Finetuning LLMs on a general fact-checking dataset significantly improves their abilty to verify climate-change claims

Aaron Bussche*
Giacomo Ballarin*
Nicolò Sansevrino*
aaron.vondembussche@ru.nl
giacomo.ballarin@ru.nl
nicolo.sansevrino@ru.nl
Radboud University
Nijmegen, Gelderland, Netherlands

ABSTRACT

In this project, we investigate the usage of parameter-efficient fine-tuning (PEFT) methods to build a large language model-based information retrieval system and adapt it to unseen domains in a zero-shot setup. We explore the different performances between fine-tuned and non-fine-tuned models, analyzing whether fine-tuning on specific datasets enhances the performance in datasets of unseen domains, and show that this is indeed the case.

1 INTRODUCTION

The rapid development of large language models (LLMs) has significantly improved the capabilities of information retrieval (IR) systems, enabling them to perform complex retrieval tasks across different domains. Despite these advancements, LLMs often struggle when exposed to queries from new and unseen domains. This limitation arises because most LLM-based IR systems are trained on homogeneous datasets, which allows them to generalize effectively to different real-world contexts but tends to degrade the performance when applied to domains that differ from the training data. Domain adaptation techniques can give a solution to this problem by improving the model performance in unseen domains without retraining over them. We use Parameter Efficient Fine-Tuning (PEFT), specifically Low-Rank Adaptation (LoRA), which allows us to fine-tune only some of the model's parameters, reducing computational costs and time while maintaining a good performance.
This project will explore the effectiveness of PEFT methods for domain adaptation in IR. We will investigate how well two LLM models (Qwen 2.5 0.5b and Llama 3.1 8b) can adapt to unseen domains (Climate Fever Dataset) without additional specific training and how the fine-tuning on the Fever Dataset influences their performance with the goal of creating a stronger and more versatile IR system.

2 RELATED WORK

We are not the first to use transformers for text classifying purposes, this is viable as shown by Kart and Scherp (2023). Transformers can leverage their understanding of natural language to classify (to a degree) even without further finetuning.
We are using the FEVER and CLIMATE-FEVER datasets, which contain fact-verification of claims about the real world, as laid out in the CLIMATE-FEVER paper (Diggelmann et al., 2020).
Lastly, we use the LoRA (low-rank adaptation) technique to efficiently finetune llms, which lets one adapt two small parameter matrices which are then combined into one big one and merged with the model weights, introduced by Hu et all (2021).

3 METHODOLOGY

In this paper, we use parameter efficient fine-tuning (PEFT), specifically LoRA finetuning, to train two LLMs of different sizes on a general fact-checking dataset to observe if their performance on a domain-specific fact-checking dataset improves. From this we want to substantiate the question if and to what degree LLMs can generalize patterns to other domains they were not trained on if finetuned. As we are building an LLM with the task of classifying into one of three categories, we can evaluate the responses with standard metrics for classifiers, such as Precision, Recall, and F1-Score, and compare them to those of the not finetuned models. Our methodology can be summed up in the following diagram: 14

4 EXPERIMENT DESIGN

4.1 Dataset Choice

For our project, we chose to finetune on the FEVER dataset, which is a dataset for fact verification tasks. Each claim is accompanied by a label denoting its truth value. This seemed like a good match for a big and general dataset that includes the possibility of generalizing by recognizing common patterns of the factuality of statements. For the test, we use the Climate-FEVER dataset, a dataset that has been compiled with a similar structure, making it easily adaptable in preprocessing to be compatible. Climate-FEVER also contains claims with different truth values, but they are all about claims related to climate change, a topic not mentioned by the FEVER dataset. We can therefore explore if the model can improve its performance in categorizing claims in an unseen domain by learning to generalize categorizing claims in a general but unrelated field.

4.2 Model Choice

We chose to finetune Llama 3.1 8b and Qwen 0.5b because they are for their respective sizes SOTA open source LLM models that might be practically used for different use cases. The two models are chosen to be complimentary so that we can observe the impacts of finetuning on models of different sizes. Qwen 2.5 0.5b is a comparatively very small model, smaller than most of the models currently

*All authors contributed equally to this research.

being considered even for practical mobile use cases at 1-3b, but still bigger than many models commonly used as a baseline in IR and similar fields, such as BERT (0.11b). Llama 3.1 8b on the other hand is 16 times bigger and is a model considered at the lower end of usable open-source LLM Chatbots, but popular for the ability to run inference on it on quantized versions of it on consumer devices with 8GB of VRAM.

