Cole Stevens (300171413)

Mikaela Dobie (300164161)

William Beaupre (300170535)

Machine Learning With Large Language Models

Training Meta’s LLaMA to Detect Fake News

Professor: Mohammad Alnabhan

<https://github.com/c-stev/CSI-4900>

# Abstract

This project explores the use of Large Language Models (LLMs), which in this case was demonstrated with LLMs from Llama (Large Language Model Meta AI) – a relatively new LLM developed by Meta – for the purpose of fake news detection. Given the rapid spread of fake news, current detection methods are mostly manual, which in today’s digital age proves to be very inefficient. This project determined that while no significant improvements were observed in smaller parameter versions of the Llama models, an 8-billion parameter version showed potential, suggesting that future research into high parameter models and more epochs could produce better results.

# Introduction

Fake news detection has proven to be more and more of an important task in today’s environment, where this sort of content has proven to spread quickly and have a devastating ripple effect. Currently, fake news detection consists of human beings having to manually go through every prospect article, cross-check the facts, and then finally come to a conclusion. Large Language Models (LLMs) being able detect underlying patterns in false articles and flag them almost instantly offers significant time-saving benefits.

LLMs, such as those in the Llama (Large Language Model Meta AI) family, have proven to be effective in tasks such as text classification. The goal of this project is to evaluate the performance of Llama models given varying parameter sizes on this fake news detection task. By analyzing these models, this work aims to contribute insights towards optimizing future LLM training for fake news detection tasks.

## Report Organization

1. *Abstract:* introduces the topic and generally explains the conclusions drawn from the project.
2. *Introduction:* Discusses the significance of fake news detection, the role of LLMs in addressing this challenge, and the goals of the project.
3. *Background:* Details the division of tasks among team members and endeavours that were not able to be accomplished during this project’s timeframe.
4. *Methodology and Experimental Setup*

* *Models and Parameters:* Details the motivations behind the selection of Llama models, their specifications, and hyperparameters used.
* *Training & Fine-Tuning:* Explains the difference between training and fine-tuning, the workflow for each, and how LoRA modules were applied to optimize resource usage.
* *System Overview:* Describes the workflow of the project, including data collection, preprocessing, and training.
* *Data and Dataset Preprocessing:* Describes the datasets used to train and test the model.
* *Dataset Splits and Hyperparameter Optimization*: Explains the decision-making process behind the split percentages used and describes the hyperparameter tuning process.
* *Server/Computer Specifications and Environment Setup:* Outlines the hardware and software environments used for the project.
* *Layer Freezing and Fine-Tuning:* Details the use of LoRA for freezing specific layers to improve training efficiency.
* *Imbalance and Dataset Balancing*: Explains how class imbalance was addressed and its impact on model performance.
* *Error Analysis*: Describes key issues encountered during the project, and describes steps taken to address these problems.

1. *Performance Evaluation & Results:* Details the results and limitations of all models, while concluding on which one would be best for future research.
2. *Conclusion and Future Work*: Summarizes key findings of the project, highlights some challenges faced, and suggests future directions to take for improving LLM fake news detection.
3. *References*: Lists all datasets used throughout the project.

# Background

Most tasks, which principally include data preprocessing, model training, and model evaluation, were collectively worked on by all 3 participants at different points in the project. To make more efficient use of time, each participant was assigned to train and evaluate a separate model on the same datasets on various epochs (and perform intermediate bug-fixes if the situation called for it); Mikaela Dobie was responsible for Llama 3.2 1b, William Beaupre was responsible for Llama 2 7b, and Cole Stevens was responsible for Llama 3.1 8b.

The initial goal of this project was to train the models on all categories of news at first and then train speciality models that excel at predicting the veracity of news articles for a single category. However, due to time constraints, this endeavour was marked as something that can be pursued in the future and was not pursued any further for this project.

# Methodology and The Experimental Setup

## Models and Parameters

The decision to focus on Llama models was guided by its relative newness and proven effectiveness in language-based tasks. For this project, we decided to use Llama 3.2 1b, Llama 3.1 8b and Llama 2 7b. The decision to pick such low parameter counts for the models was primarily due to hardware limitations. Specifically, our team lacked the RAM/VRAM capacity required to fine-tune models with high parameter counts. We included Llama 2 7b to compare the performance of a previous-generation Llama model against the newer Llama 3 versions. By including both Llama 3.2 1b, and Llama 3.1 8b, we aimed to evaluate the impact of parameter size on language-processing classification tasks. The chosen models possess the following specifications:

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Attention Heads** | **Hidden Layers** | **Trainable Parameters** |
| Llama 2 7b | 32 | 32 | 39976960 |
| Llama 3.1 8b | 32 | 32 | 41943040 |
| Llama 3.2 1b | 32 | 16 | 11272192 |

