Reducing Computational Costs in Fine-Tuning Large Language Models with TF-IDF Compression

*First and second authors contributed equally to this research.

James Lee
Computer Science
New York University
New York, USA
jhl834@nyu.edu

Jack Yang
Computer Science
New York University
New York, USA
jacky@nyu.edu

Adam L. Meyers, PhD
Computer Science, Linguistics
New York University
New York, USA
meyers@cs.nyu.edu

Abstract—This research delves into the effects of integrating a compression layer, utilizing a TF-IDF filtering technique, on the training duration and performance efficacy of expansive language models in disease categorization tasks. The initial benchmark is set using a pre-trained GPT-Neo model, and two supplemental models are fine-tuned, both in the presence and absence of the compression layer. The study design pivots around the variable of the compression layer's presence or absence, while the dependent outcomes are gauged in terms of training duration and model performance, the latter quantified through accuracy metrics on a test dataset of disease symptom relationship. This experiment took advantage of the generative nature of the model and used a natural language understanding approach to prediction allowing for a high variance in output tokens to capture a correct disease classification.

Three distinct datasets are employed in this study: a consolidated corpus drawn from seven medical textbooks, a preprocessed rendition of this merged dataset, and a compressed version of the preprocessed dataset. The compressed dataset is constructed by preserving a certain percentage (n%) of the words exhibiting the highest TF-IDF values from each abstract postfiltering.

Performance appraisal of the models is conducted on a

Kaggle medical dataset. Notably, the fine-tuned model showcased enhanced performance, attaining a score of 0.39%. However, the model faced a convergence issue and scored 0% when subjected to training on the compressed dataset.

This investigation elucidates that the introduction of a compression layer holds the potential to expedite the training process of extensive language models. Nevertheless, careful deliberation is warranted in selecting the compression ratio and technique to avert potential convergence dilemmas. As we extend our research into future studies, we aspire to uncover more efficient and effective compression methodologies that preserve model token distributions.

Index Terms—Large Language Models (LLM); Fine-Tuning; Generative Pre-trained Transformer (GPT); GPT-Neo; Computational Efficiency; TF-IDF (Term Frequency-Inverse Document Frequency); Text Classification

I. Introduction

The advent of large language models and transformers, such as GPT-3 and BERT, has significantly advanced the

Computing resources for this research were sponsored by New York University using the *Greene Supercomputer*.

performance in various natural language processing (NLP) tasks. However, training these models is often constrained by factors like computational resources and time requirements.

In this research, we explore methods to make the finetuning process of large language models less computationally demanding, with a focus on reducing training time while maintaining quality. We use a pre-trained GPT-Neo model from Hugging Face, containing 2.7 billion parameters, as a baseline and test its performance on a comprehensive dataset for disease classification tasks, such as symptom diagnosis. Accuracy is used as the evaluation metric.

Our study compares the performance of the baseline model with control and experimental models. The baseline model serves as a point of comparison to evaluate the effectiveness of fine-tuning with and without a compression layer applied to the dataset using a TF-IDF filtering technique. The compression layer aims to reduce the dataset's dimensionality by filtering out less important words based on their TF-IDF scores. The TF-IDF weighting scheme determines a term's importance in a document relative to its frequency in the dataset and inversely proportional to the number of documents containing it. We chose this method because it is a widely used vectorization technique that can offer insights into individual words' significance within a corpus. By comparing the baseline and experimental model performances, we assess the impact of the compression technique on the model's performance.

In conclusion, our research investigates the possibility of making large language models more accessible to researchers and hobbyists while preserving their quality. It highlights the trade-off between training time and quality, offering insights that could inform future developments in fine-tuning large language models.