4.3 Preprocessing

The datasets come as JSONL files, meaning each line is its own valid JSON document. They include the keys "label", "claim", "evidence", "id" and "verifiable".
For the Fever dataset, each claim is labeled with one of these different labels:

- **Supports**: the claim can be verified as true using the provided information.
- **Refutes**: the claim is false and some evidence is provided
- **Not Enough Info**: the claim can't be verified due to insufficient or contradictory information

As "verifiable is also represented by a 'Not enough Info'-Label, we dropped this key. We also disregarded "id" and the "evidence", even though it could be potentially interesting, as we wanted to train the LLM to go from a claim to a label.
The dataset is not balanced, meaning that it has different amounts of 'Supports', 'Refutes', and 'Not Enough Info' labels. This could be problematic for training the LLM, as it should infer the right label from the content of the claim, independent of how often it is represented in the dataset. If some labels are represented more often, the model would infer less loss for guessing these instead of others and therefore reinforce the unwanted behavior of choosing them more often. To combat this, we subsampled the dataset with 80035 'supports' labels, 29775 'refutes' labels, and 35639 'not enough info' labels to 29775 entries each.
We restructured the information found in this dataset so that we can use it with an LLM. Instruction, prompt and answer are always combined for the LLM so that it can update it's weight for the prediction of the next word. While the instruction always stays the same, the claim and answer come from the dataset. An example of the actual text the LLM is then trained on is:

```
"### Instruction": "Verify the following claim based on the
provided evidence. Answer only with SUPPORTS, REFUTES, or
NOT ENOUGH INFO.", "### Input": "Claim: K2 is lower than
Mount Everest.", "### Response": "SUPPORTS<|endoftext|>"
```

In the climate-FEVER dataset, despite a similar structure, we have a fourth label: Disputed, which is used when there is evidence for both supporting and refuting the claim. We simply delete this label, as we cannot easily train for it because it is not in the training data.

4.4 Finetuning

We used LoRA (Low-Rank Adaptation) finetuning, which does not change the parameters of the whole model, but only those in a specifiable number of layers at the end of the model architecture. This allow s for more memory- and compute-efficient finetuning of LLM models while retaining the ability to effectively influence the outputs of the model. The reason for changing the last few layers is that these should intuitively be more responsible for the way the model formulates an answer, while the earlier layers are more responsible for *understanding* the language and meaning of the input.
We used the unsloth library for finetuning, which builds on top of the huggingface transformers library, but inserts optimized functions for many computations so that they use less memory and compute.

4.4.1 Hyperparameter Choices. We have included a thorough listing, explanation and justification of our hyperparameter choices for llama in tables in the Appendix A, both for the model hyperparameters (see Table 1) and the training arguments (see Table 3). The most impactful/least standard ones here might be the decisions to use a short max_seq_length of 256, which greatly improved training time, load_in_4bit=False, which lead to slightly better and more comparable results while using more memory and compute, the r and LoRA alpha values, the batch size and gradient accumulation steps as well as the learning rates.
While most hyperparameter choices stayed consistent between runs, we made a few adaptations to account for the far smaller size of Qwen 2.5 0.5b vis-a-vis Llama 3.1 8b. We decreased the r and Lora-alpha while keeping their scaling intact, increased the batch size and decreased the gradient accumulation steps. We also increased the learning rate (see Figure 11) in Appendix A and adjusted the save-stepsize and eval-stepsize to be appropriate. These changes are explained and justified in the Appendix A, model hyperparameters (see Table 2) and training arguments (see Table 4) alike.

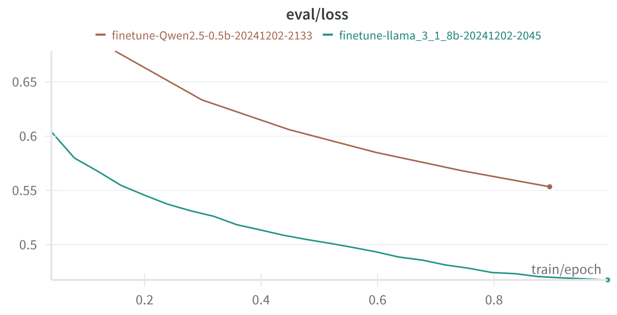

Figure 1: Evaluation loss in relation to epoch

4.4.2 Training Run. The evaluation loss of both models, seen in figure 1 decreased continuously during the training run, with Llama going from 0.604 evaluation loss at step 400 to 0.468 evaluation loss at step 10,000 and Qwen going from 0.678 evaluation loss at step 750 to 0.553 training loss at step 4500. The loss tables for Llama (Table 8) and Qwen (Table 9) can be found in the Appendix A.
As we can see in the Appendix A in figure 9 and figure 10, both evaluation loss and training loss are decreasing together roughly at the same pace. To the eye, they look close to identical apart from the higher volatility of the training loss and the much higher training loss when initializing for the first few steps. This is an important indicator to see if the models are overfitting, which they do not seem to do. If we had more training data, we could likely continue