*Table: Llama models used and their specifications*

To keep the experiment results consistent for each of the models, the training arguments remained consistent. Some values were set due to the hardware limitations, but others – such as learning rate – were determined by testing out different values and concluding which was the most optimal. Below is a subset of these training arguments:

|  |  |
| --- | --- |
| **Argument** | **Value** |
| per\_device\_train\_batch\_size | 16 |
| gradient\_accumulation\_steps | 4 |
| learning\_rate | 2e-4 |
| optimizer | “lion\_8bit” |
| weight\_decay | 0.01 |
| lr\_scheduler\_type | “cosine” |

*Table: Hyperparameters used for LLM fake news classification*

## Training & Fine-Tuning

Training a model is when you start with a base model from scratch with the default random weights then you give the model data and set the parameters. Then you adjust the model based on the given loss and optimizer. Fine-tuning a model on the other hand is taking a model that has been trained on a set and then adapting it to suit the task by further training the model on new data. Fine-tuning a model uses the model's base knowledge and modifies some specific parts of the model or even fine-tunes the whole model.

The workflow of training a model requires data preparation that has a larger dataset of labelled data for supervised learning as it is starting from scratch. Then with the data you clean it by removing unlabelled data and making all data have the same columns. Then you split the data into a training and an evaluation dataset. The model initialization for training will start with a small number of random weights as it is not already trained on any previous data. Then for training, it does two steps a forward pass that passes the data in the model and then backpropagation which calculates the gradients using the loss function and updates the weights. The training then continues for the number of epochs chosen. After training the model you evaluate it to get the loss and f1 score on a test dataset and based on the result you fine-tune the model to adjust the model.

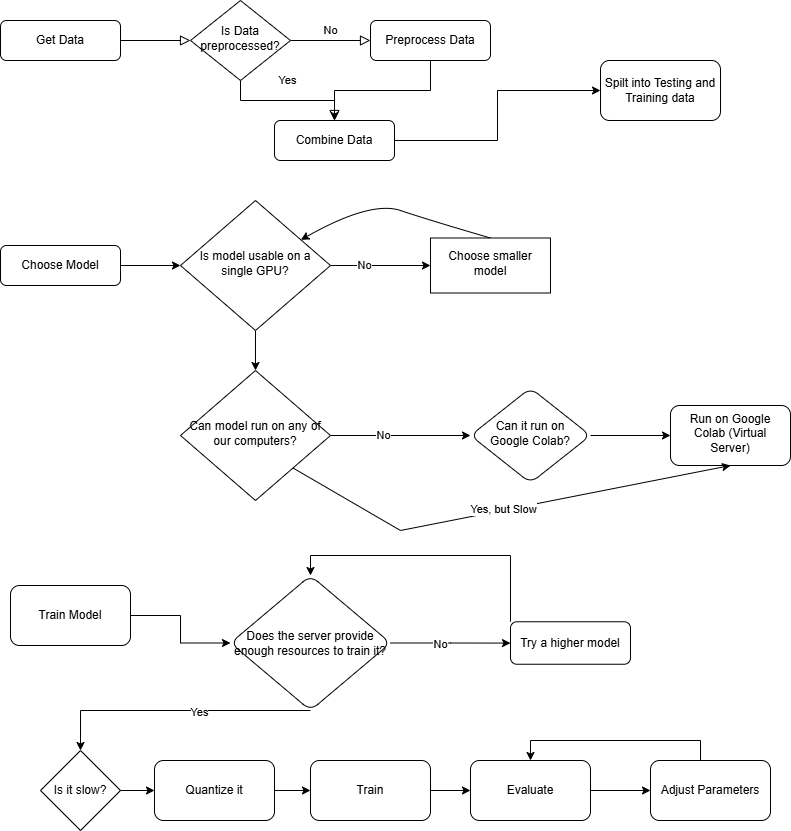
For fine-tuning a pre-trained model, the workflow is like training but slightly different. For fine-tuning you start with selecting the model to work with which is based on what you need, this model will have already been trained on large open-source datasets. Usually when fine-tuning a model, you need a smaller dataset that is more specific to the needs of the model. When fine-tuning a model, you usually need to freeze some layers to keep the general features and just change the more detailed layers. Then we can fine-tune the model where we train the unfrozen layers, when fine-tuning we reduce the learning rate as we do not want to overfit the model. Lastly, we evaluate the results like when training the model and decide whether more fine-tuning is needed or if the model is ready.

The need for training a model is a large dataset as the model has never seen any data, a large amount of computing power as there is a large amount of data and parameters needed to train. Training a model will also take much longer than fine-tuning as there is more data and it is starting from scratch. Fine-tuning a model on the other hand requires a small specific dataset as it has already been trained on a large dataset. Fine-tuning needs less computing power as it is fine-tuning fewer parameters and using a smaller dataset. Fine-tuning requires less time as it updates only a small portion of the parameter.