II. Methodology

A. Research Design

In this study, we adopt an experimental, quantitative research design to investigate the impact of applying a compression layer to the dataset using a TF-IDF filtering technique [1] (see Equation 3) on the training time and performance of large language models in disease classification tasks. Our research hypothesis posits that the use of the compression layer will reduce the training time of the large language model while maintaining its performance. We compare three models in our study: the baseline GPT-Neo model, the control model (finetuned without compression),

and the experimental model (finetuned with the compression layer). The presence or absence of the compression layer serves as the independent variable, while the dependent variables are the training time and model performance, measured by accuracy.

$$\text{TF}(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d} \quad (1)$$

$$\text{IDF}(t,D) = \log \frac{\text{Total number of documents in corpus } D}{\text{Number of documents containing term } t} \quad (2)$$

$$\text{TF-IDF}(t,d,D) = \text{TF}(t,d) \times \text{IDF}(t,D) \quad (3)$$

B. Datasets

- raw_merged_dataset.txt: This dataset comprises a merged corpus from seven medical textbooks. Sections within the dataset are labeled using "ABSTRACTID", effectively partitioning the corpus into several relevant documents for TF-IDF calculations. The partitions are based on the U+000C format feed Unicode preserved when converting PDFs to text. The dataset contains approximately 9,210,256 words and has a file size of 73.1 MB.
- preprocessed_dataset.txt: This dataset is a preprocessed version of the merged_dataset.txt and is fed into the compression layer. Preprocessing steps involve removing stop words, non-basic ASCII characters, digits, hyphens, capitalization, and words with less than four characters. The preprocessed dataset has approximately 5,303,557 words, which is 3,906,699 words less than the merged_dataset.txt, and has a file size of 47.2 MB.
- raw_compressed60_dataset.txt: This is a compressed version of preprocessed_dataset.txt, created by retaining n% of the highest TF-IDF words in each abstract from filtering. The number at in file name indicates the compression amount. In our case, the compression value is 60 percent and it is calculated as the ratio of words in the compressed dataset to the total words in the preprocessed_dataset.txt. The raw compressed contains approximately 2,121,114 words, which is about 3,182,442 words less than the preprocessed_dataset.txt, and has a file size of 19.5 MB.
- compressed60_dataset.txt: This dataset is created by removing section labels and duplicate lines from the raw_compressed60_dataset.txt. Additionally, periods are added to the end of each line to optimize the max_length parameter in the tokenizer. This optimization is necessary to prevent significant decreases in training performance due to the absence of delimiters in the training dataset. This dataset is the final version to be used for training.
- *Kaggle* Medical Dataset: This dataset is sourced from *Kaggle* and contains structured lines of symptoms and their corresponding diagnoses. It serves as an answer key for the evaluation of our models in disease classification tasks.

C. Models

1) Baseline Model: GPT-Neo: The baseline model is a pre-trained GPT-Neo model provided by Hugging Face. It is an open-source GPT-3 developed by EleutherAI, featuring 2.7 billion parameters. GPT-Neo is a state-of-the-art transformerbased language model, which is built upon the foundational concepts of self-attention mechanisms and multi-head attention. It is known for its high performance on various NLP tasks.

The architecture of GPT-Neo (see Figure 1) is derived from the GPT series, following the Transformer architecture introduced by Vaswani et al. in their 2017 paper "Attention is All You Need" [2]. The model consists of multiple layers of transformer blocks, each comprising a multi-head selfattention layer and a position-wise feed-forward neural network layer. The self-attention mechanism enables the model to focus on different parts of the input sequence, depending on the context, while the position-wise feed-forward layer helps process and capture local information within the sequence.

GPT-Neo utilizes layer normalization and residual connections to facilitate the training of deep networks by mitigating the vanishing gradient problem. Additionally, the model employs positional encoding to inject information about the relative positions of tokens in the input sequence, as the selfattention mechanism itself is not sensitive to the order of the input elements.

Compared to other large-scale language models like GPT-3 or BERT, GPT-Neo is relatively smaller in terms of parameter count, making it a more efficient choice for our study. Its smaller size allows for a more manageable fine-tuning process while still maintaining state-of-the-art performance on various NLP tasks. As a baseline, the GPT-Neo model serves as a reference point for comparing the performance of the control and experimental models.