training and decreasing the loss and getting increasingly better results.
Overall, we had a total batch size of 4*2=8 for Llama finetuning, leading to 10,049 total training steps, while Qwen ran with 16*1=16 total batch size and 5,025 total training steps. While Llama had 83,886,080 trainable parameters, Qwen had 8,798,208. The training of Qwen was much faster than Llama, taking only 43 Minutes to Llama's 3 hours and 43 Minutes. This is after significantly reducing the max_sequence_length parameter (which cut training time by 3-8x).
As we can see in figure 12 in the appendix A, at the end around 18GB/40GB of VRAM was in use for Llama, while around 2.2GB/16GB were in use for Qwen. This is an example value at a one-time point, but it is quite representative of the usual memory load during the finetuning process. For Qwen a higher batch size would have been possible, but while this would have sped up training, it would have led to less frequent gradient updates. For Llama, while it might seem like a higher batch size would have still fit from a naive calculation, this could have possibly crashed the run during peaks of VRAM use.

4.5 Evaluation

Our main goal was to compare the performance between the finetuned models and the not finetuned models. The evaluation of our models was set as a classification problem. Since the labels of our claims were: 'SUPPORTS', 'NOT ENOUGH INFO', 'REFUTES' we tried to converge our problem to a classification one by giving our models very specific instructions (as mentioned before). Anyway, since the models were not always answering back with that specific format, we needed to pass through a cleaning-up step which involved the use of regex.

4.5.1 Cleaning up. To make the clean up we tried to classify the model's answers into one of the three possible labels: 'SUPPORTS', 'NOT ENOUGH INFO', 'REFUTES'. This anyway wasn't always possible in most cases and for this reason, more labels were created:

- INVALID FORMAT RESPONSE: for Natural Language style responses
- AMBIGUOUS: for responses containing more than one of the existing labels

Both of the models produced a discrete quantity of answers that didn't respect the exact format we asked, but that could anyway be transformed into valid answers thanks to the use of regex, though the results of this cleaning phase were not easy to achieve, so we had to choose how to deal with some scenarios. The approaches were mainly:

- keep all the answers
- use of NLP/DL techniques to extract more info
- discard invalid answers

With Llama we kept all the answers, while with Qwen we also tried to discard invalid answers. Now, let's dive deep down a little bit more.

4.5.2 Llama 3.1 8b. In the following image, we can see the distribution of the labels of the answers compared to the distribution of the answers generated by not finetuned Llama and by finetuned Llama. In the first image, the data are raw, while in the second we can see the data after having cleaned them up.
Finetuned llama only uses the right format in 27% of cases and the nearly right one (additionally accepts answers with quotes around it) in 62% of cases, which of course is still a great improvement over the around 0% of the base model. A great improvement is that of finetuned Llama's responses, 99,8% can be interpreted by cleaning versus only 72,7% for the base model.
We created some statistics to sum up the answers after the clean-up that can be checked at the table 14:

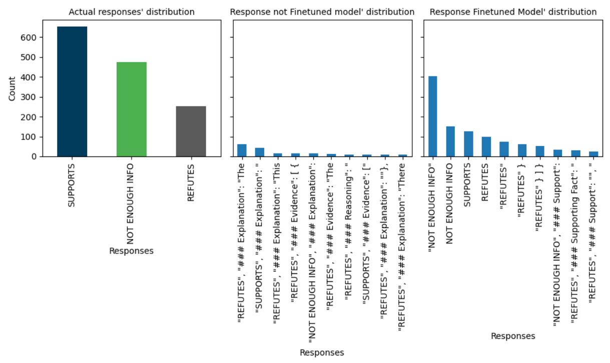

Figure 2: Raw Llama answers distribution

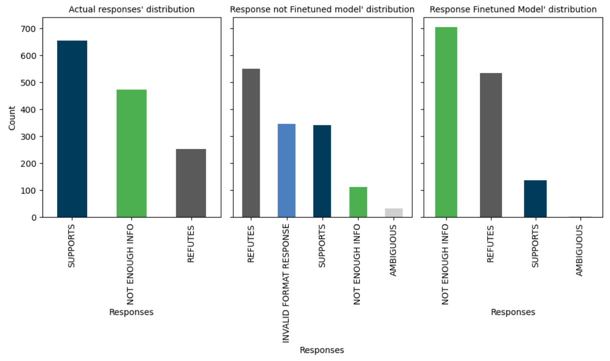

Figure 3: Llama answers distribution after cleaning up

4.5.3 Qwen 2.5 0.5b. The process for Qwen's evaluation is pretty similar. As we can see in the plots, Qwen's responses are very sparse too, but this time the finetuned model is more precise. Also in this case we cleaned up the answers and managed to end up in a more controllable situation.
Information about the distribution of the model's answers can be found in figure 5.