For parameters training a model from scratch will train all the parameters and will have a higher learning rate. The loss function will be set based on the task and is done automatically in some cases. The optimizer will be chosen based on the task that the model needs to accomplish, batch size should be higher than fine-tuning with a larger epoch count to converge. Fine-tuning will only have a subset of parameters be trainable and earlier layers that are more general are usually frozen. The learning rate for the fine-tuning will be lower than training to prevent overfitting. The loss function and the optimizer like in training are based on the task and it will have smaller epochs and batch sizes compared to training as it requires less updating.

When fine-tuning the model, we did not fine-tune all the layers. Instead, we fine-tuned a subset of layers using LoRA, specifically, we targeted the attention and feedforward projection layers ("q\_proj", "k\_proj", "v\_proj", "o\_proj", "gate\_proj", "up\_proj", "down\_proj"). These layers were chosen as they are the most relevant for the classification of text. Q\_proj projects the input to a query vector, k\_proj projects the input to a key vector, and v\_proj projects the input to a value vector. Then we have o\_proj that outputs the processed attention result. We chose to target gate\_proj as it controls the flow of information, which helps the model represent complex relationships in text. Lastly, we chose to target up\_proj and down\_proj as down\_proj reduces the dimensionality of intermediate representations and up\_proj expands the dimensionality back for non-linearity. Updating up and down projection helps the model capture features for tasks like classification. We chose these LoRA modules to target the layers responsible for these modules.

## System Overview



|  |  |
| --- | --- |
| **Model Used** | **Parameters** |
| Llama 2 7b | Trainable Layers: 448  Total Layers: 739 |
| Llama 3.2 1b | Trainable Layers: 224  Total Layers: 370 |
| Llama 3.1 8b | Trainable Layers: 448  Total Layers: 739 |

*Table: Models used and their parameters.*

As seen in the flowchart, we had three main processes in this experiment, starting with data collection and ending in a fine-tuned model for detecting fake news. Firstly, we started by collecting the data from open-sourced websites provided by Professor Alnabhan. We then preprocessed the data and combined it based on their subject matter and whether we had it labelled for training or testing. That is the end of step one. Following the data preparation we move on to choosing and downloading the base models that would be used for training. The main roadblock encountered in choosing a model is finding one that will run on a single GPU. Even so, we still needed to find a better way to run the model as the time it would take to train the model would not be useful for getting proper results. Which led us to the use of the virtual server on Google Colab. This allows us to run the model at a reasonable pace to do multiple experiments and compare. This leads to the last step of training and fine-tuning the model. On Colab there are four different GPUs that are available for use, with how large some of the models were and how often we needed to run them we ended up having to use the highest model, A100, to save the most time and ensure it runs without hitting an error. We then quantized the model to be more efficient. After ensuring it runs, we evaluated the models adjusted hyperparameters and trained further.

To attempt to run this experiment yourself, you must first request permission to use the models via HuggingFace and get a token. They then must use the correct jupyter notebook file regarding the model on the GitHub repo. Then they must paste their HuggingFace token into the first code block and ensure the clean data files exist on the Google Drive account associated with the Colab project. They then must either subscribe or purchase credits for Google Colab to be able to run the script on the A100 GPU, any other selection with get an error when training. One then should be able to click run all to train the model and adjust parameters as needed. Adjustments too high to any parameter may cause the server to run out of the maximum allowed resources.

## Data & Dataset Preprocessing

This project involves collecting data, preprocessing the data so that it is fit to use, choosing a large language model, training the model, and lastly optimizing it to get the best results. For this experiment, as Professor Alnabhan has done prior research, he provided us with 12 open-source datasets to use for training and testing. The data sets include FA-KES, Snopes, Covid Claims, Covid-19 Fake News Infodemic Research Dataset, Covid FNIR, FakeNews, ISOT, LIAR, Pheme Veracity, Politifact, Climate Dataset, and GossipCop. The data can be classified into five categories depending on the types of news: Crime, Health Politics, Science, and Social Media.

|  |  |
| --- | --- |
| **Dataset** | **Number of Rows Before Preprocessing** |
| FA-KES | 805 |
| Snopes | 37957 |
| COVID Claims | 7587 |
| COVID-19 Fake News Infodemic Research | 3002 |
| COVID-FNIR | 7588 |
| FakeNews | 20799 |
| ISOT | 44919 |
| LIAR | 12836 |
| Pheme Veracity | 2400 |
| Politifact | 73765 |
| Climate Dataset | 1535 |
| GossipCop | 22153 |

*Table: List of Datasets used and their sizes*

The total number of rows of the combined table would be 235,346 before preprocessing. To preprocess the datasets, we first imported them from each of their respective sources. All except Pheme were in formats that could be manipulated using the Pandas library in Python. The formatting for the Pheme dataset was in annotations which required using a Python script to convert it to a comma-delimited file. Using and compiling parts of the script provided in the annotations, a script provided by Professor Alnabhan, and some additions on our part as well, we were able to overcome this block in the data processing steps.