2) Control Model: Fine-tuned GPT-Neo: The control model is a fine-tuned version of the GPT-Neo baseline model, which is adapted specifically for disease classification tasks. The fine-tuning process involves training the model on our

raw_merged_dataset.txt without applying the compression layer. This model helps us understand the performance and training time of the GPT-Neo model when fine-

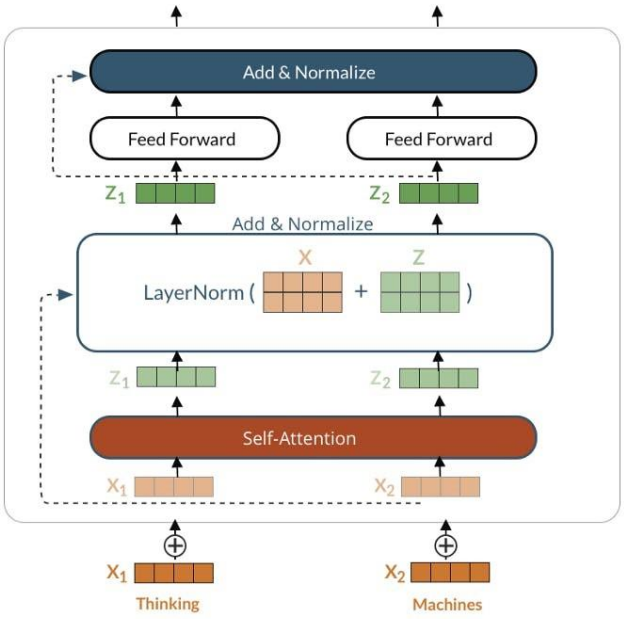

Fig. 1. GPT-Neo architecture illustrating the transformer-based structure. Image source: https://gpt3demo.com/apps/gpt-neo.

tuned for the task without any additional modifications or optimizations.

3) Experimental Model: Compressed GPT-Neo: The experimental model is a GPT-Neo model that undergoes fine-tuning with the dataset processed through the compression layer. This compression layer leverages the TF-IDF filtering method to minimize the dataset's dimensionality by preserving only the most significant words in each document as determined by their respective TF-IDF values. The purpose of this model is to examine our research hypothesis, which suggests that incorporating the compression layer should decrease the training duration for the large language model without compromising its performance.

III. Implementation

A. Data Processing

- merger.py: A Python script that combines medical texts from a given directory and splits the combined corpus into several relevant documents. It does this by replacing the U+000C special Unicode character with "ABSTRACTID" section headers, which are later used for calculating the inverse document frequency (see Equation 2). The output file generated by this script is raw_merged_dataset.txt.

- cleaner.py: A preliminary Python script that takes raw_merged_dataset.txt as an input and performs several preprocessing tasks, such as parsing, tokenization, and removal of stop words, non-basic ASCII characters, digits, hyphens, capitalization, and words with fewer than four characters. It ignores the section headers during this process. The output file produced by this script is preprocessed_dataset.txt.

  Note: Both merger.py and cleaner.py are part of the preprocessing phase.

- smusher.py: A Python script to perform the final checks before generating the final datasets for training. It takes either raw_compressed60_dataset or merged_dataset.txt as optional inputs and removes newlines and section headers. If the input is raw_compressed60_dataset, the script adds periods to the end of each line, excluding blank lines, to reintroduce the delimiters that were removed during the preprocessing step. This step is essential for preventing issues with the tokenizer's padding, which could lead to inefficient tokenization and significantly increase training time due to difficulties in the optimization process. In the absence of proper delimiters, the tokenizer might produce poorly structured input sequences, leading to longer and less informative training sequences. As a result, the model might require additional training iterations to converge, and the gradient descent optimization process could become slower and less efficient. By including periods as delimiters specifically for the raw_compressed60_dataset, we ensure that the input sequences are well-structured and informative. This, in turn, leads to faster training times and a more effective optimization process to minimize the loss function. The outputs produced by this script are raw_compressed60_dataset.txt and raw_merged_dataset.txt. However, note that raw_merged_dataset.txt has no significant impact on training time.

B. Compression Layer

The following Python libraries are used in our compression algorithm:

- collections: We use the defaultdict class from this library to create nested dictionaries with a specified default value.

- sklearn.feature_extraction.text: Additionally, the TfidfVectorizer class from this library is used to convert a collection of raw documents to a matrix of TF-IDF features.

