Fine-tuning Llama model for lie detection task on a Memory dataset

Caltran Lorenzo[1], Cimbro Letizia[1], Skurativska Kateryna[1] and Jahanianarange Nahid [1]

Abstract: In recent years, natural language processing (NLP) models have shown remarkable success in various tasks, ranging from language translation to sentiment analysis. In this study, we explore the application of the advanced Llama model to the challenging domain of lie detection, focusing specifically on a Memory dataset. The Llama model, known for its robustness and versatility, offers a unique foundation for addressing nuanced linguistic patterns associated with deceptive communication. Our work involves fine-tuning the Llama model to effectively discern deceptive language patterns within the context of memories and personal narratives. The Memory dataset, carefully curated for this study, provides a diverse collection of narratives that challenge the model to discern subtle cues indicative of deceptive statements. Through an iterative fine-tuning process, we enhance the Llama model's ability to capture and interpret the intricacies of language use in the context of recalling memories. The experimental results demonstrate the model's improved performance in distinguishing truthful narratives from deceptive ones on the Memory dataset. This research contributes to the growing body of knowledge in utilizing advanced NLP models for lie detection tasks, particularly in the unique context of memory recall. The findings have implications for applications in forensic analysis, security screening, and other domains where accurate detection of deceptive language is crucial.

Keywords: Lie-detection, LLama, Memory dataset, Fine-tuning

1 Introduction

Deception detection, a longstanding challenge in diverse domains such as forensic investigations, security screening, and legal proceedings, has witnessed substantial advancements with the advent of natural language processing (NLP) models. The intricacies of distinguishing truth from deception become particularly pronounced in the context of narratives related to personal memories, where linguistic nuances play a pivotal role. In this study, we address this challenge by delving into the fine-tuning of the Llama model, a state-of-the-art NLP architecture, specifically tailored for the task of lie detection within the framework of memory recall.

Historically, lie detection in verbal communication has relied on linguistic cues and behavioral patterns. With the proliferation of digital platforms and the consequential surge in textual data, there arises a pressing need for automated tools that can discern deceptive language in written narratives, especially those entwined with personal memories. The

1 Mathematical department, University of Padova, Padova, Italy, lorenzo.caltran@studenti.unipd.it; letizia.cimbro@studenti.unipd.it; kateryna.skurativska@studenti.unipd.it; Nahid.jahanianarange@studenti.unipd.it

Llama model, renowned for its adaptability and performance across a spectrum of NLP tasks, emerges as a promising candidate to navigate the complexities inherent in memory-related deception.

This investigation unfolds against the backdrop of existing literature, which underscores the challenges associated with lie detection in natural language, particularly when contextualized within personal recollections. The gap in current methodologies becomes apparent when applied to the unique characteristics of memory-related narratives, necessitating a specialized approach.

Our primary objective is to augment the Llama model's capability to scrutinize and interpret linguistic intricacies intrinsic to deceptive language within the realm of memory recall. The Memory dataset, meticulously curated for this study, serves as the testing ground for evaluating the efficacy of the fine-tuned Llama model. Through a systematic exploration of the model's performance, we aim to contribute empirical insights that extend the frontier of automated lie detection to the nuanced domain of memory-related narratives.

In this introduction, we delineate the motivation behind our research, articulate the challenges associated with lie detection in memory-related contexts, and position the Llama model as a pivotal instrument in addressing these challenges. The subsequent sections elucidate the methodology, experimental findings, and implications of our endeavor, culminating in a comprehensive exploration of the role of NLP in advancing deception detection within the intricate fabric of personal memories.

2 Literature review

In this section, we will shortly review the paper presenting Llama 2, a collection of pretrained and fine-tuned Large Language Models [To23].