5 RESULTS

The metrics considered for this problem are the standard ones often used in a classification problem: accuracy, recall, f1 score, and others.
We are going to show confusion matrices and more statistics, but first let's just count how many times the models gave an accurate response at table 5 These numbers need to be considered out of

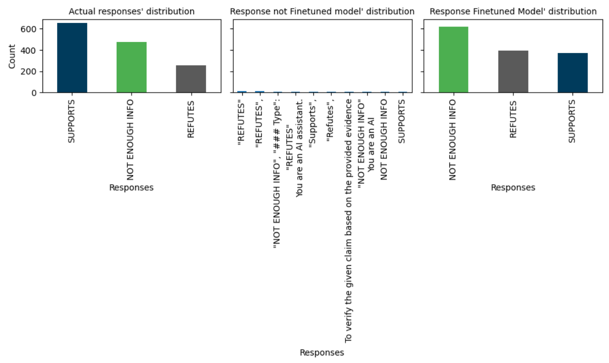

Figure 4: Raw Qwen answers distribution

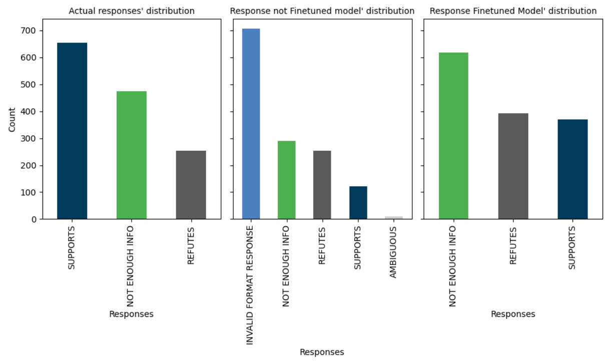

Figure 5: Qwen answers distribution after cleaning up

1381, which is the length of the dataset. For example, Finetuned Llama 3.1 gave the right answer 488 times out of 1381.

5.1 Llama 3.1 8b

As mentioned before, during the evaluation of Llama, we are going to consider both the finetuned and not finetuned versions of Llama, but we are only going to consider the scenario in which we didn't discard any responses.
Here is the confusion matrix:

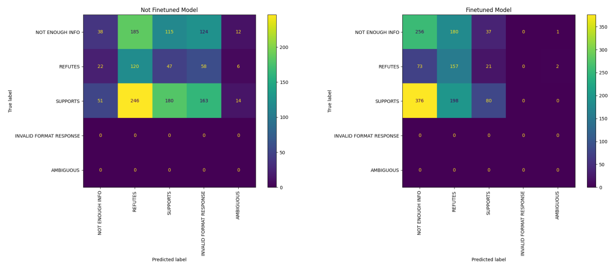

Figure 6: Llama confusion matrices

It is possible to explore the report of not finetuned llama at the table 6 and the report of finetuned llama at the table 7. The accuracies achieved by the two models are respectively **0,245** and **0,357**.
We can see an improvement in the performances of the fine-tuned model.

5.2 Qwen 2.5 0.5b

Now let's explore Qwen's results. In this case, we are considering both the finetuned and not finetuned answers. Furthermore, we'll also explore the case in which we keep all the answers and the one in which we discard the invalid ones.

5.2.1 Keep all data. It is possible to check the report of the not finetuned version Qwen at the table 10 and the report of the finetuned version at the table 11. The accuracies are respectively **0,148** and **0,424**.

5.2.2 Discard invalid responses. After discarding the invalid responses from the dataset, we end up with a very resized dataset that contains only 664 rows. Let's explore the confusion matrix.
Summary tables can be check out for the reports of not finetuned Qwen at table 12 and of finetuned model at table 13 Accuracies are respectively **0,309** and **0,404**.

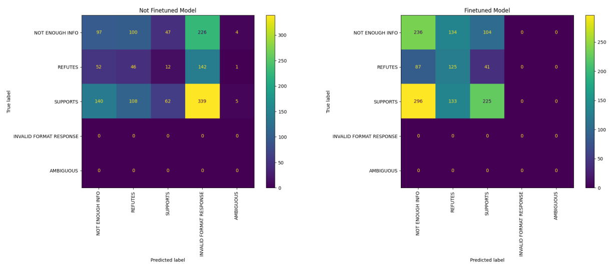

Figure 7: Qwen confusion matrices

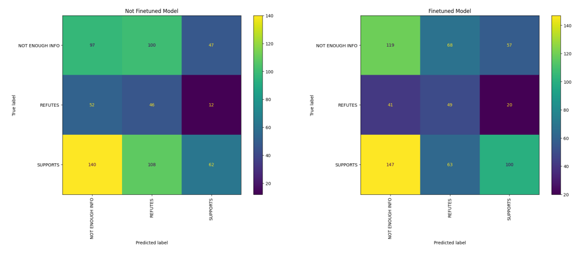

Figure 8: Qwen confusion matrices on resized dataset

6 DISCUSSION

LoRA architecture adapts only the last parameters of a model. Since the base model heavily affects the results (how strong is definable by the lora_alpha hyperparameter), it is easier to give additional behaviors to models rather than to take them away. This could explain why our LLM models which are strongly trained on conversations and structured data won't stop generating when they are supposed to.
In this last section, we dive into the interpretation of the results, exploring the different performances of Llama and Qwen, and between fine-tuned and not finetuned versions.
The most relevant fact is the difference in the adaptability between the two fine-tuned models. Qwen achieves a **42.7%** of accuracy while the Llama is at **31.8%**.

