NLP generated Fake Articles

SUDHANG SHANKAR, PAULA NAUTA*, Institute of interactive Systems and Data Science, Austria

Abstract: We have developed four language models to generate newspaper articles using a variety of Natural Language Processing techniques. These range from a simple n-gram model to fine-tuned Large Language Models (LLMs) like LLaMa2-7B, Falcon-7B and GPT-Neo. The output of these models has been evaluated using both automated metrics (BLEU) and a human evaluation in the form of a "guessing game". It was found that the fine-tuned 7B LLMs outperformed the other models both in human evaluation. However, in automated evaluation, we find that n-grams actually wins out, because of the nature of the BLEU evaluation metric.

CCS Concepts: • **Computing methodologies** → **Natural language generation**.

ACM Reference Format:
Sudhang Shankar, Paula Nauta. 2023. NLP generated Fake Articles. In . ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The field of Natural Language Generation has recently become very prominent, due to a variety of powerful offerings. Apart from the release of ChatGPT in late 2022, we also have had a slew of very high quality open- and mostly-open source models that have been released to the public. The release of these models has led to a slew of downstream research that has led to a lot of speculation and excitement about the field.

As part of the Lecture Natural Language Processing (706.230) at the TU Graz, the authors participated in a project to produce 50 Newspaper Articles, using a variety of methods:

- `n-grams`: The traditional NLP method of generating text by creating a frequency distribution of (word) n-grams
- `Own Source Code`: For this, the authors used two recently-published open-source LLMs (LLaMa2 and Falcon), which were fine-tuned using recently established techniques for resource-constrained fine-tuning.
- `GPT-X`: A GPT-Neo 1.3B model was fine-tuned without any optimization techniques.

We trained these models on a dataset of newspaper articles and evaluated them using both automated metrics (BLEU score) and a human evaluation (a Turing Test)

2 RECENT DEVELOPMENTS

For our project, we have relied on a few recent developments in the field of Natural Language Processing:

*Both authors contributed equally to this research.

Manuscript submitted to ACM

2.1 Fine-Tuning

Radfort et. al [7] from OpenAI popularized the concept of fine-tuning LLMs. They first *pre-train* a transformer model on a diverse corpus, and then transfer the model for a specific task (news article generation, in our case) by updating the weights of the model for this task, a process they call *discriminative fine-tuning*

2.2 LoRA

It is very resource intensive to to fine-tune LLMs with billions of parameters for specific tasks, and cannot easily be done on commodity hardware. To make fine-tuning on commodity hardware easier [5] introduced the Low-Rank Adaptation (LoRA) technique which works by freezing the pre-trained model weights and injecting trainable adapter matrices into the layers of the Transformer models.

The intuition behind LoRA is that updating the weights of a pre-trained LLM has a low "intrinsic rank", since the fine-tuning datasets are much smaller [9]. For example, our dataset is only 430KB, compared to the multiple terables of pre-training data for LLaMa [11]

The original LLM can be expressed as:

$$output = W_0 x + b_0 \tag{1}$$

where x is the input, W_0 is the weight matrix and b_0 is the bias term of the (frozen) LLM. Let's assume that W_0 is of shape nxn In LoRA, the equation becomes:

$$output = W_0 x + b_0 + BAx \tag{2}$$

where A and B are rank-decomposition matrices of shape $n \times rank$ and $rank \times n$ respectively. Thus, during the fine-tuning process, we update only A and B. We choose the $rank$ fairly low (8 in our case for LLaMa and Falcon). We thus, drastically reduce the amount of parameters to train to $2n \times rank$

2.3 QLoRA

Despite LoRA, one problem that remains is that the base models are too big to fit in commodity hardware like a Google Colab GPU.

Dettmers et al. [1] introduced QLoRA, which quantizes the LLM to use lower precision floating points and then applying LoRA. With this, we were able to fine-tune quantized LLaMa and Falcon LLMs with a single 16GB T4 GPU on google colab, even though their original size is much bigger than that.

The intuition behind QLoRA is that the loss in floating point precision may lead to less accuracy, but for most tasks it performs well enough for it not to be a problem. A metaphor would be: one can figure out that a JPG of a cat in very low resolution is still recognizably a cat, even though we don't have the same level of precision.

In fact, the quantization step might actually be beneficial, since the coarser precision would implicitly reduce overfitting during the fine-tuning process [8].