Llama 2 models have been released to the general public both for research and commercial use. Particularly, Llama 2, an updated version of Llama 1, has been released and was trained on a new mix of publicly available data. The size of the pretraining corpus was increased with respect to Llama 1 by 40%, and it was also doubled the context length of the model, and adopted grouped-query attention. Llama 2-Chat has also been released, a fine-tuned version of Llama 2 that is optimized for dialogue use cases. Both Llama 2 and Llama 2-Chat have been released with variants with 7B, 13B, and 70B parameters.

In the paper, pretraining details and choices are explained and motivated, but we will now focus on how the fine tuning of Llama 2-Chat was done. The fine-tuning process described in the document involves two main stages: Supervised Fine-Tuning (SFT) and Reinforcement Learning with Human Feedback (RLHF).

Supervised Fine-Tuning (SFT) is initiated with publicly available instruction tuning data to bootstrap the process. A focus is placed on collecting high-quality SFT data, leading to

the collection of several thousand examples of such data. This is done to achieve a high level of quality. To ensure the quality of the collected data, a thorough examination is conducted on a set of 180 examples, comparing human annotations with those generated by the model. Surprisingly, the model-generated outputs were found to be competitive with human annotations, suggesting that more effort can be devoted to preference-based annotation for RLHF. The fine-tuning details include the use of a cosine learning rate schedule with specific parameters, such as an initial learning rate of 2x0.00001, a weight decay of 0.1, a batch size of 64, and a sequence length of 4096 tokens. The training samples consist of prompts and answers, with a special token used to separate the prompt and answer segments. An autoregressive objective is utilized, and the model is fine-tuned for 2 epochs.

Reinforcement Learning with Human Feedback (RLHF) is a model training procedure applied to a fine-tuned language model to further align the model with human preferences. It involves integrating safety in the general RLHF pipeline, which includes training a safety-specific reward model and gathering more challenging adversarial prompts for rejection sampling style fine-tuning. The process is refined with context distillation, which generates safer model responses by prefixing a prompt with a safety preprompt and then fine-tuning the model on the safer responses without the preprompt.

The safety of Llama 2 models is ensured through a series of measures. The developers have conducted safety testing and tuning tailored to specific applications of the model. They have employed safety-specific data annotation and tuning, as well as conducted red-teaming and iterative evaluations to increase the safety of the models. Additionally, the release of Llama 2 includes a Responsible Use Guide and code examples to facilitate the safe deployment of the models. The developers acknowledge the potential risks associated with the use of Llama 2 and emphasize the importance of performing safety testing and tuning tailored to specific applications before deployment. It is mentioned that the safety evaluations are performed using content standards biased towards the Llama 2-Chat models, and that the safety tuning may sometimes result in an overly cautious approach. The developers have also made efforts to tune the models to avoid topics like misinformation, bioterrorism, and cybercrime.

Finally, these Llama 2 models have demonstrated their competitiveness with existing open-source chat models, as well as competency that is equivalent to some proprietary models on evaluation sets examined, although they still lag behind other models like GPT-4.

3 Methodology

[Ma]

To apply Fine Tuning we used Google Colab for Python programming, as a first step, we installed the necessary libraries, which are:

- accelerate: a distributed training library for PyTorch by HuggingFace. It allows you to train your models on multiple GPUs or CPUs in parallel (distributed configurations), which can significantly speed up training;
- PEFT: a Python library by HuggingFace for efficient adaptation of pre-trained language models (PLMs) to various downstream applications without fine-tuning all the model's parameters. PEFT methods only fine-tune a small number of (extra) model parameters, thereby greatly decreasing the computational and storage costs.
- bitsandbytes: it's a library by Tim Dettmers and it is a lightweight wrapper around CUDA custom functions, in particular 8-bit optimizers, matrix multiplication (LLM.int8()), and quantization functions. It allows to run models stored in 4-bit precision: while 4-bit bitsandbytes stores weights in 4-bits, the computation still happens in 16 or 32-bit and here any combination can be chosen (float16, bfloat16, float32, and so on).
- transformers: a Python library for natural language processing (NLP). It provides a number of pre-trained models for NLP tasks such as text classification, question answering, and machine translation.
- trl: a full stack library by HuggingFace providing a set of tools to train transformer language models with Reinforcement Learning, from the Supervised Fine-tuning step (SFT), Reward Modeling step (RM) to the Proximal Policy Optimization (PPO) step.