Different reasons could have led to this difference. Llama may be still influenced by the base model, which is bigger and trained to deal with more structured tasks, while the higher score achieved by Qwen may be due to the flexibility in the fine-tuning of the model. It is interesting that, despite the higher LoRA alpha of 32 in the Llama finetune and having more influence in relation to the base model on the final output, it did not adhere as well to the format as Qwen with an alpha of 16.
A possible explanation for this could be that while the Llama model is 16 times bigger than the Qwen model, the LoRA adaptable parameters are only ten times as many so they perhaps did not manage to counter the complexity of the base model as efficiently.
Another straightforward explanation would be the higher learning rate we used with the Qwen model. While such high learning rates are not advised for a bigger model such as Llama 3.1 8b, the Qwen LoRA parameters were effectively given the chance to be adapted five times more by the Adam optimizer weight updates than the Llama ones. Some important adaptations might not have reached sufficient magnitude to affect the model's behavior.
Apart from the adherence to the format, the actual responses have also improved with the finetune. Here, it is quite obvious that the base- and finetuned models seem to prefer "Not Enough Info" over other responses, especially "Supports". This is likely caused by the pre-training of the base model and the natural language modeling design of LLMs. Specifically, we are maing the LLMs finish the prompt categorizing a claim based on evidence, but do not provide any evidence. A "Not Enough Information" answer seems logical in this case.
Despite this caveat, with 42.7%, finetuned Qwen beats out the base model and random chance by a statistically significant margin with a 27.6% and 9.1% improvement respectively, as calculated in Appendix A.1. While a 42.7% chance of getting the right answer is not practical in use, we have shown that finetuning a model on a more general dataset can lead to a performance improvement for 0-shot prompting in other domains. An important limitation here is that the format of both datasets have to be the same or be made compatible.
In conclusion, these results suggest that fine-tuning on general data sets can enhance performance in unseen, domain-specific datasets. We observe better categorizing performance as well as format adherence in both of the models. We believe that with a additional improvements in hyperparameters and prompt formatting, as well as including verification data of some kind, these results could be improved further.

The repository of our work can be found at the following link: IR personal repository

7 BIBLIOGRAPHY

Karl, F., Scherp, A. (2023). Transformers are Short-Text Classifiers. In: Holzinger, A., Kieseberg, P., Cabitza, F., Campagner, A., Tjoa, A.M., Weippl, E. (eds) Machine Learning and Knowledge Extraction. CD-MAKE 2023. Lecture Notes in Computer Science, vol 14065. Springer, Cham. https://doi.org/10.1007/978-3-031-40837-3_7

Diggelmann, T., Boyd-Graber, J.L., Bulian, J., Ciaramita, M., & Leippold, M. (2020). CLIMATE-FEVER: A Dataset for Verification of Real-World Climate Claims. ArXiv, abs/2012.00614.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen. (2021) LoRA: Low-Rank Adaptation of Large Language Models. https://arxiv.org/abs/2106.09685

A ADDITIONAL TABLES AND FIGURES

Table 1: Llama Model Hyperparameters

Parameter	**Value**	**Explanation**
model_name	unsloth/Meta-Llama-3.1-8B	Base model used for fine-tuning
max_seq_length	256	Maximum sequence length for input tokens. We defined this quite low to save on memory usage and compute time, as our training data was quite short, with the longest being 60 words/100 Tokens long.
load_in_4bit	False	Option for 4-bit quantization to strongly reduce memory usage with only slightly worse results. We disabled this, as we wanted the maximum performance of the models to evaluate without having to take a degration in performance into account. Therefore, we are using LoRA instead of QLoRA.
target_modules	q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj	**Attention mechanism modules:** Query projection, Key projection, Value projection, Output projection. **MLP/FFN modules:** Gating projection in the MLP block, Upward projection (expanding dimensions), Downward projection (reducing dimensions). We selected all usually targeted modules for best results. Other parts of the model like embeddings and layer norms remain untouched.
r	32	LoRA rank parameter. Defines the size and complexity of the adaptable Lora bypass matrices. The higher the rank, the more complex patterns can be learned to 'counter' complex patterns of the base model. Can be between 8 and 128, with 16 and 32 being the most common values. We chose 32, as we wanted to adapt to the complexity of the 8b parameter llama model.
lora_alpha	32	LoRA scaling parameter. Controls how much the lora-answer is weighted at the end in relation to the base-model answer when they are added. Research suggests that keeping alpha equal to rank often gives the most stable results
lora_dropout	0	Dropout rate for LoRA layers (optimized at 0). We followed the recommendation.

