International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)
https://ijcjournal.org/index.php/InternationalJournalOfComputer/index

Optimal Use of Computational Resources when Using LLM

Sukanya Sahoo[a*], Prakhar Goel[b]

[a]*Independent Researcher, Bangalore, 560066, India*
[b]*Independent Researcher, Bangalore, 560066, India*
[a]*Email: sukanyasahoo41455@gmail.com*
[b]*Email: goelprakhar2000@gmail.com*

Abstract

Large Language Models(LLM) are the machine learning models that are used to understand and generate human type languages, and also they have proven outstanding performance on a variety of natural language processing tasks, such as sentiment analysis, text generation, text completion, question-answering, language translations, etc. LLMs models are based on neural networks, and it uses a technique of pre-training that is used to learn representations of language, that can be further fine-tuned for specific tasks. There are many ways through which the computational resources consumed by these models during inference can be optimized. The first approach is the optimization of the model architecture, where the architecture is modified in such a way that will reduce the number of parameters, and all the computations required for the inference. There are many more optimization techniques like quantization, knowledge distillation, onnx conversion, approximation techniques like low-rank matrix factorization and so on. In this paper we discuss all these approaches and the results gained from using it.

Keywords: Optimization; Quantization; Pruning; ONNX; Knowledge distillation; Approximation techniques.

1. Introduction

Language models (LMs) are a class of machine learning models that are designed to process and understand natural language. In recent years, advancements in deep learning have led to the development of large language models (LLMs), such as BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pre-trained Transformer), etc that have achieved remarkable performance in a wide range of natural language processing tasks, including language generation, translation, and sentiment analysis. These models generally consist of billions of parameters that have been trained on a massive amount of text data and produce the responses like humans.

Received: 5/2/2023
Accepted: 6/8/2023
Published: 6/18/2023

---

* Corresponding author.

However, the complexity and the size of the model, make it computationally demanding, as it requires a significant and huge amount of resources for inference and training. Therefore, to optimize the resources consumed by the LLMs, they should deploy the devices that are resource-constrained, so that it can become more accessible to a larger audience.

This has led to the development of various techniques, including model architecture optimization, quantization, pruning, knowledge distillation, approximation techniques, and ONNX conversion, which can reduce the computational requirements of LLMs without sacrificing much in performance.

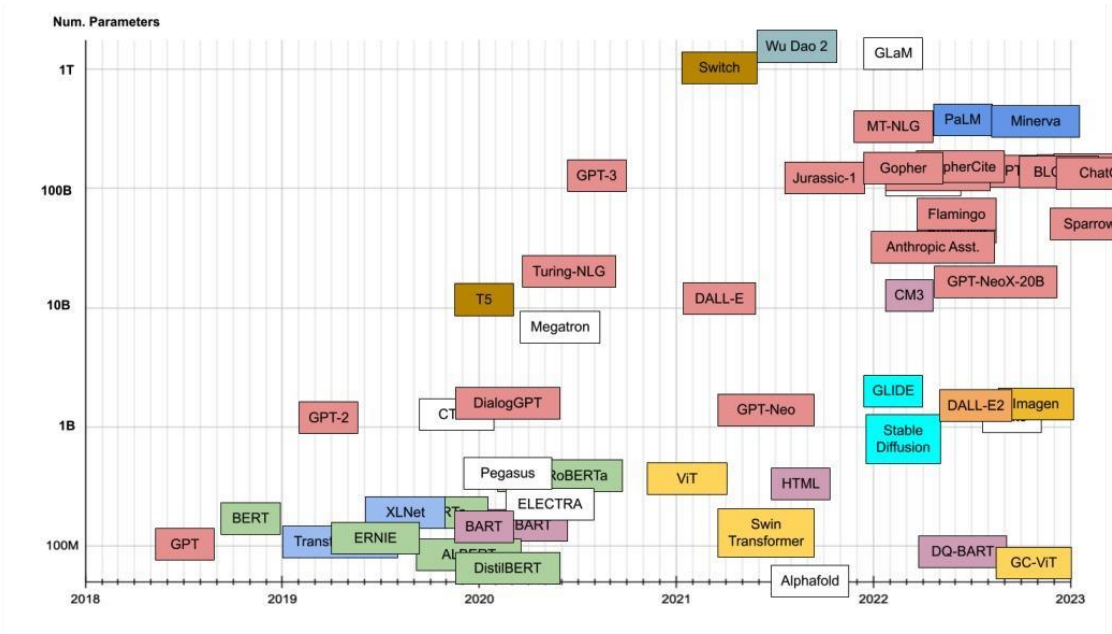

Figure 1: Evolution of LLMs throughout the decade [1].