In the th code snippets, we describe the following functions in our algorithm: compute_tfidf, sort_tfidf_dict, and get_filter_words.

```
def compute_tfidf(documents): vectorizer = TfidfVectorizer() tfidf =
    vectorizer.fit_transform(documents) feature_names = \
        vectorizer.get_feature_names_out()
    tfidf_dict = defaultdict(dict) for doc_index, doc in

    enumerate(documents): feature_index = \ tfidf[doc_index,

    :].nonzero()[1]

        for i in feature_index:
                                tfidf_dict[doc_index][feature_names[i]]\
                = tfidf[doc_index, i] return tfidf_dict
```

The compute_tfidf function determines the TF-IDF values for every word in each document. It takes a list of documents as input and uses the TfidfVectorizer to form a TF-IDF matrix. Next, the feature names (words) are obtained using the get_feature_names_out method. The function then constructs a nested dictionary (tfidf_dict) with the defaultdict class, where the outer key pertains to the document index and the inner key signifies the word. The calculated TF-IDF values are stored in this nested dictionary.

```
def sort_tfidf_dict(tfidf_dict):
    sorted_dict = {} for doc_id, tfidf_values in tfidf_dict.items():
        sorted_values = sorted(tfidf_values.items(), key=lambda x: x[1],
            reverse=True)
        sorted_dict[doc_id] = dict(sorted_values)
    return sorted_dict
```

The sort_tfidf_dict function sorts the TF-IDF values for every word in each document within the tfidf_dict dictionary. A new dictionary, sorted_dict, is created, in which each entry is a sorted dictionary of the TF-IDF values for the respective document, ordered from highest to lowest.

```
def get_filter_words(percentage, sorted_dict):
    filter_words = {} for document in
    sorted_dict:
        filter_count = int(percentage
            * len(sorted_dict[document])) curr_count = 0
        doc_filter_words = [] for key in
        sorted_dict[document].keys(): if curr_count >=
        filter_count: break
            if key not in doc_filter_words:
                doc_filter_words.append(key) curr_count += 1
        filter_words[document] = doc_filter_words
    return filter_words
```

The get_filter_words function acquires the takes the n% words in each document of the sorted dictionary. Using a percentage value and the sorted dictionary as input, it goes through every document and calculates the number of words for the filter words list. The function forms a new dictionary (filter_words), with each entry being a list of the top n% words for the corresponding document.

These functions process the input corpus to get the filter words, which are then given to the compress function. This function re-processes the corpus and compresses it by excluding words not in the filter words list. As a result, the compressed corpus maintains only the most relevant words based on TF-IDF values. The che changes are written in raw_compressed60_dataset.txt.

C. Training Process

The training of Large Language Models is often quite computationally expensive and must be done on speciality hardware. The fine tuning task on both a large corpus and compressed corpus was completed on an A100 with 40gb of GPU RAM. In the training process the GPTNeo 2.7B took on a learning rate of $1 * e^{-6}$, a batch size of 256 and a weight decay of $.01$. The model then did next token prediction with Negative Log Likelihood Loss and trained. For the uncompressed model a max token length of 25 was used meaning the model would only do next token prediction for the next 24 tokens. For the compressed model the median + 1 was also taken resulting in a max token length of 4. This example also had high variance so many examples were truncated or padded. The variable length becomes a problem as explained in results. This model then uses the Hugging Face Trainer method with those hyper parameters and trains for 3 epochs.

D. Evaluation Metrics

The model was evaluated on a test set that would take advantage of the next word generator method. This test set came from a symptom disease classification task. Each row has multiple symptoms and a single disease as target variable. To make the test data more LLM accessible, a method was written to put it in the following format. "A patient has the symptoms, symptom_1, symptom_2, symptom_3, (etc) as a doctor I classify the disease to be ". The model then predicts the following ten tokens. These are variable length based on the number of symptoms present. To evaluate the performance of the model a simple test is done to see if the actual disease string is in the generated. For example "A patient is sick and have the symptoms, chills, vomiting, sweating, headache, nausea, diarrhea, muscle pain as a doctor I classify the disease to be (model::) either acute or chronic. The acute form is characterised by a sudden onset of symptoms, usually within hours of exposure to a toxin.<|endoftext|>" The actual disease is 'Malaria'. Because Malaria was not in the response a score of 0 is given to this example. Note: lower case comparison was done to avoid losing positive hits due to capitalization.

