Using Efficient Local Fine-tuning Techniques to Extract Closure Information From Long National Park Comments

Austin Lackey

May 1, 2024

Abstract

Large language models (LLMs) have become increasingly popular in recent years for their robust text processing capabilities. Despite their advantages, LLMs are often computationally expensive to run which can be a barrier to entry for many users or organizations. In this paper, I demonstrate the feasibility of fine-tuning an LLM for specific use-cases within the National Park Service (NPS) to automate the extraction of closure-related information from long comments. The key feature of my approach is the ability to perform fine-tune training and inference locally, significantly benefiting small government entities that lack the infrastructure to run large-scale models.

Figure 1: Image generated by DALL·E 3 [6]

1 Introduction

Every year, the VUStats program at The National Park Service (NPS) combs through 400+ parks worth of data to ensure that data collection is accurate and consistent. Primarily, data collectors from each park will submit monthly numbers accompanied by a comment that describes the visitation for that month. Often times, these data collectors will submit non-zero numbers for locations but mention that the location was closed in their comment. This is a problem as the comments and the numbers contradict each other. Whether it be due to human error, fat-fingering, or a lack of attention to detail, it is the VUStats program's job to catch these errors and correct them. However, this requires a lot of reading and skimming as comments can range from a few words to sometimes hundreds. Often, most of the comment contains information that is not relevant to the data collection process and can be cumbersome to read through. This is where the use of modern language models can

be of great use.

1.1 Motivation

Although LLMs are great for dealing with subjective text, any model that is robust enough to handle the nuances of human language is inherently large and computationally expensive to run. A common solution to this problem is paying for API access to a model that is hosted on a server. However, this is not feasible for the VUStats program as they are a smaller government entity and lack the resources to pay for such a service. The end-goal is simple, create a model that can take a comment as input, extract relevant closure information, and output it in a clear and concise format.

Table 1 below, showcases an example of an input comment that is long and verbose, and the output that we want the model to generate. The closure related information is highlighted in yellow for clarity. We want a model that can output concise enough information that makes it easy for the data checkers to quickly glance at and understand what was closed for the month. Any other information might be useful when a deeper dive is needed, but for the most part, we only care about the closure information.

The Goal: Extract Closure Information	
Input:	`Comment for January: Averages for the following counters were submitted: Belle Haven (vehicles and MVT) Daingerfield (vehicles and MVT) Fort Hunt Park Turkey Run Gravelly Point Note: The sign crew supervisor will check the above counters. Averages for the follwing vehicle and traffic counters were used: U.S. Marines Corps War Memorial, Great Falls Park, Lane 1(National Airport) and Lanes 2, 3,4,5 (National Airport) Note: Data Collector was notified by email of the vehicle and traffic counters.` **Claude Moore Colonial Farm - annual January closure**
Output:	**January: Claude Moore Colonial Farm was closed for the entire month**

Table 1: Input and Output Example

2 Methodology

In order to tackle this problem, we need to utilize as many efficiency techniques as possible to create a model that is feasible for local use. The following steps were taken to create the model:

1. **Data Collection:** Create an I/O domain specific-dataset that is relevant to the use-case.

2. **Model Selection:** Choose a model that is small enough to run but large enough to be useful.

3. **Quantization:** Further reduce the model size by reducing the precision and memory requirements.

4. **MLX:** Use Apple's MLX technology to take advantage of Metal's GPU acceleration and speed up inference.

5. **QLoRA Fine-tuning:** Train a subset of the model on the domain-specific dataset.

2.1 Data Collection

The data collection and preparation process is arguably the most important step in creating a model that is useful. Garbage in, garbage out. Comments within the dataset are collected from the VUStats database which contain monthly comments left by data collectors for each park. We have access to all 34,862 comments that range from 1993-2024. While most of the comments are short, and concise, some are long and verbose so its important to build a dataset that is representative of the entire population. Its important to note that these comments often follow a pattern within each park since data collectors generally format their comments the same for each month. That means one park may post a comments that are short and concise while another park may post comments that are long and verbose. To ensure the training dataset was well represented, a stratified split was used to select 100 random parks, with each strata randomly selecting 3 years worth of data. After sampling, we have the following dataset:

	Split	Number of Comments	Token Count
Train	75%	1,192	45,649
Test	25%	398	17,552
Total	**100%**	**1,590**	**63,201**

Table 2: Dataset Split and Size

Tokens are not too critical to understand in this context, but to give a brief overview, a token can be thought of as a word or a part of a word. Tokens are important because they are how computers are able to understand and process text. Essentially on a high level, each word is assigned a number, and after enough training, a model can understand these patterns to predict what the next token should be.

