma vs Llama

We also find that the LlaMA model performs slightly better than Gemma across all metrics.

Further, the poor performance in terms of the bleu, rouge-L and meteor scores can be reasoned as follows: the dataset we generated consists of problem-approach pairs where the approach is specific to a research paper. Thus, the approach includes details very specific to the paper. These details might include any acronym or terminology defined in the paper itself. When validating against a dataset like this, it understandably does poorly in terms of the scores it generates.

For understanding how performant the model actually is, we need to conduct a human study to look at the response, research the suggested approach relevance, novelty, and feasibility.

At a glance, we find the suggestions given by the model to be far more reasonable and relevant than the previous untuned models. Of course, a much more rigorous study is needed to understand the quality of the responses.

6. Lessons Learned

Let us discuss the lessons learned via this project. Starting with the dataset, we got to know that building a custom dataset or **dataset from scratch** is a time consuming, essential and persevering. Building a task specific corpus from ArXiv required us to handle noisy and malformed outputs (like extra JSON texts and incomplete entries). We got to know the essence of readily available datasets and the time and effort it take to scrap data out of a pile of gibberish and unnecessary information. We had to go through robust preprocessing steps like few-shot prompting, truncation checks and manual spot checks in order to make the fine-tuning stable.

Next, we got to know that standard automatic metrics have domain limits. BLEU, ROUGE-L, and METEOR measures the surface overlap but they too can struggle when the approach section contains any paper specific acronym or new terminology. Lastly, although our model's generations often read fluently and address the prompts sensibly, we found that the standard automated metrics yielded relatively low scores. This mismatch between metric results and actual output quality highlights the limitations of relying solely on automatic evaluation especially for complex, research style text and demonstrates the importance of incorporating human judgments to fully assess the model's usefulness and coherence.

Lastly, we can say that automated scores alone cannot capture relevance or novelty of research plans and human study like rating feasibility, completeness or clarity is essential for validating whether the fine-tuned model truly helps in real world research problem solving.

7. Reference

[1] Hugging Face, "Hugging Face – The AI community building the future," https://huggingface.co, accessed Apr. 25, 2025.

[2] Meta AI. (2024). Llama 3.2 1 B [Large language model]. Hugging Face. Retrieved April 25, 2025, from https://huggingface.co/meta-llama/Llama-3.2-1B

[3] Gemma Team, Riviere, M., Pathak, S., … & Andreev, A. (2024). Gemma 2: Improving open language models at a practical size [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2408.00118

[4] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., … & Lowe, R. (2022). Training language models to follow instructions with human feedback (arXiv:2203.02155). arXiv. https://doi.org/10.48550/arXiv.2203.02155

[5] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., … & Chen, W. (2021). LoRA: Low-Rank Adaptation of Large Language Models (arXiv:2106.09685). arXiv. https://doi.org/10.48550/arXiv.2106.09685

[6] Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). QLoRA: Efficient Finetuning of Quantized LLMs (arXiv:2305.14314). arXiv. https://doi.org/10.48550/arXiv.2305.14314