2.4 Open Source Models

Over the past couple of months, a slew of LLMs have been released to the public under (mostly) open licenses, leading to much innovation by enthusiasts. Concerns have also been raised regarding safety and ethics of open-source LLMs. On the other hand, Meta argue [12] that "the open release of LLMs, when done safely, will be a net benefit to society".

In this project, we have leveraged three open source models:

- `LLaMa2`: Meta introduced the LLaMa and LLaMa2 models in 2023, touting the fact that it had been trained exclusively using publicly available datasets unlike other models [11]. We use the `LLaMa2-7B` model in this project.
- `Falcon`: This model comes from the Technology Innovation Institute in the UAE who claim [6] to achieve high quality because their training dataset (called RefinedWeb) is built entirely from "properly filtered and deduplicated" data from the web, rather than curated sources. We use the `Falcon-7B` model.
- `GPT-Neo`: This model from EleutherAI attempts to replicate the GPT-3 architecture, by training on a custom dataset called The Pile, [4]. We use the `GPT-Neo-1.3B` variant.

3 DATA

Our project relies on the dataset `10,700 articles from the front page of the Times`, (hereafter called the **NYT Dataset**) for training the language models. This dataset consists of articles from the front pages of the print and web verions of the New York Times from the end of 2017 to mid-late 2018[10].

Table 1. Characteristics of dataset

Characteristic	Value
Article Count	10732
Avg. Word Count	1406.5
Avg. Sentence Count	55.81
Avg. Character Count	8499.54

The 5 most common bigrams are: *('United', 'States'), ('White', 'House'), ('New', 'York'), ('Islamic', 'State'), ('I', 'think')*

The 5 most common trigrams are: *('New', 'York', 'Times'), ('The', 'New', 'York'), ('Affordable', 'Care', 'Act'), ('New', 'York', 'City'),, ('President', 'Barack', 'Obama')*

A sample article from the dataset (truncated to save space):

> WASHINGTON — President Trump kicked off the new year with more than a dozen Twitter posts castigating his enemies and political foes and complimenting himself. Seventeen times on his first work day of the new year in Washington, the president thumbed his thumbs at convention. The objects of his attention had a vast and seemingly disconnected range, from taunting Kim Jong-un about his nuclear prowess to flaunting his own successes. But there was more...

The dataset was ultimately split 80-20 into Train/Test datasets.

4 MODEL TRAINING AND FINE-TUNING

In this section, we will examine the four models that were developed, the pre- and post- processing steps taken. For hyperparameters, please check out the appendix 9.3.

All models were run on Google Colab. Though none of the models required more than a T4 GPU with 16GB RAM, we did use an A100 at times to speed up development.

4.1 n-grams

For the n-gram language model, we relied on the `nltk` library in python. Using this library, we used a `word_tokenizer` and constructed a Maximum Likelihood Estimator.

The intuition behind the n-gram model is that we calculate conditional probabilities of a word appearing given the context. During the training process we calculate the frequency of a certain word w appearing after $n-1$ words have been observed. This gives us the probability of the next word being w. The language model then semi-randomly chooses the next word based on the conditional probabilities calculated from the training data and the context currently observed. More formally,

$$P(w_{i+1}|w_i, ..., w_{i-n+2}) = \frac{P(w_{i+1}, w_i, ..., w_{i-n+2})}{P(w_i, ..., w_{i-n+2})} \tag{3}$$

The language model thus constructed had 84643935 ngrams after training, with ngrams of orders 1 to 6.

4.2 LLaMa2-7B and Falcon-7B

The training for both LLaMa2-7B and Falcon-7B leveraged the Huggingface ecosystem. Both were trained in a very simialr way

- The pre-trained model (`meta-llama/Llama-2-7b-hf` or `tiiuae/falcon-7b`) and its corresponding tokenizer were obtained from huggingface
- The NYT dataset was tokenized with this tokenizer.
- The `bitsandbytes` library was used to quantize the model down to 4-bit precision.
- The huggingface `peft` library was used to inject this quantized model with LoRA adapter layers.
- The adapter model was then trained on the NYT dataset and uploaded to huggingface (see appendix for the URL)

4.3 GPT-Neo 1.3B

The training for Falcon-7B leveraged the Huggingface ecosystem.