For examples where the disease is present it will receive a score of 1. These training examples lie a data set of 4096 examples with even distribution among diseases (Figure 2. They are randomly sub sampled to recreate a 1000 row data set for speeding up evaluation tests. For the final evaluation metric, a models score will be a sum of all the positives or 'hits'.

IV. Results

The performance of the model was evaluated across three categories: the baseline, the complete corpus, and the compressed corpus models. The baseline model, GPTNeo2.7B, was selected primarily due to its ability to deliver coherent responses that generally aligned with the evaluation metric. However, the model's capacity to accurately predict the specific disease within the next ten generated tokens was found lacking. The baseline model scored a 0.18%.

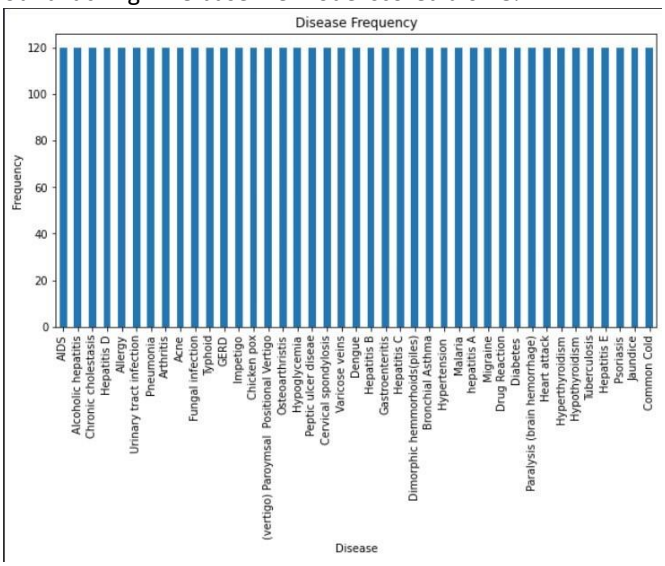

Fig. 2. Distribution of disease frequency in the evaluation dataset, showing an even distribution across disease categories.

Upon closer examination of the results (see Figure 3), it was noted that the model could only identify instances of Diabetes, yielding a score of 18/20 for the total number of diabetesrelated target variables in the test set. Despite providing coherent responses to most queries, the model struggled with moving beyond broad disease generalizations. Diabetes, being a commonly known disease and thus prevalent in the pile (GPTNeo2.7's original training set), may have contributed to the model's ability to identify it. However, due to the limited exposure to other diseases and associated symptoms, the model often resorted to generic statements.

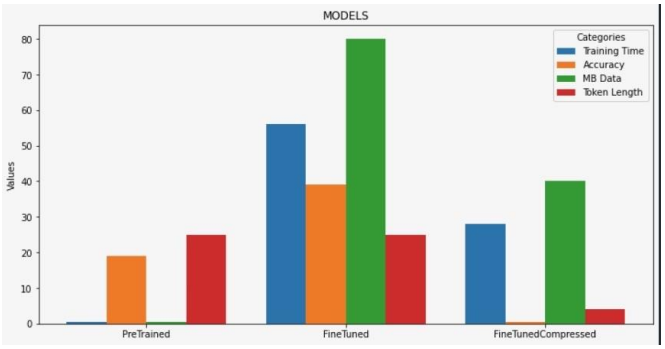

Fig. 3. Comparative analysis between the baseline, control, experimental models.

The fine-tuned model demonstrated a marked improvement in performance, achieving a score of 0.39%. This model managed to identify all instances of Diabetes and Urinary Tract Infections, as well as some instances of diseases such as Gastroenteritis and Arthritis. The model's responses also exhibited a coherence not accounted for in the evaluation metrics. It provided comprehensive information, including whether the condition was chronic or acute, bacterial or viral, and even suggested additional symptoms and treatment options.