The training of LLMs consists of multiple steps. The model is first trained using an unsupervised technique, from which they infer the connection between words and concepts. After that, supervised learning is used to polish it. After passing the training data via a transformer, the LLM can use a self-attention mechanism to identify connections and linkages. Once the LLM is trained, it can be used as the base for any AI uses. LLM has the capability for generating text, translating languages, organizing content, rewriting or summarizing content, and analyzing the sentiment of content like tone or humor. LLMs can be particularly useful as a foundation for customized uses for both individuals and businesses. They are accurate, fast, flexible, and easy to train. Despite having too many positive points, there are too many challenges that LLM poses.

- Cost of deployment and development:
- Bias on what data the training is being done on
- AI hallucinations (where responses are not based on the training data)
- Troubleshooting complexity
- Glitch tokens or words or inputs maliciously designed to make the LLM malfunction.

1.1. Types of LLM

There are many different types of large language models, and each of the models has its own strengths and weaknesses.

1.1.1. Autoencoder-Based Model

Autoencode-based models learn from the text data that are in a compressed representation. In this method, the input text is transformed into the lower-dimensional latent space, and then later decodes the latent representation to recover the same text as it was, i.e., the original text. This full procedure produces the new text using the compressed representation similar to data compression and decompression.

These models are trained using the concept of unsupervised learning. This model accepts a huge amount of text input and then later it is trained to identify the underlying patterns in the data. This model further can be utilized for a bunch of tasks after it is trained, such as text completion, text generation, and classification text.

A lot of advantages are there of the Autoencode model as compared to the conventional language model. First, by learning more condensed versions of the input data, they can use less computing power during training and inference. Second, the model is more adaptable because of the compressed representation's application in downstream tasks like text categorization and sentiment analysis. Finally, autoencoder-based models are advantageous for text production tasks like dialogue generation and language translation because they can learn to produce high-quality text.

1.1.2. Sequence-to-Sequence Model

Sequence-to-sequence (Seq2Seq) models are a kind of LLM that can handle variable-length input data sequences and generate variable-length output data sequences. The encoder receives the input sequence and creates a fixed-length representation of it, and the decoder receives the fixed-length representation and creates the output sequence in seq2seq models.

This model aims to pass a fixed length input and to get the fixed length output also, where the length of the input and output can vary accordingly. Let’s consider one example like if any English text needs to be converted to a Chinese text, so 6 words input may result in 8 symbols. So, in these cases, we can’t use a normal LSTM network in order to map each word from a particular English sentence with a Chinese sentence. In order to address these types of issues, the best model to be used is the Sequence-to-Sequence model.

Some of the applications of the Seq2Seq model are that it is used in machine translations, Speech Recognition, and Capturing of Videos (to generate the descriptions of movies).

This model consists of 2 things and that are encoder and a decoder. In the form of a hidden state vector, the encoder records the context of the input sequence and passes it to the decoder, which subsequently creates the output sequence. Both the encoder and the decoder often use some type of RNNs, LSTMs, GRUs, etc. because

the task is sequence-based. Although the concealed state vector can be any size, it is typically taken to be a power of two and a large number (256, 512, or 1024), which can in some ways indicate the complexity of the entire sequence as well as the domain.