- The pre-trained model `EleutherAI/gpt-neo-1.3B` and its corresponding tokenizer were obtained from huggingface
- The NYT dataset was tokenized with this tokenizer.
- The data was then fed into the model for further training (i.e., fine tuning). Note that no optimization techniques were used.

Note: Due to some memory usage issues, we had to truncate the each article to 512 tokens for the 3 LLMs.

5 GENERATION

The generation process for all four models was as follows:

- The fine-tuned or trained model was retrieved. In case of Falcon-7B and LLaMa2-7B, the quantization configuration was applied again. The tokenizers for the models that used Huggingface was also retrieved
- An article from the original dataset was retrieved, and the first two sentences from this model were fed as a prompt to each of the models, which were then made to generate a certain amount of text.
- The generated text was fed back into the very same model to generate more text. This process was repeated until at least 51 sentences had been created. The sentence count was done using nltk.
- The text with at least 51 sentences was saved in the dataframe as a new column
- The whole process was repeated until the desired number of articles had been generated

For some of the models, we applied post-processing with regular expressions to handle common problems, for example texts ending abruptly in the middle of a sentence, double punctuation, whitespace around punctuation etc.

6 EVALUATION AND RESULTS

To evaluate our language models, we took the testing subset of the NYT dataset, and compared the generated texts to the original texts from which the prompts that generated the texts were made. This was done using an automated metric (BLEU) and a human evaluation (Guessing Game).

6.1 BLEU

BLEU works by testing for overlaps in the n-grams of the two texts. Due to the fact that we trained our data on the training set from the NYT Dataset and are now comparing n-grams against a testing set, it turns out that n-grams are performing best in this metric.

Table 2. Automated Evaluation (BLEU Score) results

Model	Avg. BLEU Score (%)	Comments
n-grams	9.89	
LLaMa2	8.55	
Falcon	4 in 5	
GPT-Neo	9.62	

6.2 Guessing Game

As pointed out in [12], human evaluation is considered the gold standard for natural language generation. In light of this we developed a Turing test in the form of a guessing game.

- For each model, 10 texts generated by that model were sampled.
- The generated text and the original article that was the source of its prompt were presented to the evaluator. They were then asked to choose which of these was the original and which was the generated one.
- The evaluators guess if correct or not was recorded for each pair of articles.
- After all evaluators have finished, Cohen's Kappa was used to calculate the inter-coder reliability.

Most of the methods fared poorly. However, the LLaMa and Falcon models, fine-tuned with QLoRA outperformed all the others.

Table 3. Human Evaluation (Guessing Game) results

Model	Evaluator1 Accuracy(%)	Evaluator2 Accuracy(%)	Avg. Accuracy (%)
n-gram	100	100	100
LLaMa2-7B	100	100	100
Falcon-7B	100	100	100
GPT-Neo-1.3B	100	100	100

The inter-rater reliability of the two evaluators was calculated, using Cohen's alpha to be: .