However, when exposed to the compressed dataset, the model encountered a convergence issue and scored 0%. Upon further inspection, the model, although maintaining grammatical coherence, repetitively predicted "erythema infectiosum". It did not appear to reference the preceding symptoms. This may be attributed to the fragmented nature of the data it was fed and the exceedingly small token length. Consequently, the model's token or look-back range was significantly reduced. It was observed that the model only registered the invariant phrase "as a doctor, I classify the disease to be" and subsequently generated the same output token.

It's also worth noting that the loss for the fine-tuning task plateaued at about 2.1. In contrast, the compressed model never managed to reduce its initial loss of 4, indicating its inability to master next word prediction. This observation aligns with the fact that all standard grammatical structures were eliminated, leaving only complex, domain-specific words. The token length posed an additional challenge. Given the compressed dataset's average sentence structures were reduced to approximately four tokens, the model was trained to attend over very short periods, rendering it ill-suited for the classification task.

V. Further Experimentation

This experiment had many fascinating results and thus many interesting avenues for further experimentation. The following are areas of research that will be pursued in follow up. Creating longer token length for the compressed model by formatting

the training examples with be of length 25 as they were in the fine tuning example. This would give the model to attend to more of the test data. As well different examples of testing could take place to see how the model would preform even with shorter token lengths for instance "diabetes has the symptoms" as a input and testing for symptoms instead of disease. These models with low billion parameter counts do not seem to hold on to identity as well as larger models so simply changing the task to question and answer could be beneficial and capture the medical domain task better. As seen with the lack of ability for training loss to go down when training the compressed model inference can be drawn about the lack of grammar structures and how that would throw off a model's ability to predict next token as normal relations are removed. This leaves two main options. Create a testing set similar to the compressed dataset and see if the domain only words hold relation in the task, as well as do training on greater number of epochs to account for the extreme out of distribution token arrangement. The other and more feasible option is to rewrite the compression algorithm to take sentences with domain important words and preserve the whole sentence. Thus taking a dataset and returning only valuable sentences. This could also use the TFIDF based methods. This would prevent the model from losing grammar and casual based relationships that it had learned during its standard training.

Lastly, experimentation on other filtering techniques such as cosine similarity would be relevant to our experiment. By creating a distribution of domain words and then comparing corpus words to these domain words through a similarity score, then taking the top n% would also yield a powerful filtering result. Overall the purpose of this experiment was to explore techniques on how a dataset can be optimized for Large Language Model fine-tuning. It seems that the difficulties of word filtering create serious issues in the alignment of the model with this task. Going forward sentence based filtering whether through TF-IDF or cosine similarity seem like the best options.

VI. Conclusion

In conclusion the loss of sentence structure and exclusive domain word fine tuning to a Large Language Model such as GPTNeo2.7b causes far worse model performance than untrained model. Further use of compression techniques that preserve essential structures are necessary for tasks involving domain specific language generation are needed.

Ultimately, this study presented an analysis on the impact of using TF-IDF compression on noisy datasets and how it performed in addition to fine-tuning. Our results show that the technique can significantly reduce file size, effectively reducing the time it takes the models to train, albeit at the potential cost of performance. While our initial findings may not show state-of-the-art improvements, this investigation is the first step needed to explore new optimization techniques, which we will comprehensively test in further research.

Acknowledgments

We would like to extend our appreciation to Thaison Le (tnl2012@nyu.edu) for his invaluable support in training our models on the *Greene* HPC cluster. His assistance has been instrumental to the development of this research.

Furthemore we would like to acknowledge the authors of the medical textbooks *Goldman-Cecil Medicine*, 24th edition; *Kumar and Clark's Clinical Medicine*, 7th edition; *Macleod's Clinical Examination*; *Step-Up to Medicine* (Step-Up Series); *Davidson's Principles and Practice of Medicine*, 21st edition; *Harrison's Principles of Internal Medicine*, 16th edition; and *The 5 minute clinical consult*, 2021, from which we sourced the merged dataset used in this study. Note: The use of these sources does not imply their endorsement of the conclusions drawn in this research.

References

[1] Qaiser, S. and Ali, R., "Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents," *International Journal of Computer Applications*, vol. 181, 2018. doi: 10.5120/ijca2018917395.

[2] V. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 5998–6008.