To preprocess the data, we discarded all features except for the title, the subject, and the label. The reason we chose to keep these specific features was that we planned to combine all the acquired datasets and needed a common value for identification and the label to classify it. As some of the datasets like Snopes and Politifact only contained URL values for the piece of information, we are only able to use the titles and labels to join the datasets. The subject was kept for the original datasets to be sorted into the five topics. The next step we took was to make sure all the labels were either binary 0 or 1. As some of the files contained indistinct labels such as “mostly true” and “probably false,” we went through the list of possible values in the datasets and defined them to either be true or false to be able to have a binary classification and then converted them to be numerical values of 0 and 1 so that the model could make use of them. Finally, we concatenated all the datasets together based on their subjects and the process we were using them for, training or testing.

|  |  |  |
| --- | --- | --- |
| **Subject** | **Training** | **Testing** |
| Crime | FA-KES | SNOPES |
| Health | COVID-FN + COVID-FNIR | COVID\_CLAIMS |
| Politics | FAKENEWS + ISOT + LIAR | PHEME + POLITIFACT |
| Science | CLIMATE | ISOT (Science Part) |
| Social | GOSSIPCOP | ISOT (Social Part) |

*Table: Concatenated datasets spilt into their subject matter*

As this experiment faced many challenges, we focused on training and finetuning a model with all the datasets rather than on their classifications, keeping only the title or renamed “text” value and their binary labels. This left us with a dataset with 139,438 rows for testing and 70,383 for training. After using this dataset for some training attempts and having very skewed results due to the imbalance of data, we decided to balance the data to have equal parts true and false labelled data, 55,210 for training and 88,582 for testing.

## Dataset Splits and Hyperparameter Optimization

The dataset split that we used on the training dataset was a 70/30 split meaning that we kept 70% of the dataset rows for testing and then 30% for validation. We thought that a 70/30 split would be best as it would allow us to validate more than a 25% validation or lower but would leave enough data for training. We then used the testing dataset that was unseen from the training and used it to evaluate each model and get the average, f1 score, and confusion matrices.

For hyperparameters, we started with Adamw-8bit as the optimizer, with a learning rate of 2e-4 and a train batch size of 64 total with 16 per device and a gradient accumulation step of 4. We had also selected a max token length of 64 per text section to minimize the RAM utilization as we were running out of the 40Gb of RAM on the GPU. We decided to pick these hyperparameters as well as warmup step 5 weight decay of 0.01. We chose to start with a low weight decay as we were not worried about overfitting the dataset and could change it later if overfitting became an issue. The results that we got using these hyperparameters were not satisfactory so we decided to change the hyperparameters more, we found the max tokens that we could use on the A100 GPU to be 256 for each text. After we found that out, we decided to look at batch size but when we tried to increase the batch size anymore, we would run out of RAM before the end of the fine-tuning. We chose to select LoRA r of 16 as it was a good middle ground between the size of the low-rank matrices and memory utilization. Next, we chose LoRA alpha of 16 as we saw that it was common practice to set it equal to r. We decided on a LoRA dropout of 0 as we did not run into overfitting and found it unnecessary. Lastly for LoRA, we decided to target the modules “q\_proj", "k\_proj", "v\_proj", "o\_proj", "gate\_proj", "up\_proj", "down\_proj" as the modules are important for classification. We wanted to look at all the possible optimizers as well as AdamW is the default option and found Lion which is more efficient for large models.

To balance the datasets, we noticed that there was more false news than true news thus we decided to remove the excess false news by running a code that would pick a random sample of true and false news equivalent to the lowest number which in our case was true. Once these samples were taken, we had 28663 true and 28663 false news in our training dataset. We also noticed that since it was guessing mostly false it would be best to pick a balanced dataset for the testing as well to get a more accurate f1 score.

To adjust the weights in our model we decided to use LoRA which modifies a subset of the parameters while keeping most of the pre-trained weights frozen. LoRA introduces trainable low-rank matrices into the model. These matrices modify the outputs of the pre-trained layers, especially the projection layer in our case that we targeted (“q\_proj", "k\_proj", "v\_proj", "o\_proj", "gate\_proj", "up\_proj", "down\_proj"). During fine-tuning only, the LoRA matrices were updated while the original weights of the model stayed frozen. Using LoRA to freeze the layers was necessary as the amount of memory training all the model layers or even a portion of the layers needed was way higher than the memory, we had access to of 40GB. We also adjusted the weights using Lion-8bit optimizer in the end which calculates the loss and updates the weights to minimize the loss. Lastly, we used a learning rate of 2e-4 to control the size of each weight update and made gradual adjustments using the cosine learning rate scheduler to decay the learning rate and stabilize training.