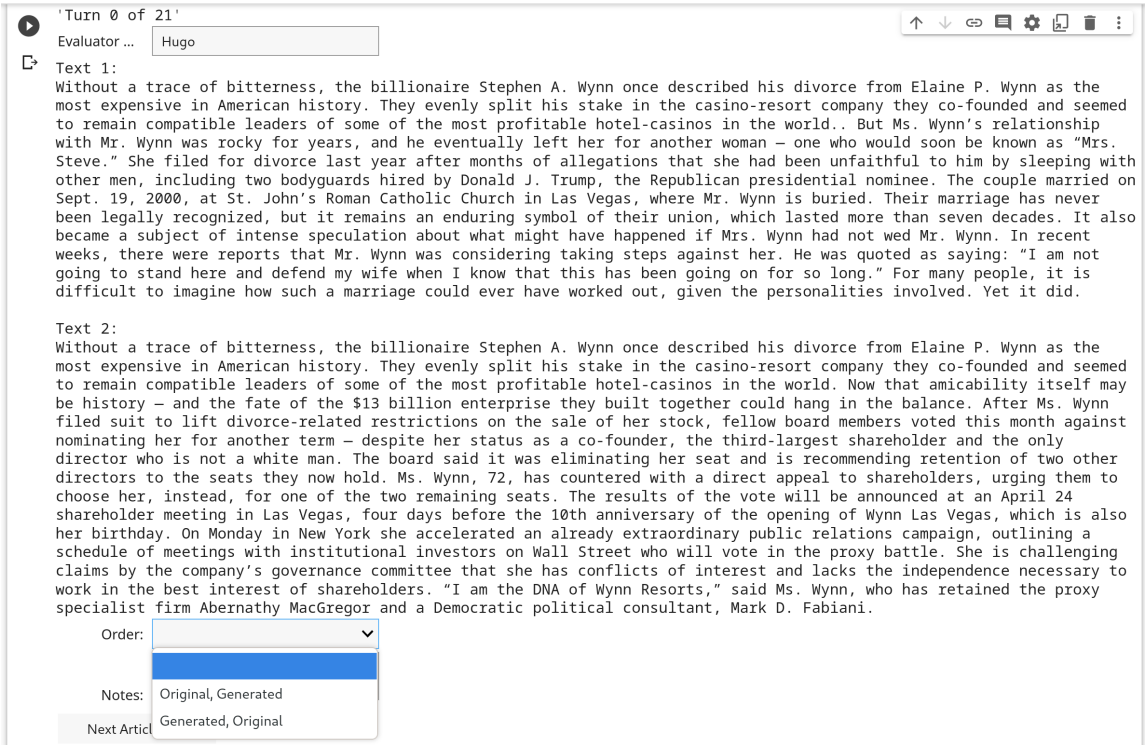

Fig. 1. Screenshot of Guessing Game

7 LIMITATIONS AND DISCUSSION

7.0.1 Truncation of Articles. For the LLMs, we truncated the articles to only 512 tokens at a time. This can lead to the models losing long term context of their articles.

7.0.2 Limitations of Human Evaluation. Although human evaluation is considered to be the gold standard for judging models for natural language generation [12], there are limitation

- Human evaluation for generative models is inherently subjective and noisy [12].
- We only show the first ten sentences of the generated and source texts to the evaluator. This may prevent some common problems like repetition from appearing in the part of the generated text that is shown to the evaluator. We have tried to minimize the amount of repetition that occurs in the generated text by specifying a repetition penalty for the LLMs. Another issue is that abrupt endings of an article would not be visible in the evaluation.
- The evaluation was not double-blind in the strictest sense, as the evaluators (being the authors) were aware of the shortcomings of the models and were able to "look out for" common issues with the generated texts.

8 CONCLUSION AND FUTURE WORK

We were able to generate articles using a variety of techniques of differing complexity. We found that automated evaluation metrics gave misleading results in comparison to human evaluation, which should be taken as a gold standard.

REFERENCES

[1] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. arXiv:2305.14314 [cs.LG]

[2] Hugging Face. 2023. How to Generate Text: Using Different Decoding Methods for Language Generation with Transformers. https://huggingface.co/blog/how-to-generate

[3] Daniel Falbel. 2023. Posit AI Blog: Understanding LoRA with a minimal example. https://blogs.rstudio.com/ai/posts/2023-06-22-understanding-lora/

[4] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *arXiv preprint arXiv:2101.00027* (2020).

[5] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685 [cs.CL]

[6] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The RefinedWeb Dataset for Falcon LLM: Outperforming Curated Corpora with Web Data, and Web Data Only. arXiv:2306.01116 [cs.CL]

[7] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving Language Understanding by Generative Pre-Training. (2018). https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf

[8] Philipp Singer. 2023. While quantization can degrade inference accuracy, it can also act as a cheap way of adding regularization to the model and even has the potential to improve downstream performance by reducing potential training overfit by reducing precision. [Tweet]. Twitter. Retrieved from https://twitter.com/ph$_s inger/status$/1681961743096901634.

[9] Keras Team. 2023. Parameter-efficient finetuning of GPT-2 with LoRA. https://keras.io/examples/nlp/parameter_efficient_finetuning_of_gpt2_with_lora/

[10] Andrew Thompson. 2019. *10,700 articles from the front page of the Times.* Retrieved from https://components.one/datasets/above-the-fold.

[11] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

[12] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

9 APPENDIX

9.1 Authors

Both authors contributed equally to this project. Specifically

9.1.1 Sudhang Shankar.

- LLaMa2, Falcon and GPT-Neo-1.3B models
- Evaluation notebooks
- Guessing Game
- Report

9.1.2 Paula Nauta.

- N-Gram model
- Exploratory Data Analysis
- Guessing Game
- Report

However, it is worth pointing out that the splitting was not always clean and we cross-collaborated throughout the project.