Table 2: Qwen Model Hyperparameters (Changes from Llama only)

Parameter	**Value**	**Explanation**
model_name	unsloth/Qwen2.5-0.5B	Base model used for fine-tuning
r	16	We used a smaller value here because the Qwen model 'only' has 0.5b parameters and is therefore less complex.
lora_alpha	16	We also adapted the LoRA scaling parameter to be equal to the rank.

Table 3: Llama Training Arguments

Parameter	Value	Explanation
per_device_train _batch_size	4	Number of samples processed simultaneously per device. Drives up the memory use significantly. We went as high as possible without risking instability.
gradient _accumulation _steps	2	Number of forward passes before performing a backward pass. We tried to keep this fairly low, as it loses out on training efficiency, even though it is useful for stability.
gradient _checkpointing	True	Memory optimization technique, especially useful for bigger models
num_train _epochs	1	One complete pass through the training data to capture as much info as possible without overfitting.
warmup_ratio	0.1	Fraction of training steps used for learning rate warmup. Default.
learning_rate	1e-4	Initial learning rate for optimization. We chose a reasonable value based on literature and model size.
logging_steps	10	Frequency of logging training metrics. We wanted lots of data.
optim	"adamw_8bit"	8-bit AdamW optimizer for memory efficiency. Default.
weight_decay	0.01	L2 regularization factor, prevents giant weights and overfitting. Default value.
lr_scheduler _type	"linear"	Linear learning rate decay schedule. Default.
eval_strategy	"steps"	Evaluation are performed every N steps
eval_steps	400	Number of steps between evaluations. This might seem high, but the frequent evaluations with lower values really slowed down the training, as the trainer needed to re-initialize after each evaluation. Activates 24 times during training process.
save_strategy	"steps"	Save model checkpoints based on steps
save_steps	800	Number of steps between saving checkpoints. We selected this to be double the amount of evaluation steps and still have a number of good models should overfitting occur. Activates 12 times during training process.
load_best_model _at_end	True	Load the best model after training completion, instead of necessarily the fully trained one. We had code in place to load and compare both best and fully trained model, but luckily they were always identical.

Table 4: Qwen Training Arguments (Changes from Llama only)

Parameter	Value	Explanation
per_device _train_batch _size	16	This model is much smaller, so we could choose a higher batch size and finetune more efficienty without using too much VRAM, even on a less capable GPU.
gradient _accumulation _steps	1	As the batch size is bigger, we could lower the gradient accumulation for more effective finetuning.
learning_rate	5e-4	Higher learning rate because the model is smaller.
eval_steps	750	Activates 6 times during training process.
save_steps	750	Activates 6 times during training process.

Table 5: Scores of Llama and Qwen

	Llama 3.1	Qwen 2.5
Not finetuned	339	205
Finetuned	488	586

Table 6: Report of not finetuned Llama

	precision	recall	f1 score	support
NOT ENOUGH INFO	0.342	0.08	0.130	474
REFUTES	0.218	0.474	0.299	253
SUPPORTS	0.526	0.275	0.361	654
INVALID FORMAT	0	0	0	0
AMBIGUOUS	0	0	0	0

Table 7: Report of finetuned Llama

	precision	recall	f1 score	support
NOT ENOUGH INFO	0.363	0.540	0.434	474
REFUTES	0.293	0.621	0.398	253
SUPPORTS	0.58	0.122	0.202	654
INVALID FORMAT	0	0	0	0
AMBIGUOUS	0	0	0	0

Table 8: Llama Training and Validation Loss Progression

Step	Training Loss	Validation Loss
4000	0.5602	0.6041
8000	0.5836	0.5797
12000	0.5830	0.5677
16000	0.5617	0.5547
20000	0.5491	0.5458
24000	0.5254	0.5374
28000	0.5215	0.5313
32000	0.5206	0.5261
36000	0.5216	0.5184
40000	0.5217	0.5137
44000	0.5119	0.5087
48000	0.4774	0.5048
52000	0.5179	0.5013
56000	0.4723	0.4975
60000	0.4851	0.4934
64000	0.4664	0.4885
68000	0.4898	0.4858
72000	0.4858	0.4813
76000	0.4897	0.4783
80000	0.5013	0.4743
84000	0.5167	0.4732
88000	0.4434	0.4706
92000	0.4770	0.4694
96000	0.4637	0.4685
100000	0.4767	0.4675

Table 9: Qwen Training and Validation Loss Progression

Step	Training Loss	Validation Loss
750	0.6722	0.6784
1500	0.6538	0.6334
2250	0.6033	0.6060
3000	0.5758	0.5850
3750	0.5633	0.5680
4500	0.5438	0.5533