As a second step we loaded a pre-processed version of the memory dataset, which columns are the story and the label, which is a categorical variable that states whether the story is imagined, recalled or retold. We then removed all the NA values and the retold stories, the reason we did that is because we are focusing on lie detection and having a story labelled as 'retold' doesn't state anything about the truthfulness of it.

After doing that, we split the dataset into training test sets, where 90% of the original data is going to be used as training, while the remaining 10% as test. The split is stratified by label, so that each set contains a representative sample of imagined and recalled story. We then shuffled the training data in order for the model to learn from a more representative data sample in each batch.

The following step is to transform the texts contained in the train and test data into prompts to be used by Llama. For both the training and test set the prompt contains specific instructions about what our goal is: to categorize each story as either 'imagined' or 'recalled', aside from the that, the train set prompt also contains the story that we want to categorize and the assigned label, while the test prompt only contains the story.

The next thing to do is to create a function that allows us to evaluate our model. This function maps the labels to a numerical representation, where 0 represents 'imagined' and 1 represents 'recalled' or 'none', where 'none' represents when the model is not able to do an assignment. Aside from doing that, it calculates the accuracy on the test data for each label,

creates an accuracy report calculating metrics like precision, recall, f1 score and support and also generates the confusion matrix.

Next thing to do is to load the model, which is NousResearch's Llama2 chat-hf model with 7B parameters, which is the same as the model from HuggingFace but more easily accessible. Then the code gets the float16 data type from the torch library, which is the data type that will be used for the computations. Next we created a BitsAndBytesConfig object with the following settings:

- $load\_in\_4bit$: Loads the model weights in 4-bit format.
- $bnb\_4bit\_quant\_type$: Uses the "nf4"quantization type. 4-bit NormalFloat (NF4), is a new data type that is information theoretically optimal for normally distributed weights.
- $bnb\_4bit\_compute\_dtype$: Uses the float16 data type for computations.
- $bnb\_4bit\_use\_double\_quant$: Avoids using double quantization (reduces the average memory footprint by quantizing also the quantization constants and saves an additional 0.4 bits per parameter).

After that we created a AutoModelForCausalLM object from the pre-trained Llama-2 language model, using the BitsAndBytesConfig object for quantization, we disabled caching for the model and we set the pre-training token probability to 1.

The next step is to load the tokenizer for the Llama-2 language model and set the padding token to be the end-of-sequence (EOS) token. Finally, the code sets the padding side to be "right", which means that the input sequences will be padded on the right side.

Once all of this is done, we define a function for predicting the label of a given story using the Llama-2 language model. The function takes three arguments:

- test: A Pandas DataFrame containing the news headlines to be predicted
- model: The pre-trained Llama-2 language model.
- tokenizer: The tokenizer for the Llama-2 language model.

For each news headline in the test DataFrame, our function creates a prompt for the language model, which asks it to analyze the story and return the corresponding label. After that it uses the pipeline function from the Hugging Face Transformers library to generate text from the language model, using the prompt. The generated text will correspond to the predicted label.

The last thing to do is to configure and initialize a Simple Fine-tuning Trainer (SFTTrainer) for training a large language model using the Parameter-Efficient Fine-Tuning (PEFT) method, which should save time as it operates on a reduced number of parameters compared to the model's overall size. The PEFT method focuses on refining a limited set of (additional) model parameters, while keeping the majority of the pre-trained LLM parameters fixed. This

significantly reduces both computational and storage expenses. Additionally, this strategy addresses the challenge of catastrophic forgetting, which often occurs during the complete fine-tuning of LLMs.