1.1.3. Transformer-Based Models

Transformer-based models are a type of LLM that uses self-attention mechanisms to process sequential data. Recurrent neural networks (RNNs) are not used by transformers unlike other LLMs to process sequential data, which makes them easier to parallelize and more efficient. The transformer architecture was introduced in a seminal paper called "Attention is All You Need" by Vaswani and colleagues in 2017. The main idea behind transformers is to use self-attention mechanisms to capture long-range dependencies between the tokens in a sequence. Self-attention allows the model to attend to different parts of the input sequence to compute a fixed-length representation of the sequence [2]. Transformers consist of an encoder and a decoder, similar to seq2seq models. The encoder processes the input sequence using self-attention and produces a fixed-length representation of the sequence. The decoder uses self-attention to attend to different parts of the encoder output to generate the output sequence. One of the most well-known transformer-based models is the GPT (Generative Pretrained Transformer) family of models, which was introduced by OpenAI. GPT models can be fine-tuned for a range of downstream tasks, including language modeling, text categorization, and question-answering, and are pre-trained on massive amounts of text data.

Compared to other LLMs, transformers have a number of benefits. First off, they are more suited for tasks like language modeling and text production because they can capture long-range dependencies in the input sequence. Second, they can be parallelized more easily and effectively than RNN-based models, making them more scalable to bigger datasets. As a result of their exceptional performance on a number of NLP tasks, they are a great option for many NLP applications.

1.1.4. Recursive Neural Network Models

A recurrent neural network is a type of neural network that is trained specifically to analyze time-stamped sequences of data. RNNs are frequently preferable for jobs involving sequential inputs, such as voice and language. Knowing the terms before a given word in a sentence is crucial if you want to forecast it in an NLP challenge. Recurrent neural networks (RNNs) are so named because they consistently complete the same task for every element in a sequence, with the results depending on earlier calculations.

Although the RNN is a straightforward and effective model in theory, it is challenging to train well in practice. The disappearing gradient and ballooning gradient issues are two of the key causes of this model's unwieldiness. The gradients during back-propagation during training must travel all the way from the final cell to the first cell. These gradients' product either reaches zero or grows exponentially. The term "exploding gradients problem" describes the significant growth in the gradient's norm during training. The converse behavior, the vanishing gradients problem, prevents the model from learning the connection between temporally far-off events when long-term components go exponentially quickly to norm zero.

More advanced RNN models have been developed to address this issue, such as Gated Recurrent Unit (GRU) networks and Long Short-Term Memory (LSTM). These models use specialized memory cells and gating mechanisms to handle long-term dependencies better and prevent the vanishing gradient problem.

1.1.5. Hierarchical Models

The hierarchical structure models are the ideal tools that are used for measuring and identifying the structural relationships that basically fall at multiple levels in the procedure of generating data. These models basically have no limits to the dimensions of the hierarchy they follow.

These models have the capability to facilitate the testing of the hypothesis for analysis across different levels of data.

Varieties of NLP tasks use hierarchical models such as machine translations, text classifications, and machine modeling. They are particularly effective for modeling long sequences of text, where capturing the hierarchical structure of the language can be critical for accurate predictions. However, hierarchical models can also be more computationally expensive than simpler models that only consider one level of abstraction, and may require more training data to learn effectively.

1.2. Where can LLM be used?

In natural language processing (NLP), there is a variety of uses of the Large Language Model (LLM), and some of those are

1.2.1. Language Generation

To generate the natural language text, like machine translations, dialogue generation, and summarization, LLM can be widely used.

1.2.2. Language Understanding

To perform language understanding tasks, like classification text, answering the question, and sentiment analysis, LLM can be widely used for this.

1.2.3. Language Modeling

To perform the tasks such as text segmentation, named entity recognition, or language modeling, it can be achieved by using LLM which will help to model the patterns and structure of natural language.

1.2.4. Speech Recognition

To improve the accuracy of the Speech Recognition System, LLM can be used, it is achieved by converting the audio input to text.

1.2.5. Text-to-speech

From text input, LLM can help in generating Natural sounding speech.

1.3. How have LLMs changed in recent years?

Large Language Models (LLM) have received a lot of popularity worldwide and also have gained a lot of appreciation in the field of Natural language processing (NLP). We are now more capable than ever of describing and explaining an intelligent system with more and better linguistic articulation because of the LLM. The effectiveness of models like the T5, GPT-3, PaLM, etc. has made this possible. These models can do exceptionally well tasks such as imitating humans by learning to text generations, to read, and to summarize long paragraphs, etc.