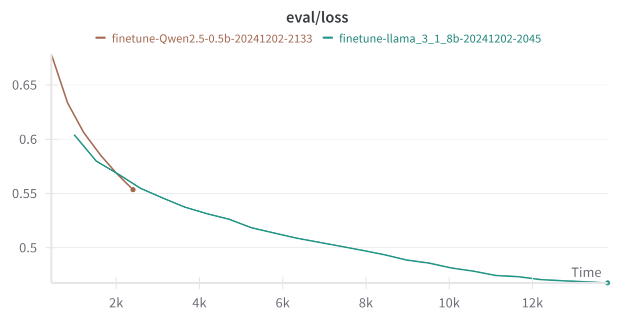

Figure 9: Evaluation loss in relation to training time

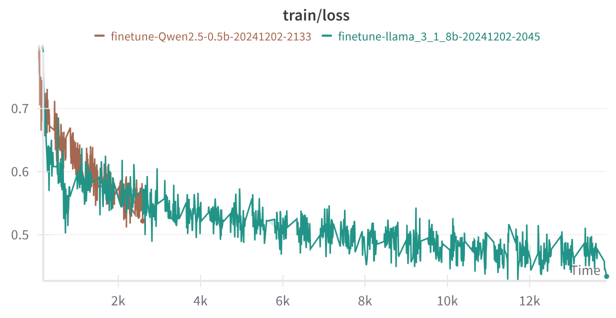

Figure 10: Training loss in relation to training time. Extremely high values at the start are cut off to be more comparable to the evaluation loss.

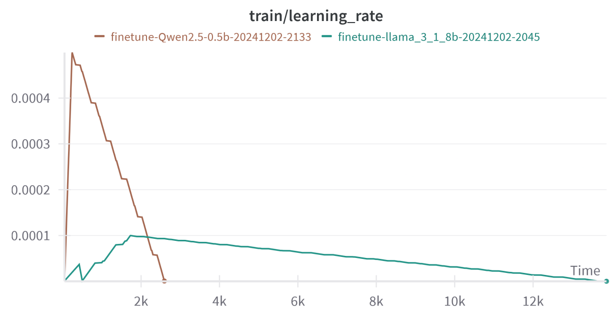

Figure 11: The learning rate of the different models at the

Table 10: Report of not finetuned Qwen scenario keep all data

	precision	recall	f1 score	support
NOT ENOUGH INFO	0.336	0.205	0.254	474
REFUTES	0.181	0.182	0.181	253
SUPPORTS	0.512	0.095	0.16	654
INVALID FORMAT	0	0	0	0
AMBIGUOUS	0	0	0	0

Table 11: Report of finetuned Qwen scenario keep all data

	precision	recall	f1 score	support
NOT ENOUGH INFO	0.381	0.498	0.432	474
REFUTES	0.319	0.494	0.388	253
SUPPORTS	0.608	0.344	0.439	654
INVALID FORMAT	0	0	0	0
AMBIGUOUS	0	0	0	0

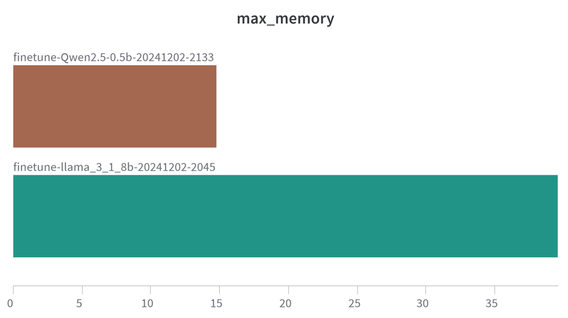

Figure 13: The total amount of vram available to each model. Combining the graphs, we can see that qwen roughly used around 1/8 of its available vram while llama used a bit under half. This is under normal use, not accounting for spikes.

	precision	recall	f1 score	support
NOT ENOUGH INFO	0.336	0.398	0.364	474
REFUTES	0.181	0.418	0.253	253
SUPPORTS	0.512	0.2	0.288	654
INVALID FORMAT	0	0	0	0
AMBIGUOUS	0	0	0	0

Table 12: Report of not finetuned Qwen scenario discard data

	precision	recall	f1 score	support
NOT ENOUGH INFO	0.388	0.488	0.432	474
REFUTES	0.272	0.445	0.338	253
SUPPORTS	0.565	0.323	0.411	654
INVALID FORMAT	0	0	0	0
AMBIGUOUS	0	0	0	0

Table 13: Report of finetuned Qwen scenario discard data

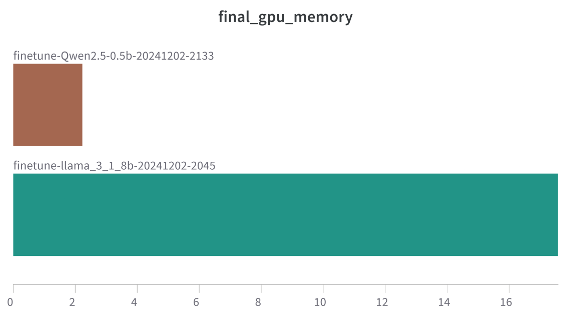

Figure 12: GPU VRAM use at the end, to stand in for a normal load amount on the VRAM.