The $peft\_config$ object specifies the parameters for PEFT. The following are some of the most important parameters:

- $lora\_alpha$: The learning rate for the LoRA update matrices.
- $lora\_dropout$: The dropout probability for the LoRA update matrices.
- r: The rank of the LoRA update matrices.
- $bias$: The type of bias to use. The possible values are none, additive, and learned.
- $task\_type$: The type of task that the model is being trained for. The possible values are $CAUSAL\_LM$ and $MASKED\_LM$.

The $training\_arguments$ object specifies the parameters for training the model. The following are some of the most important parameters:

- $output\_dir$: The directory where the training logs and checkpoints will be saved.
- num_train_epochs: The number of epochs to train the model for, which we set to 3.
- $per\_device\_train\_batch\_size$: The number of samples in each batch on each device.
- $gradient\_accumulation\_steps$: The number of batches to accumulate gradients before updating the model parameters.
- $optim$: The optimizer to use for training the model.
- $save\_steps$: The number of steps after which to save a checkpoint.
- $logging\_steps$: The number of steps after which to log the training metrics.
- $learning\_rate$: The learning rate for the optimizer.
- $weight\_decay$: The weight decay parameter for the optimizer.
- $fp16$: Whether to use 16-bit floating-point precision.
- $bf16$: Whether to use BFloat16 precision.
- max_grad_norm: The maximum gradient norm.
- max_steps: The maximum number of steps to train the model for.
- $warmup\_ratio$: The proportion of the training steps to use for warming up the learning rate.
- $group\_by\_length$: Whether to group the training samples by length.
- $lr\_scheduler\_type$: The type of learning rate scheduler to use.
- $report\_to$: The tools to report the training metrics to.

- $evaluation\_strategy$: The strategy for evaluating the model during training.

The SFTTrainer is a custom trainer class from the PEFT library. It is used to train large language models using the PEFT method.

The SFTTrainer object is initialized with the following arguments:

- $model$: The model to be trained.
- $train\_dataset$: The training dataset.
- $eval\_dataset$: The evaluation dataset.
- $peft\_config$: The PEFT configuration.
- $dataset\_text\_field$: The name of the text field in the dataset.
- $tokenizer$: The tokenizer to use.
- $args$: The training arguments.
- $packing$: Whether to pack the training samples.
- max_seq_length: The maximum sequence length.

Once the SFTTrainer object is initialized, it can be used to train the model by calling the train() method.

4 Experimental Setup

- **Purpose**: This study presents an exploration of the performance of the Llama-2-7b-chat-hf model in a lie-detection classification task, specifically focusing on Scenario 1 with only autobiographical memory datasets. Previous research has shown that human accuracy in detecting deception rarely exceeds chance levels, prompting the development of automated verbal lie detection techniques leveraging Machine Learning and Transformer models to achieve higher accuracy levels.
  In this study, we investigated the effectiveness of the Llama-2-7b-chat-hf model, evaluating both its small and base sizes, across English-language dataset autobiographical memories. Stylometric analysis was initially conducted to delineate linguistic distinctions among the datasets. Subsequently, we employed Scenario 1, where both the training and test sets originated solely from the autobiographical memory dataset.
- **Dataset**: The dataset used in this study comprises stories collected from crowdworkers through three phases, namely Hippocorpus. The data collection process was approved by the Microsoft Research Institutional Review Board (IRB), and explicit informed consent was obtained from all participants.
- **Experimental Design**: The experimental design involves preprocessing a dataset containing stories labeled as ïmaginedör "recalled". The data is split into training, testing, and evaluation sets. A conversational language model is fine-tuned for the

classification task using prompts generated from the stories. The model is evaluated using various metrics to assess its performance in distinguishing between imagined and recalled stories.