Let's have a look at some of the notable examples:

1.3.1. Transformer-based architectures

The transformer-based architectures do not depend on the convolutions and recurrence in order to generate the output, they basically use the encoder-decoder-based structure. Here the encoder is responsible to map the sequence of inputs in order to produce a series of continuous representations.

1.3.2. Pre-training on massive amounts of data

Pre-training is a common technique used in LLMs, where the model is first trained on a large corpus of text data and then fine-tuned on a specific downstream task. Recent advancements in pre-training techniques, such as masked language modeling and sequence-to-sequence pre-training, have led to significant improvements in LLM performance.

1.3.3. Efficient training methods

To train large language models, significant computational resources are required, which can be expensive and time-consuming. Recent advancements have made it quite possible to train the models at a lower cost and faster, such as distributed training and parallel training.

1.3.4. Model compression techniques

Model compression aims to provide a model that is more straightforward than the original while maintaining a high level of accuracy. A model that has been simplified typically has less latency and/or size than the original. This is advantageous since it frees up memory for other components of the application to use. This can be done by deploying LLMs more effectively over a wider range of devices by adopting model compression techniques including quantization, pruning, and knowledge distillation.

1.3.5. Multi-task learning

The model learns numerous tasks concurrently while simultaneously optimizing different loss functions in

multi-task learning. Instead of training separate models for each task, a single model can learn to execute all of the tasks simultaneously. The model learns generalized representations of the data that are applicable in a variety of settings by using all of the data that is accessible across the many tasks.

1.3.6. Transfer learning

Transfer learning is a method that involves using parts of an already-trained model in a fresh one. Generalized information can be transferred across the two models if they are created to execute comparable tasks. By using this method, fewer resources and fewer labeled data are needed to train new models. Transfer learning is used to improve performance on specific downstream tasks, such as text classification, sentiment analysis, etc.

1.3.7. Adversarial training

If a model needs to be trained to generate resilient outputs to adversarial attacks, then adversarial training is used. In LLMs, adversarial training has been used to improve the robustness of the model to input sequences that have been modified to cause errors.

1.3.8. Increase in the Model Size

LLMs have become much more complex in recent years as complex models like GPT-3 contain 175 billion parameters. This generally allows the models to learn more sophisticated and nuanced representations of the language.

1.3.9. Improved Task-Specific Models

Now there are many LLMs now which are task-specific like language translations, text completion, and question answering. These task-specific models generally require less computational resources than the general-purpose LLMs, and these models can achieve very high performance on their target tasks.

1.3.10. Ethical Considerations

As LLMs have become more ubiquitous and powerful, there has been a lot of concern about their use and potential ethical implications, such as the environmental impact on training, running these models, bias in their outputs, and the impact on the economy and jobs.

2. Reasons behind high usage of computational resources

Some of the major reasons for the high requirements of computational resources by LLMs are as follows:

2.1. Model size

LLMs have many parameters and layers, which can require more computational resources to run during inference compared to smaller models.

2.2. Sequence length

The attention mechanism in LLMs does a pairwise similarity computation between all the pairs of positions in the input sequence. This becomes computationally expensive when LLMs are used to generate long sequences of text.

2.3. Beam search

LLM beam search is an algorithm for decoding big language models. Finding the most likely group of words from a set of input tokens is the aim of beam search. The algorithm generates the most likely set of words one token at a time through an iterative process. As a result, it requires expensive computation, especially for wide beams.

2.4. Temperature sampling

A hyperparameter that regulates the sampling process's unpredictability is the sampling temperature. The sampling process is more arbitrary the higher the sampling temperature. We introduce less unpredictability into the sampling process by lowering the sampling temperature. For high-temperature values, this may need expensive computational processing.

2.5. Top-k sampling

LLMs may use top-k sampling during inference to limit the number of possible outputs. This technique involves selecting the k most probable tokens at each step, which can be computationally expensive, especially for large values of k.