## Server/Computer Specifications and Environment Setup

Our local machines lacked the necessary RAM and GPU power required to handle the fine-tuning process of any Llama model. Because of this, we rented the significantly more powerful GPUs offered by Google Colab so they could help in training our models. Specifically, the A100 GPU was used, which boasts an impressive 83.5 GB of system RAM, 40 GB of GPU RAM, and 235.7 GB of disk space. Google Colab offered some lower-RAM GPUs, such as the L4 and T4 GPU, but these weren't large enough to handle the processing of our entire dataset, so they weren't selected. Additionally, Google Colab offered some Tensor Processing Units (TPUs), but these TPUs were not able to work with our code as it was configured, so they were not selected either.

The training and evaluation processes were conducted on a Jupyter Notebook file hosted on Google Colab. The notebook was integrated with Google Drive, which served as the host of our cleaned datasets. The raw datasets were prepared and cleaned in a separate Python script and then accessed remotely by the Jupyter Notebook file. Google Drive was used to store our datasets because manual uploads to Google Colab were not possible because of the size of the datasets.

Despite the computational power of the A100, it alone still could not process the datasets through the models due to their size and complexity. To fix this issue, we used Unsloth's configuration of Meta's Llama models, which allows for a lower memory overhead and faster training efficiency. By using Unsloth we were able to fine-tune the Llama models within the constraints of our hardware environment.

## Layer Freezing and Fine-Tuning

For this project, we dealt with many challenges due to limited hardware and software resources, which hindered our ability to experiment with common hyperparameter tuning techniques like grid search or random search. To overcome this, we implemented quantized training, which allowed us to reduce the computational power used. Additionally, we implemented Low-Rank Adaptation (LoRA), a modular and efficient approach specifically for fine-tuning LLaMA models. LoRA enabled us to focus on specific layers, q\_proj, k\_proj, v\_proj, o\_proj, gate\_proj, up\_proj, and down\_proj, while freezing the remaining layers implicitly, thereby conserving resources and improving training efficiency.

Despite resource constraints, we explored other fine-tuning strategies, including introducing dropout; however, this did not yield better results. Weight decay was applied as a regularization technique to prevent overfitting. To optimize training time, we adjusted the batch size and incorporated gradient accumulation, allowing us to effectively simulate larger batches without exceeding memory limits.

Finally, we used mixed-precision training with the fp16 setting, which significantly reduced memory usage and enabled us to train multiple epochs without encountering out-of-memory errors. These adaptations were crucial to dealing with our resource constraints and ensuring successful model training.

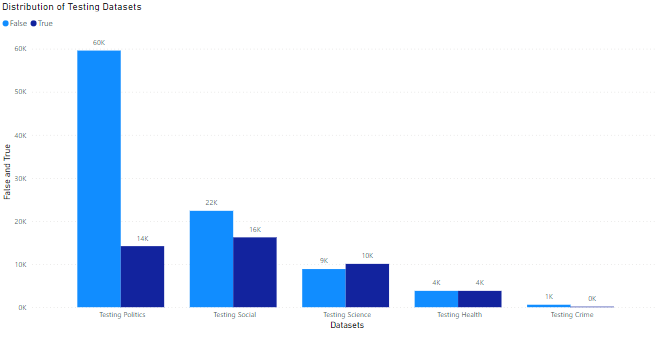
## Imbalance and Dataset Balancing

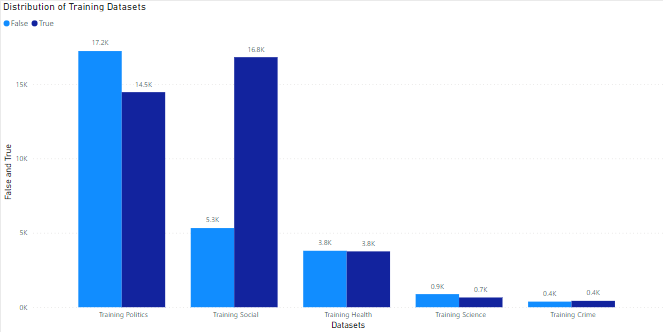
We studied imbalanced with 2 baselines when fine-tuning our model, we started with no domains to get a baseline of the model and noticed that the model was guessing 0 almost every time. We figured that it might be guessing 0 as the training data had about double the amount of 0s as there were 1s so we attempted to fix this imbalance by down sampling the dataset using pandas.DataFrame.sample() with replace=False and a fixed random seed 42 to ensure reproducibility. We down-sampled the data by removing samples from the majority class 0 until the dataset was balanced. We removed these samples randomly but with a fixed random seed. The resulting dataset had 57,326 samples with 28,663 true and false samples. Although we balanced the dataset the model still predicted mostly false which could be because the model was guessing the same probability for 1 and 0 and selecting 0 as the default which would make it so that the dataset was not causing the issue. We kept the balanced dataset for the rest of the testing just to make sure.