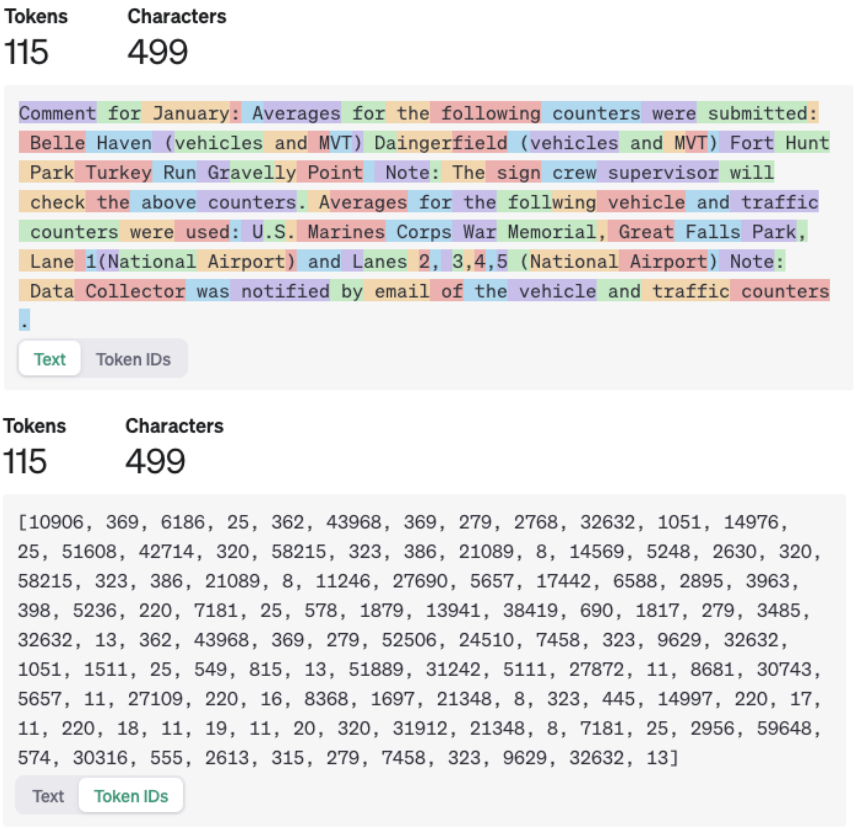

Figure 2: GPT-3.5 Tokenization Example [7]

The last step is to generate the output, which is the format in which we want the model to convert these comments to. This was a hard part of the project to plan for, as I had considered many different ways to generate the output. The most obvious way was to manually rewrite the comments by hand, which also ensures that the output is correct. In relation to training a model from scratch, fine-tuning does not require a large dataset by any means. In some cases, a few hundred to a few thousand examples might be sufficient enough to get a decent model. However, rewriting 1,590 comments by hand is not feasible by any means for a single person. After deliberation, I opted to user OpenAI's GPT-3.5 Turbo API to generate the output for each comment. This took only a few minutes to run along with a cost of 35 cents. The output quality and accuracy was more than satisfactory for what I needed and saved me a lot of time. What's interesting about this technique is

we are sort of 'distilling' the capabilities of a larger Transformer into a smaller one. **To be clear, the use of OpenAIs GPT-3.5 Turbo was only used to expedite the dataset building process and is not used in the final model.** Now you might be wondering, why not just use GPT-3.5 Turbo for the entire project? The use of OpenAIs GPT models are actually great options for companies that do not have the resources to train their own models. However, the VUStats program needs a model that can be run locally, without the need for an internet connection. So using GPT-3.5 Turbo to generate 1,500 outputs to train a model to run locally is a great use-case for this technology.

2.2 Model Selection

There are many open-source language models available to choose from, but as of completion of this project, Meta recently launched their Llama-3 models in both 8B and 70B parameter sizes. In fact, the Instruct tuned version of Llama-3-8B outperforms Llama-2-70B on many benchmarks, which is quite impressive considering the size difference [5]. Llama-3 was trained on over 15 trillion tokens, of publicly available text with a cutoff date of March, 2023 [5]. This is great since we get both a large variety of knowledge that is up to date. Its important to note here that there are multiple versions of Llama-3-8B, which are fine-tuned for tasks like chatbots, instruction following, or basic autocompletion. In this project, I used the base transformer version of Llama-3-8B, which means it has not been fine-tuned for any specific task. This is important as we do not want to fight against the model's preconceived notions of how to format the generated text. Fine-tuning a model that is already fine-tuned can dilute the model's capabilities.