2.6. Ensemble models

LLMs may use ensemble models during inference to improve performance. This involves running multiple models in parallel and combining their outputs, which can be computationally expensive.

2.7. Hardware limitations

LLMs may require specific hardware, such as GPUs or TPUs, to run efficiently during inference. Applications that do not have access to specialized hardware, become the limitations of that.

2.8. Attention mechanism

LLMs often use an attention mechanism to focus on relevant parts of the input sequence during inference. This mechanism involves computing pairwise similarities between all pairs of positions in the input sequence, which can be computationally expensive, especially for long sequences.

2.9. Vocabulary size

LLMs have a large vocabulary size, which can require more computational resources to process during inference compared to smaller vocabularies.

2.10. Memory constraints

LLMs may require more memory during inference to store intermediate computations and generated outputs, especially for long sequences.

2.11. Batch size

LLMs may require larger batch sizes during inference to achieve optimal performance, which can require more computational resources.

2.12. Model parallelism

LLMs may require parallelism during inference to distribute the workload across multiple devices or nodes. This can require additional computational resources and infrastructure.

2.13. Language constraints

LLMs may require additional computational resources during inference to handle language constraints, such as grammar rules or word order.

2.14. Post-processing

LLMs may require additional computational resources during post-processing to clean up generated outputs, such as correcting spelling mistakes or removing redundant information.

3. Optimizing the use of computational resources during inference

At the time of inference, the model utilizes a lot of computational resources, so let's have a look at some of the ways this utilization of the computational resources can be optimized.

Model architecture optimization Model Architecture plays a vital role in utilizing computational resources. If the parameters used in the models or the number of layers embedded in the models can be reduced, it can be done by using more efficient activation functions or using smaller embeddings of the hidden layer. This approach will help in reducing the computational resources used by LLMs.

Quantization The continuous infinite values are reduced to a more manageable collection of discrete finite values during the quantization process. Through this, the computation time will become lesser and it will use lesser memory too. Thus effectively reducing the precision of the model's parameter.

Pruning Pruning itself says the compression of data, which basically represents that all the unimportant connections or parameters can be removed from the model.

Knowledge distillation Knowledge Distillation is the process where the knowledge is transferred from the larger model to a smaller one. So, the smaller model is being trained, which basically is a copy of the behavior of the larger model.

Quantized fine-tuning In this process, the model is to be trained with lower precision weights and biases, that basically helps in reducing the time required for computation during inference and also helps in reducing memory.

Approximation techniques In this process, the models are learned by doing a mapping or creating approximate functions between the inputs and outputs sequences. In LLM, a low-rank approximation can help in reducing the time required for computation and memory usage too.

Hardware optimization The efficiency of large language models can be improved by optimizing the hardware used by the LLM.

Inference optimization Inference optimization can be accomplished in several ways, including lowering the length of the sequence, using a reduced beam width during decoding, or employing caching techniques to prevent some repetitive computations.

Data pre-processing Data preprocessing is a technique that reduces the complexity or amount of the data by processing the input data as well as the raw data. In this pre-processing, superfluous data from the input data may be removed.

Augmentation of Data By introducing realistic or random changes to the current database, it aids in the creation of a new training dataset, which essentially aids in enhancing the effectiveness and results of the models.

Early stopping Early stopping involves stopping the training process once the validation loss stops improving, which can reduce training time and computational resources.

Gradient accumulation Gradient accumulation involves accumulating gradients over multiple batches before updating the model parameters, which can reduce memory usage and allow for larger batch sizes during training.

Parallelism Using parallelism techniques, such as data or model parallelism, can distribute the computational workload across multiple devices or nodes, which can significantly reduce training time and improve efficiency.

Knowledge reuse Reusing pre-trained LLMs or leveraging pre-trained language models for transfer learning can reduce the computational resources required for training new LLMs.

Model compression Model compression techniques, such as using smaller embeddings, quantization, or pruning, can reduce the size and computational requirements of LLMs without significant loss of performance.