## Error Analysis

When working on this project we had many errors with the experiment that led to the misclassification of data. When we ran our first attempts at fine-tuning the model, the model guessed 0 for almost everything we tried to isolate hyperparameters as well as variables to determine where the error was coming from. We started by looking at how the model found similarities and correlations between the true parameter as well as the text to find out if there was a reason that the model was guessing all 0. We thought that maybe there was not enough text fed to the model with the 64 tokens we had selected. We changed the token length for text to 256 as this was as high as we could go without the model either running out of memory or taking longer than 2 hours per epoch. When we changed this value, we found out that this was not what caused the error as the model still guessed all 0. The next thing we tried was to find an optimizer that was better suited for large tasks as we thought maybe the default optimizer was not good enough, we found Lion which was better suited for large models and got a slightly better result than with Adamw but still most of the predicted labels were false. Next, we looked at the way we were evaluating the model as the method we used was better suited for text generation rather than classification of text. We had used FastLanguageModel.for\_inference(model), so we changed the evaluation to use logits to get the model output then used SoftMax to convert them to probabilities for each class and lastly with argmax picked the class that had the highest probability as the predicted label. We decided to use this evaluation method to simplify the class selection process and make it more efficient for classification tasks. With the new evaluation method, we did manage to get a better f1\_score, but it was still unsatisfactory and still guessing mostly 0. Since the evaluation method did not fix the problem with guessing 0, we decided to balance the data to make sure that the model did not train preferring 0 as it had more rows. We decided to use down-sampling as we had a huge dataset thus losing some data would not be critical. With the balanced dataset the model still predicted mostly 0. We took another look at the evaluation to find out why it could possibly predict 0 most of the time and we found that it is possible that if the model does not enough data it may simply be giving the same probability for 0 and 1 and thus picks 0 as they are equal, this made us decide to run more epochs on the model and see if that would change anything. For 3-8b the model did improve and guess truer with more epochs which makes us think if we can run more epochs, it would improve but we were running out of time. We tried increasing the learning rate to 5e-4 to maybe converge faster and require fewer epochs, but this made the model unstable and made it predict only 0 so we reverted the learning rate back. Then we ran the model on 2e-4 which is what we had before to see if we could improve it enough with the limited time. We ran the model but since the model and dataset are complex each run, and the evaluation cycle takes 3 hours per epoch, so we decided to train 3 epochs at a time and then evaluate which took 4 hours to do approximately.

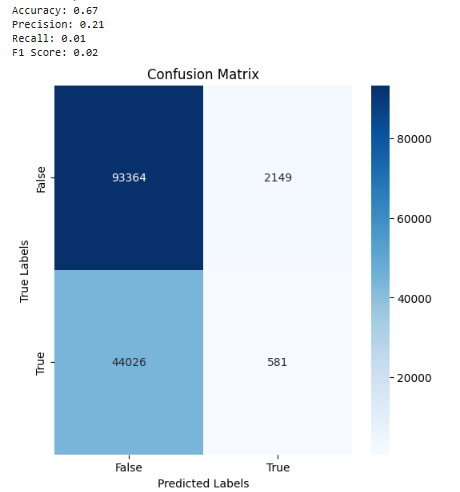
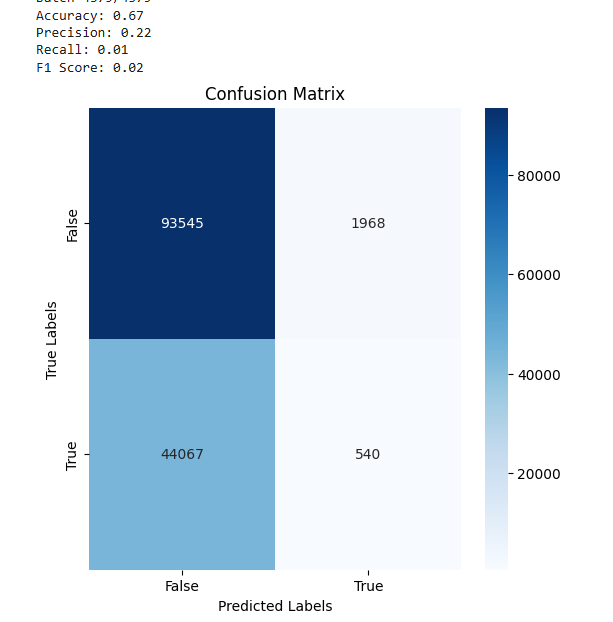
# Performance Evaluation & Results

*Chart: Distribution of True and False labels for each of the Testing Datasets*

*Chart: Distribution of True and False labels for each of the Training Datasets*

Given the distributions of each of the datasets, the training data was very biased towards false and thus needed to be balanced.