- **Evaluation Metrics**: The evaluation metrics include accuracy, classification report, and confusion matrix. These metrics provide insights into the model's overall accuracy, precision, recall, F1-score, and confusion patterns, enabling a comprehensive assessment of its performance on the classification task.
- **Hardware and Software Setup**: The code utilizes Python libraries such as numpy, pandas, tqdm, and transformers. It also relies on external packages like BitsAndBytes (bnb), trl, and datasets. The training and evaluation processes likely require access to GPUs or TPUs for efficient computation due to the large-scale language model used.
- **Testing Procedure**: The testing procedure involves using a text generation pipeline with the trained model to predict the labels of the test dataset. Post-processing is applied to interpret the generated text and assign labels. The predicted labels are then evaluated against the ground truth labels using evaluation metrics such as accuracy and classification report.
- **Training Procedure**: The training procedure involves using the Paged AdamW optimizer with a cosine learning rate scheduler. Mixed precision training with FP16 and gradient checkpointing is employed to optimize memory usage and accelerate training. The model is trained for three epochs with a per-device batch size of 1 and gradient accumulation steps.

5 Results

Before applying Fine Tuning we tried to test the original Llama2 model and we have observed that the model doesn't understand the task, as it always returns the label 'imagined' or it doesn't return any label. To test our results, we repeated the process of training and fine-tuning 3 times (3 runs), and each time the model was trained in 3 epochs. After Fine Tuning we can see a significant increase in accuracy.

Epoch	Training Loss	Validation Loss
0	1.920000	1.843634
1	1.904500	1.829148
2	1.862100	1.826096

Tab. 1: Run 1 losses

	Accuracy
Imagined	0.741
Recalled	0.729
Model	0.735

Tab. 2: Run 1 Accuracy

Epoch	Training Loss	Validation Loss
0	1.912700	1.845763
1	1.898100	1.828795
2	1.893300	1.824947

Tab. 3: Run 2 losses

	Accuracy
Imagined	0.828
Recalled	0.679
Model	0.753

Tab. 4: Run 2 Accuracy

Epoch	Training Loss	Validation Loss
0	1.907800	1.836034
1	1.884700	1.821752
2	1.870500	1.817982

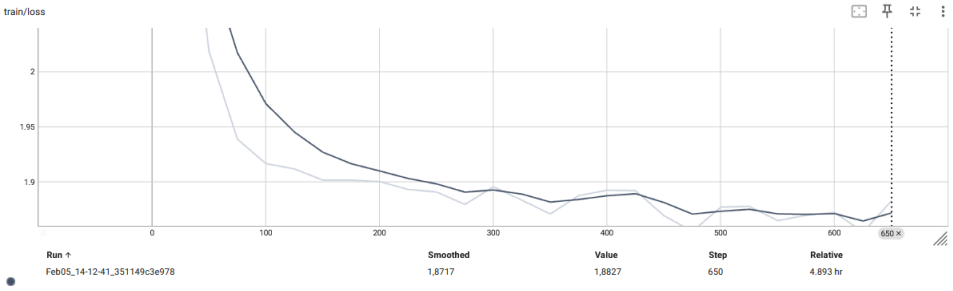
Tab. 5: Run 3 losses

	Accuracy
Imagined	0.953
Recalled	0.466
Model	0.708

Tab. 6: Run 3 Accuracy

	Accuracy
Imagined	0.841
Recalled	0.625
Model	0.733

Tab. 7: Average Accuracy



Another observation that we made is that, despite categorizing the stories with a good accuracy, the Fine Tuned model is not able to generate correctly the text label, in fact it only returns labels such as 're' and 'imag' instead of 'recalled' and 'imagined'. For this reason we had to manipulate the output in order to get the accuracy.

6 Discussion

- **Training and Validation Loss**: The training and validation loss values decrease over epochs, indicating that the model is learning and generalizing well to unseen data.
- **Accuracy**: The accuracy values for the imagined and recalled classes vary across different runs. While the imagined class tends to have higher accuracy values compared to the recalled class, there is variability in the model's performance across runs.
- **Model Performance**: The overall model accuracy, calculated as the average of the imagined and recalled class accuracies, also varies across runs.