Dynamic evaluation Dynamic evaluation involves using different decoding parameters or techniques during inference, such as varying the beam width or using dynamic programming, to improve efficiency without sacrificing performance.

Algorithmic optimization During the training of the model, the number of iterations needed to reach convergence can be reduced and the memory footprint can be reduced. It will involve techniques like SGD i.e., stochastic gradient descent or Adam optimization.

Hardware optimization While training and inference the hardware used can be optimized by selecting the most suitable one. So basically right GPU selection or using good specialized hardware like TPUs i.e., Tensor processing units

3.1 Model Architecture Optimization

The computational resources used by LLMs can also be modified by using Model Architecture Technique. The structure of the model is being changed in this technique to make it more faster and efficient during inference. The parameters of the models are being reduced in this technique, without actually affecting the performance of the model.

One of the most popular model architecture techniques is to reduce the number of model layers. The only goal is to strike a balance between model capacity and performance, allowing for a reduction in the number of layers without compromising the model's potential to produce high-quality outputs and learn.

Skip connections are yet another method for improving LLM's architectural design. In addition to learning from the preceding layer, this enables the model to have direct access to the input layers. This technique helps the model to reduce the number of layers and maintain its performance.

One such strategy for model architecture that enables the model to learn from the discrepancies between the input and output is the usage of residual connections. The quantity of calculations needed to perform convolutions is decreased by using depth-wise separable convolution.

Several methods for LLM architecture optimization:

Depth reduction: If the depth in the LLM architecture is reduced, the model's processing needs are also reduced, along with the number of parameters. However, the model's performance is unaffected by depth reduction.

Width reduction By decreasing the number of hidden units in each layer can reduce the width of the LLM architecture. It helps in reducing the computational requirements and the number of parameters of the model.

Embedding dimension reduction Computational requirements of the LLM can also be reduced by reducing the dimensionality of the input embeddings

Dropout Dropout is a regularization technique. During training the models, dropout randomly drops out some of the neurons from it, to prevent overfitting.

Dilated convolutions It helps in increasing the receptive field of the model without increasing any computational requirements or without increasing any number of parameters. Dilated convolutions is a type of convolutions layer.

Grouped convolutions Here, the input channel is being divided into groups, and then applying the convolution mechanism separately to every single group.

Factorized convolutions Here the convolutional kernels are decomposed into smaller sub-kernels that generally reduce the number of parameters and also the computational requirements.

Low-rank factorization Here the weight matrices of the LLM architecture are converted into low-rank matrices by decomposing, which reduces the computational requirements of the model.

Attention head reduction If the attention heads are being reduced in the LLM architecture, then the number of parameters is also being reduced.

Transformer architecture modifications The original Transformer architecture can be modified to reduce computational requirements. For example, using a reversible encoder or performing attention over a smaller window size can reduce the number of computations.

3.2 Quantization

By quantization technique, the memory and the computational resources that are being used to store and run the Large Language Model are being reduced. It works by reducing the precisions of the model's parameters. Usually, every single parameter is represented by either a 16-bit, 32-bit, or 64-bit value, which needs significantly high computational resources and memory to store and process. These parameters can be reduced in terms of precision with fewer bits, like 8 bits or even less [10]. Doing this would reduce the computational and memory requirement of the LLM. Moreover, it allows the LLM to run on less powerful hardware or to run more efficiently on more powerful hardware.

Some of the methods to quantize a model include

Post-training quantization Using full precision floating point numbers, the quantization of the model's parameter is being done. The models have to be retrained to minimize the loss caused by the quantization as the parameters are quantized to a lower bit precision [12].

Quantization-aware training In quantization-aware training, the forward pass models low precision behavior while the backward pass is left unchanged. Due to the quantization error this causes, which adds to the model's overall loss, the optimizer makes an effort to lower it by modifying the parameters. This increases the quantization resistance of our parameters, making our procedure nearly lossless.

Dynamic quantization In this method, the activations are dynamically quantized while the weights are quantized beforehand. The run-time overhead of dynamic quantization comes from quantizing activations as they happen. Therefore, this is advantageous for scenarios where memory bandwidth rather than compute (where the overhead will be introduced) dominates the model execution time. For LSTM and Transformer type models, this is accurate.