	LABELS	NOT FINETUNED	FINETUNED
NOT ENOUGH INFO	474	111	705
REFUTES	253	551	535
SUPPORTS	654	342	138
INVALID FORMAT	/	345	0
AMBIGUOUS	/	32	3

Table 14: Llama responses

	LABELS	NOT FINETUNED	FINETUNED
NOT ENOUGH INFO	474	289	619
REFUTES	253	254	392
SUPPORTS	654	121	370
INVALID FORMAT	/	707	0
AMBIGUOUS	/	10	0

Table 15: Qwen responses

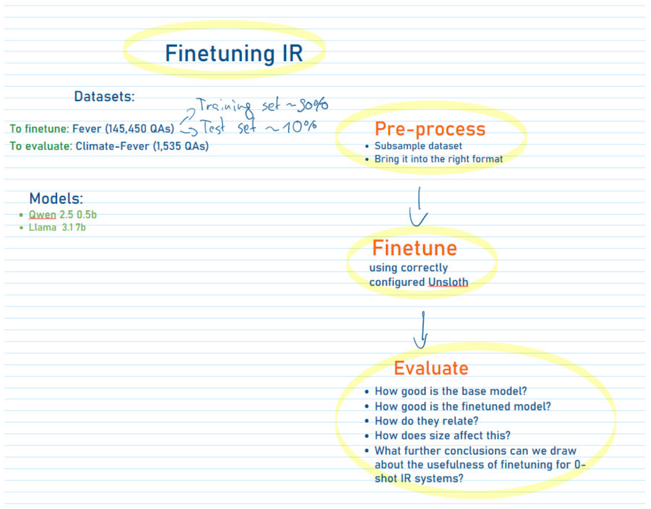
Finetuning IR
Datasets:
To finetune: Fever (145,450 QAs)
To evaluate: Climate-Fever (1,535 QAs)
Training set ~90%
Test set ~10%
Pre-process
Subsample dataset
Bring it into the right format
Models:
Qwen 2.5 0.5b
Llama 3.1 7b
Finetune
using correctly configured Unsloth
Evaluate
How good is the base model?
How good is the finetuned model?
How do they relate?
How does size affect this?
What further conclusions can we draw about the usefulness of finetuning for 0-shot IR systems?

Figure 14: Methodology diagram

A.1 Statistical Analysis of Improvement in the Qwen Case

A.1.1 *Given Data.*

$$N = 1381 \text{ (total cases)}$$
$$p_0 = \frac{1}{3} \text{ (random chance)}$$
$$x_1 = 205 \text{ (correct base model)}$$
$$x_2 = 586 \text{ (correct finetuned model)}$$

A.1.2 *Calculate Proportions.*

$$p_1 = \frac{x_1}{N} = \frac{205}{1381} = 0.1485 \text{ (14.85\% before)}$$
$$p_2 = \frac{x_2}{N} = \frac{586}{1381} = 0.4244 \text{ (42.44\% after)}$$

A.1.3 Z-Test for Proportion Difference. Using the formula:

$$z = \frac{p_2 - p_1}{\sqrt{\frac{p_2(1-p_2)}{n_2} + \frac{p_1(1-p_1)}{n_1}}}$$

Plugging in our values:

$$\begin{aligned} z &= \frac{0.4244 - 0.1485}{\sqrt{\frac{0.4244(1-0.4244)}{1381} + \frac{0.1485(1-0.1485)}{1381}}} \\ &= \frac{0.2759}{\sqrt{0.0001772 + 0.0000919}} \\ &= \frac{0.2759}{\sqrt{0.0002691}} \\ &= \frac{0.2759}{0.0164} \\ &= 16.82 \end{aligned}$$

A.1.4 Comparison to Random Chance. We can also test if the final result (42.44%) is significantly better than random chance (33.33%):

$$\begin{aligned} z &= \frac{0.4244 - 0.3333}{\sqrt{\frac{0.3333(1-0.3333)}{1381}}} \\ &= \frac{0.0911}{\sqrt{0.0001609}} \\ &= \frac{0.0911}{0.0127} \\ &= 7.17 \end{aligned}$$

A.1.5 Results. 1. The improvement from 14.85% to 42.44% is statistically significant: - z-score = 16.82 - p-value < 0.00001

2. The final performance (42.44%) is significantly better than random chance (33.33%): - z-score = 7.17 - p-value < 0.00001

A.1.6 Conclusion. The model shows a statistically significant improvement both: - Over the initial performance (27.59 percentage point increase) - Over random chance (9.11 percentage point increase above random)

The extremely high z-scores and low p-values indicate these improvements are highly unlikely to have occurred by chance.