9.2 Online Resources

- https://github.com/sudhang/css-nlp: The github repo of this project. It contains python files and notebooks. In addition, it includes generated text files, and evaluation results.
- https://huggingface.co/sudhangshankar: Huggingface fine-tuned models

9.3 Configuration

This section describes the hyperparameters and inference parameters used for each model. Note that this doesn't include every configuration value, but the most important ones:

Table 4. Hyperparameters for n-gram MLE model

Hyperparameter	Value	Comment
n	6	We observed that there was not much regurgitation despite this relatively high value

Table 5. Hyperparameters for LLaMa2-7B model

Hyperparameter	Value	Comment
LoRA Adapter Matrix Rank	8	This is a fairly low value for a 7Billion parameter model
LoRA Alpha	32	Determines scaling factor for LoRA (a kind of "learning rate" for the LoRa Update
LoRA Dropout	0.05	5% of the neurons will be dropped to prevent overfitting
LoRA Bias	None	LoRa will only be applied to weights, not bias term
LoRA Target modules	["q_proj", "v_proj"]	Names of matrices to update using LoRA
Per Device Train Batch Size	1	
Gradient Accumulation Steps	4	
Training Max Steps	300	
Training Learning Rate	2e-4	
Optimizer	paged_adamw_8bit	Paging needed for better memory management

Table 6. Generation params for LLaMa2-7B model

Parameter	Value	Comment
do_sample	True	We want to use sampling which is better than greedy decoding or beam search [2]
top_k	50	Distribute probability mass along only these most likely k words [2])
max_length	1000	Max Token Length for LLaMa2 is 4096. However, 1000 was a better fit on a T4 with 16GB VRAM only
top_p	0.95	Distribute probability mass along only those tokens whose cumulated probability is 95% [2])
temperature	0.4	Low temperature because a lot of tokens were being generated and it would often go off on a tangent.
repetition_penalty	1.2	Had to set a somewhat high penalty in order to stop it repeating itself all the time

Table 7. Hyperparameters for Falcon-7B model

Hyperparameter	Value	Comment
LoRA Adapter Matrix Rank	8	This is a fairly low value for a 7Billion parameter model
LoRA Alpha	32	Determines scaling factor for LoRA (a kind of "learning rate" for the LoRa Update [3])
LoRA Dropout	0.05	5% of the neurons will be dropped to prevent overfitting
LoRA Bias	None	LoRa will only be applied to weights, not bias term
LoRA Target modules	["query_key_value"]	Names of matrices to update using LoRA
Per Device Train Batch Size	1	
Gradient Accumulation Steps	4	
Training Max Steps	300	
Training Learning Rate	2e-4	
Optimizer	paged_adamw_8bit	Paging needed for better memory management

Table 8. Generation params for Falcon-7B model

Parameter	Value	Comment
do_sample	True	We want to use sampling which is better than greedy decoding or beam search [2]
top_k	50	Distribute probability mass along only these most likely k words [2])
max_length	1000	Max Token Length for Falcon is 2048.
top_p	0.95	Distribute probability mass along only those tokens whose cumulated probability is 95% [2])
temperature	0.3	Low temperature because a lot of tokens were being generated and it would very often go off on a tangent.
repetition_penalty	10.0	Had to set an extremeley high penalty in order to stop it repeating itself all the time, despite a lot of prodding.

Table 9. Hyperparameters for GPT-Neo-1.3B model

Hyperparameter	Value	Comment
num_train_epochs	3	
Per Device Train Batch Size	4	
Gradient Accumulation Steps	8	
Training Max Steps	300	
Training Weight Decay	0.05	To avoid overfitting

Table 10. Generation params for GPT-Neo-1.3B model

Parameter	Value	Comment
do_sample	True	We want to use sampling which is better than greedy decoding or beam search [2]
top_k	50	Distribute probability mass along only these most likely k words [2])
max_length	1000	Max Token Length for GPT-Neo-1.3B is 2048.
top_p	0.95	Distribute probability mass along only those tokens whose cumulated probability is 95% [2])
temperature	0.4	Low temperature because a lot of tokens were being generated and it would very often go off on a tangent.
repetition_penalty	1.5	Had to set an high penalty in order to stop it repeating itself all the time.