For reducing the computational requirements and the memory of the LLMs, quantization is an effective technique for that, as it makes them more efficient to run on varieties of hardware. But the drawback is that quantization may result in a loss of accuracy when compared to full-precision models. So, it becomes of utmost importance to evaluate carefully between accuracy and memory when using quantization.

Here are the steps for performing model quantization:

Choose the precision Choosing the correct precision can happen by determining the target precision for the activations and the model weights. The basic precisions include 4-bit, 8-bit, or even binary weights.

Train the model LLM model should be trained using the full precision technique as usual.

Quantize the model By using low-precision data types, such as 8-bit integers instead of the more common 32-bit floating point, to represent the weights and activations, a technique known as quantization can lower the computational and memory costs associated with conducting inference. This makes operations like matrix multiplication much faster with integer arithmetic.

Calibration Calibration data is used to collect the dynamic ranges of the weights, biases, and activation in the convolution and fully connected layers of the network. Doing this helps in determining the scaling factor that is used to represent the weight and activation at the chosen precision.

Fine-tuning The accuracy loss during quantization is restored by fin-tuning the model again on the training dataset using the weights and activations at full precision.

Validation The quantized model is then validated with a completely different test set to measure the model performance.

Table 1: Accuracy performance after quantization [5].

Model	FP32	INT8 (w/o SmoothQuant)	INT8 (WI SmoothQuant)	INT8 (WI SmoothQuant auto-tuning)
bigscience/bloom-560m	65.20%	63.44%	66.48% (alpha=0.5)	64.76% (alpha: 95.9% over 0.6, 4.1% in [0.4,0.6])
bigscience/bloom-Ib7	71.43%	67.78%	72.56% (alpha=0.5)	72.58% (alpha: 55.1% over 0.6, 30.6% in [0.4, 0.6], 14.3% under 0.4)
bigscience/bIoom-3b	73.97%	69.99%	74.02% (alpha=0.5)	74.16% (alpha: 100% over 0.6)
bigscience/bIoom-7b1	77.44%	75.46%	77.02%(alpha=0.5)	77.45% (alpha: 91.8% over 0.6, 4.9% in [0.4, 0.6], 3.3% under 0.4)
bigscience/bIoom-176b	84.17%	82.13%	83.52% (alpha=0.6)	-
facebook/opt-125m	63.89%	63.48%	63.44% (alpha=0.5)	64.14% (alpha: 59.4% over 0.6, 8.1% in [0.4, 0.6], 32.4% under 0.4)

facebook/opt-1.3b	75.41%	73.59%	70.94% (alpha=0.5)	74.80% (alpha: 69.9% over 0.6, 24.7% in [0.4, 0.6], 5.5% under 0.4)
facebook/opt-2.7b	77.79%	78.57%	78.60%(alpha=0.5)	78.25% (alpha: 73.2% over 0.6, 21.6% in [0.4, 0.6], 5.2% under 0.4)
facebook/opt-6.7b	81.26%	76.65%	81.58%(alpha=0.5)	81.39% (alpha: 68.0% over 0.6, 26.8% in [0.4, 0.6], 5.2% under 0.4)
EleutherAl/gpt-j-6B	79.17%	78.82%	78.84%(alpha=0.6)	79.29% (alpha: 96.4% over 0.6, 3.6% in [0.4, 0.6])

3.3 Pruning

In the pruning technique all the less important parameters are removed from the neural network. This helps in reducing the model size and the computational requirements [3]. Pruning makes the model capable of running on resource constraint hardware or training large models with the same resources.

Weight pruning Weight pruning is responsible for identifying and removing the smallest connections of weights in the LLM. This is done by either removing the small weights entirely or setting the values of small weights to zero.

Neuron pruning The neurons which are least important for predictions are removed by using neuron pruning. Finding the least significant neurons in an LLM may depend in part on the neurons' activity values or gradients during training. Structured pruning: In this technique, the entire structure—for example, the complete filter layer from a convolutional layer—or a collection of parameters from the LLM are deleted [4].