2.3 Quantization

Quantization is a technique that reduces the precision of the model's weights and activations. This is extremely important as it lowers the barrier to entry for people to run models on their local machines. However, this comes at a tradeoff, we are increasing the efficiency at the

cost of accuracy [2]. By converting models from 32-bit floating point to a lower precision, like 4-bit integers, we can reduce the model size by 4x. Llama-3-8B has about 8 billion parameters, which works out to be $8,000,000,000 \times 32/8 = 32,000,000,000$ bytes or 32 GB. With my M1-Pro MacBook Pro (16GB of RAM), I would not be able to run this model without quantization. After quantization, the model size is reduced to 8 GB, which is more than feasible to run on my machine.

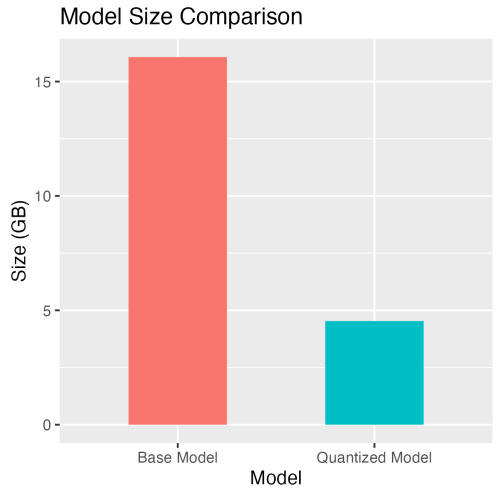

Figure 3: Quantization

2.4 MLX

MLX is a NumPy-like library that is optimized for machine learning on Apple Silicon. It is built on top of Metal, which is Apple's GPU acceleration framework [3]. This is important as it allows me to take advantage of the M1 Pro's GPU to speed up computation. This is just another layer of optimization that allows me to run the model on my local machine, allowing me to train more advanced models. This is important because training LLMs on my MacBook would not have been feasible without MLX.

2.5 LoRA & QLoRA Fine-tuning

The process of fine-tuning a large language model is typically done by training the model on a domain-specific dataset to improve the model's performance on that specific task. However, this requires training all 100% of the model's parameters, which is computationally expensive. LoRA is a supervised learning technique that allows us to train a subset of the model's parameters, by introducing small, low-rank matrices that work in correspondence with the original model's weights to produce adapted outputs [4, page 2]. The authors of LoRA present two main advantages of this technique: 1) one singular pre-trained model can be used to build many task-specific LoRA adapters, and 2) LoRA's efficiency reduces the hardware requirements for training [4, page 2]. Think about these adapters as nozzles to a power washer. By swapping out these fine-tuned adapters, we can drastically change the way the model behaves. For example, we could train many adapters for different use-cases, effectively building a "toolbox" of LLM adapters. QLoRA extends LoRA even further by introducing quantization to further increase efficiency. Authors of QLoRA claim that their new technique reliably matches performance of 16-bit floating point precision on academic benchmarks [1, page 7].

The main takeaway here, is we are utilizing all of the efficiency techniques available to us to fine-tune an LLM on a domain-specific dataset for a specific use-case; and most importantly, we are doing this **locally**.

This took a lot of trial an error to get right, but after many iterations, overnight runs, and a "toolbox" full of QLoRA adapters, we can compare the performances of each adapter. In training the adapters, I experimented with many prompt formats, batch sizes, and the number of layers to fine-tune. The largest factor in performance was how the prompt was formatted, not so much the number of layers or batch size. Although increasing the number of layers and batch size did increase performance, it was not as significant as changing the prompt format.

The advanced prompt formats are not shown here due to length, but are available in the

code repository. The main difference between the basic and advanced prompts is the inclusion of separators between the prompt, input and output tokens along with a few other formalities. Ex. `### INSTRUCTION: <prompt> ### INPUT: <input> ### OUTPUT: <output> ###`. This helps the model understand where each part begins and ends.