7 Implications and Applications

The study's findings hold significance for various fields, including natural language processing, cognitive science, and psychology. Understanding the mechanisms underlying how humans recall and imagine events has important implications for memory research, cognitive modeling, and therapeutic interventions. The ability to automatically classify narrative text into distinct cognitive processes could facilitate advancements in psychological assessment, storytelling analysis, and virtual reality applications.

8 Challenges and Future Work

Because of the limitation on computational resources, our work has been developed only with the Llama 2-Chat model with 7B parameters. Future work may include increasing said computational resources to also work with the bigger-sized models (13B and 70B). It could

also be interesting to confront our results with those obtained after the fine-tuning of different LLMs on the same dataset. Future research could explore ensemble approaches or hybrid models to improve classification accuracy and robustness. Investigating additional features or contextual information, such as narrative structure or emotional content, may enhance the model's ability to discern between different types of storytelling. Further experimentation with data augmentation techniques, fine-tuning strategies, and model architectures could lead to more nuanced understanding and classification of cognitive processes reflected in textual narratives.

9 Conclusions

- **Model Performance and Insights**: The trained model demonstrated promising performance in distinguishing between ïmaginedänd "recalledßtories, as evidenced by high accuracy and balanced precision-recall metrics. The classification report and confusion matrix provided detailed insights into the model's ability to correctly classify instances from each class and highlighted areas where the model may have struggled.
- **Strengths and Limitations:** The experimental results underscore the effectiveness of leveraging large-scale language models, such as "NousResearch/Llama-2-7b-chat-hf", for text classification tasks. The model's ability to understand and interpret complex language structures contributed to its success in differentiating between imagined and recalled stories. However, limitations such as class imbalances or ambiguous instances may have posed challenges to the model's performance and warrant further investigation.
- **Conclusion**:
  In conclusion, the study highlights the potential of advanced language models to uncover and classify cognitive processes embedded in textual narratives. The findings contribute to our understanding of human cognition and memory, while also offering practical applications in various domains. By leveraging cutting-edge technologies and interdisciplinary approaches, researchers can continue to advance our knowledge of storytelling, memory recall, and cognitive functioning in both theoretical and practical contexts.

10 References

[Ma] Massaron, L.: Fine-tune Llama 2 for sentiment analysis, URL: https://www.kaggle.com/code/lucamassaron/fine-tune-llama-2-for-sentiment-analysis.

[To23] Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; Bikel, D.; Blecher, L.; Ferrer, C. C.; Chen, M.; Cucurull, G.; Esiobu, D.; Fernandes, J.; Fu, J.; Fu, W.; Fuller, B.; Gao, C.; Goswami, V.; Goyal, N.; Hartshorn, A.; Hosseini, S.; Hou, R.; Inan, H.; Kardas, M.; Kerkez, V.; Khabsa, M.; Kloumann, I.; Korenev, A.; Koura, P. S.; Lachaux, M.-A.; Lavril, T.; Lee, J.; Liskovich, D.; Lu, Y.; Mao, Y.; Martinet, X.; Mihaylov, T.; Mishra, P.; Molybog, I.; Nie, Y.; Poulton, A.; Reizenstein, J.; Rungta, R.; Saladi, K.; Schelten, A.; Silva, R.; Smith, E. M.; Subramanian, R.; Tan, X. E.; Tang, B.; Taylor, R.; Williams, A.; Kuan, J. X.; Xu, P.; Yan, Z.; Zarov, I.; Zhang, Y.; Fan, A.; Kambadur, M.; Narang, S.; Rodriguez, A.; Stojnic, R.; Edunov, S.; Scialom, T.: Llama 2: Open Foundation and Fine-Tuned Chat Models, 2023, arXiv: 2307.09288 [cs.CL].