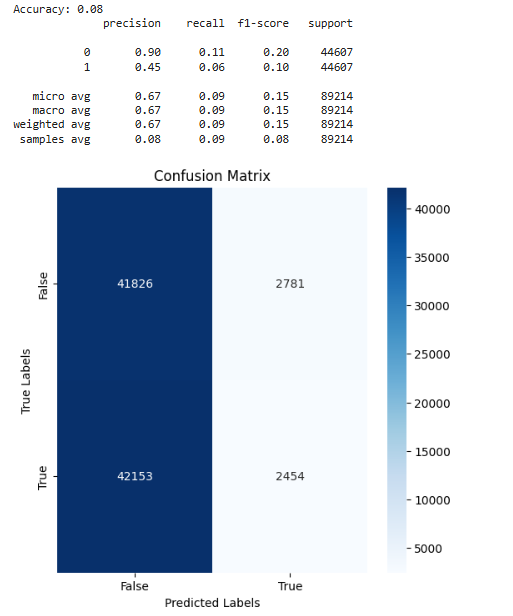
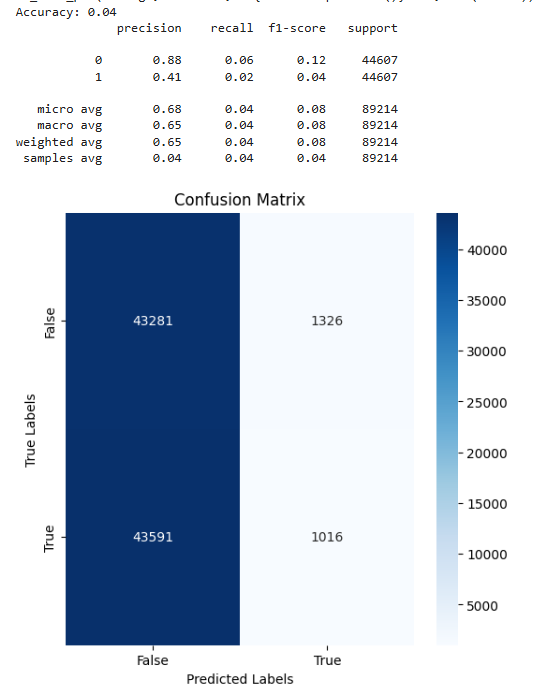
## LLaMA 2-7b



*Untrained Model Confusion Matrix* *Best Trained Confusion Matrix: Epochs = 2*

There was no change with changing this model under our original evaluation method using FastLanguageModel.for\_inference(model) and it did not compute with the new evaluation method using logits, so we do not have any quantitative results running 2.7-b after the change in evaluation methods.

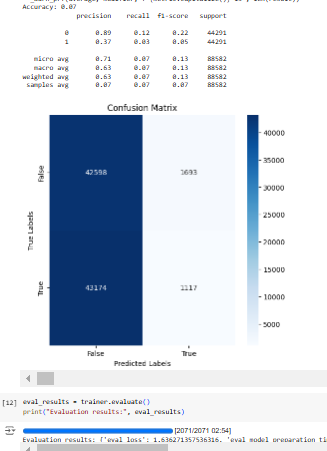
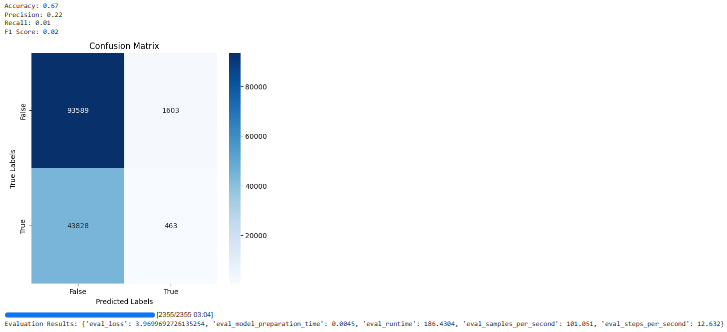
## LLaMA 3.1-8b



*Untrained Model Confusion Matrix* *Best Trained Confusion Matrix: Epochs = 6*

We originally were only training a large model from LLaMA 2 and a smaller model from LLaMA 3.2. After having issues running LLaMA2-7B we decided to run 3.1-8B with our newer evaluation method. We could only do a maximum of 6 epochs for training as the cost to run this model in both time and money was too much to continue further with our deadline approaching. However in our comparisons, this would be the best model to continue to train as it has had the best results from learning out of the three.

## LLaMA 3.2-1b



*Untrained Model Confusion Matrix*  *Best Trained Confusion Matrix: Epochs = 6*

This was by far the quickest and easiest to run though as it has less trainable parameters it did not have great improvements running the same number of epochs. A lot of our troubleshooting and testing was done using this model to avoid lengthy time and cost constraints, but that did not translate to the results.