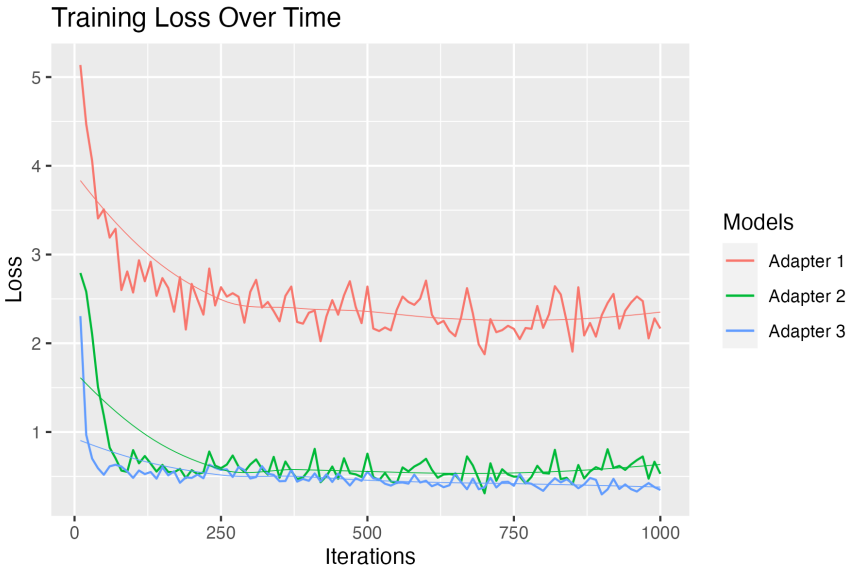

Figure 4: Adapter Performance

Adapter	**Prompt**	**Batch Size**	**Lora Layers**	**Trainable Parameters**
Adapter 1	Basic	1	4	0.426M
Adapter 2	Advanced	1	4	0.426M
Adapter 3	Advanced	4	32	3.408M

Table 3: Adapter Performance

In the end, Adapter 3 was the best performing adapter. Once fused with the base Llama-3-8B model, we can now run the model for inference. Below is an example of the model's output for a given input comment. For clarity, the closure related information is highlighted

in the main comment. The model accurately identifies the closure information and outputs it in a clear and concise format. This is a huge win for the VUStats program as it will save them time and resources in the long run.

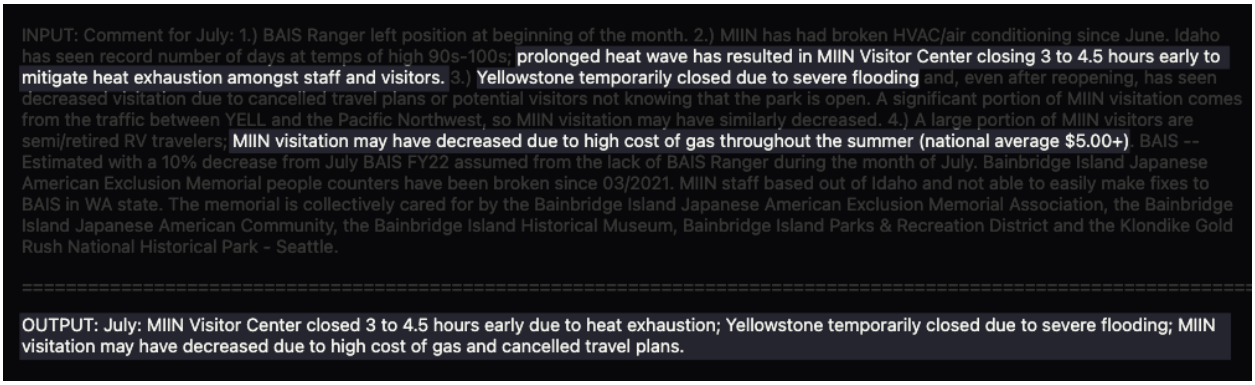

Figure 5: Adapter 3 Output Example

3 Conclusion

In this paper, we show not only the feasibility of fine-tuning a large language model on a local machine with innovative techniques, but also the importance of domain-specific datasets. For model performance, it is clear that the prompt format is more significant than just feeding it the correct input/output and throwing it into a training loop. Although the model does a great job at extracting closure information, there is still plenty of room for improvement, even with the current resources I have access to. I think the next steps would be to experiment with better prompt formats along with more training data. This project would not have been possible without recent advancements in machine learning, such as LoRA and QLoRA. It is an exciting time to witness the forefront of NLP research in the open source community and I am excited to see how it will continue to evolve in the future.

References

[1] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.

[2] Wenqi Glantz. Democratizing llms: 4-bit quantization for optimal llm inference. *Towards Data Science*, 2024.

[3] Awni Hannun, Jagrit Digani, Angelos Katharopoulos, and Ronan Collobert. MLX: Efficient and flexible machine learning on apple silicon, 2023.

[4] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[5] Meta. Llama-3-8B base model used for fine-tuning, 2024.

[6] OpenAI. DALL·E 3: Prompt: "photorealistic version of a llama with 4 legs wearing a park ranger vest and a park ranger hat wearing sunglasses", 2024. This image was generated with the assistance of AI.

[7] OpenAI. Tokenizer tool, 2024. Tokenizer example.