For further information regarding the results of this experiment, please reference the .xlsx file located in the project’s associated GitHub repository.

# Conclusion and Future Work

The results of our experiments highlight the difficulties that come with fine-tuning LLM models such as Llama with limited hardware and computational resources. With our smaller models, such as Llama 3.2 1b and Llama 2 7b, although we were able to iterate comparatively faster than Llama 3.1 8b through the datasets, the lower memory requirements of these models consistently produced undesirable results, even with an increase in epochs used. The fact that no improvement was seen in these lower-parameter models for our news predicting task suggests that parameter capacity is a very important factor when it comes to effectively learning from the complex data it is provided.

Conversely, although Llama 3.1 8b didn’t produce any consistently good predictions, it demonstrated promise, with noticeable differences in accuracy and F1 score as it was trained on more epochs. Additionally, although all models showed a strong bias towards predicting the false class, as epochs increased, Llama 3.1 8b showed a higher willingness to predict true. Due to the substantial computational cost, both in terms of time and hardware, we were only able to train the 8 billion parameter model for 6 epochs, limiting our ability to fully evaluate the model’s potential. Despite showing promise for improvement given more epochs, the results we obtained were still far from good, suggesting that even larger models (parameter-wise) and additional epochs will be necessary for meaningful improvements.

Any future iteration of this project would benefit from access to better hardware, enabling training on higher-parameter models for more epochs, such as Llama 13b, or even 70b. The evidence discovered during our research suggests that larger models with high epoch counts would yield significantly better results.

# References

* [1] F. K. A. Salem, R. A. Feel, S. Elbassuoni, M. Jaber, and M. Farah, “FA-KES: A Fake News Dataset around the Syrian War,” *Proceedings of the International AAAI Conference on Web and* *Social Media*, vol. 13, pp. 573–582, Jul. 2019, doi: <https://doi.org/10.1609/icwsm.v13i01.3254>.
* [2] “Fake News Detection,” *fakenews.research.sfu.ca*. <http://fakenews.research.sfu.ca/>
* [3] J. A. Saenz, S. R. K. Gopal, and D. Shukla, “Covid-19 Fake News Infodemic Research Dataset (CoVID19-FNIR Dataset),” *IEEE Dataport*, Jun. 12, 2021. <https://ieee-dataport.org/open-access/covid-19-fake-news-infodemic-research-dataset-covid19-fnir-dataset> (accessed Dec. 18, 2024).
* [4] Abhishek Koirala, “COVID-19 Fake News Dataset,” vol. 1, Feb. 2021, doi: <https://doi.org/10.17632/zwfdmp5syg.1>.
* [5] I. Agarwal and D. Rana, “COVIFN : Fake News on COVID19,” *IEEE Dataport*, Nov. 03, 2021. <https://ieee-dataport.org/documents/covifn-fake-news-covid19>
* [6] “Fake News,” *kaggle.com*. <https://www.kaggle.com/competitions/fake-news/data>
* [7] “Fake News Detection Datasets | ISOT research lab,” *Uvic.ca*, 2022. <https://onlineacademiccommunity.uvic.ca/isot/2022/11/27/fake-news-detection-datasets/>
* [8] “liar · Datasets at Hugging Face,” *huggingface.co*. <https://huggingface.co/datasets/liar>
* [9] “PHEME dataset for Rumour Detection and Veracity Classification,” *figshare.com*, Jun. 2018, doi: <https://doi.org/10.6084/m9.figshare.6392078.v1>.
* [10] R. Misra, “Politifact Fact Check Dataset,” *Kaggle.com*, 2022, doi: <https://doi.org/10.13140/RG.2.2.29923.22566>.
* [11] “climate\_fever · Datasets at Hugging Face,” *huggingface.co*. <https://huggingface.co/datasets/climate_fever>
* [12] KaiDMML, “FakeNewsNet/dataset at master · KaiDMML/FakeNewsNet,” *GitHub*, 2017. <https://github.com/KaiDMML/FakeNewsNet/blob/master/dataset> (accessed Dec. 18, 2024).
* [13] Harsh Vardhan, “Fine-Tuning Llama 2 Using LoRA and QLoRA: A Comprehensive Guide,” *Medium*, Jun. 18, 2024. [https://medium.com/@harsh.vardhan7695/fine-tuning-llama-2-using-lora-and-qlora-a-comprehensive-guide-fd2260f0aa5f](mailto:https://medium.com/@harsh.vardhan7695/fine-tuning-llama-2-using-lora-and-qlora-a-comprehensive-guide-fd2260f0aa5f) (accessed Dec. 19, 2024).
* [14] “Llama 3.2,” *Meta Llama*, 2022. <https://www.llama.com/>
* [15] Hugging Face, “Hugging Face – On a mission to solve NLP, one commit at a time.,” *huggingface.co*, 2024. <https://huggingface.co/>