A procedure called pruning can be applied in one of two ways: either during training or after training. Pruning is used in conjunction with regularization approaches during training to help the model learn more quickly and effectively while retaining key parameters [9]. After training, the pruning strategy can be used to eliminate some of the elements that are less crucial for perfect predictions with improved accuracy.

Pruning is a useful approach that reduces the amount of computing power needed to run and store LLMs; yet, it has the potential to reduce accuracy when compared to unpruned models. The evaluation should be taken carefully for the trade-off between the resource requirements and the accuracy that is necessary when using the pruning technique.

Here are the steps for performing pruning:

Train the model LLM model should be trained with full precision technique, as usual.

Determine the importance of weights or neurons The importance of each neuron and the weights in this model should be calculated using this technique. like structure pruning, weight magnitude, sensitivity-based

pruning, or pruning. This way the least important neurons or weights can be removed without significant loss in accuracy.

Remove unimportant weights or neurons This is accomplished by making the network sparse and setting the least important parameters to zero. By doing this, the model's parameters would be reduced yet the architecture would remain unchanged. It is also possible to accomplish this by eliminating the entire node from the network, which would result in a reduced network architecture while still preserving the accuracy of the original larger network.

Fine-tuning To restore the accuracy that is lost during pruning the model is fine-tuned again using the remaining neurons and the weights, while parallelly fixing the pruned weights or neurons [6].

Validation Finally, at last, the pruned model needs to be validated on a validation dataset to ensure that the accuracy is maintained.

From Figure 2 we see the F1 of the full set of pruned networks against the speedup, we can see that it outperforms fine-tuned Tiny BERT and Distilbert by some margin. Even without the optimization version shown here, which lacks the LayerNorm optimization used in the much quicker version of MobileBERT, MobileBert appears to be significantly better.

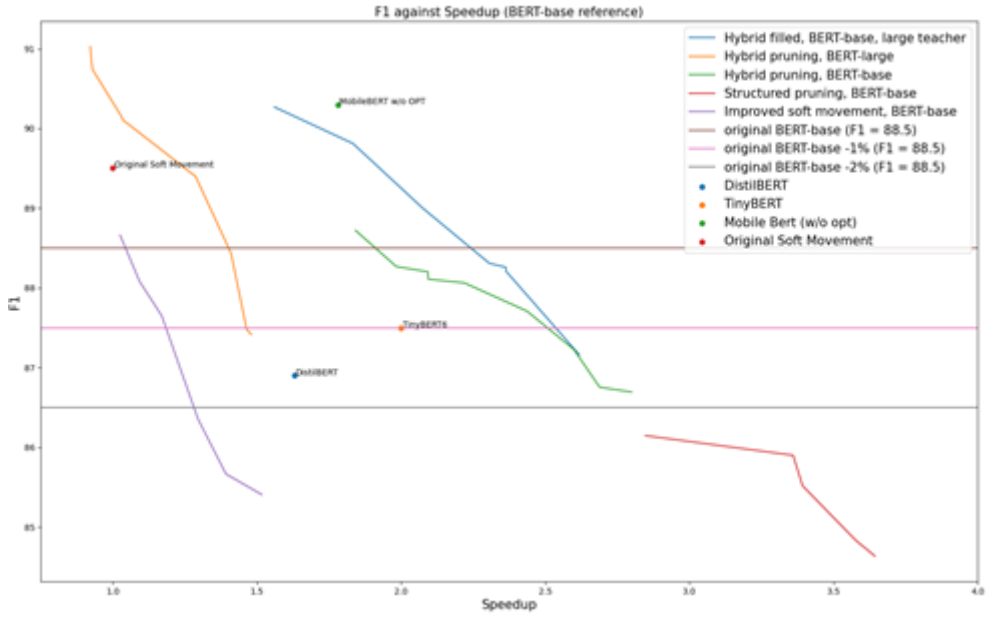

Figure 2: F1 against speedup for BERT-based models [7, 8].

Even in terms of saved size, we get smaller networks for the same accuracy, except for MobileBERT, which is better in size too.

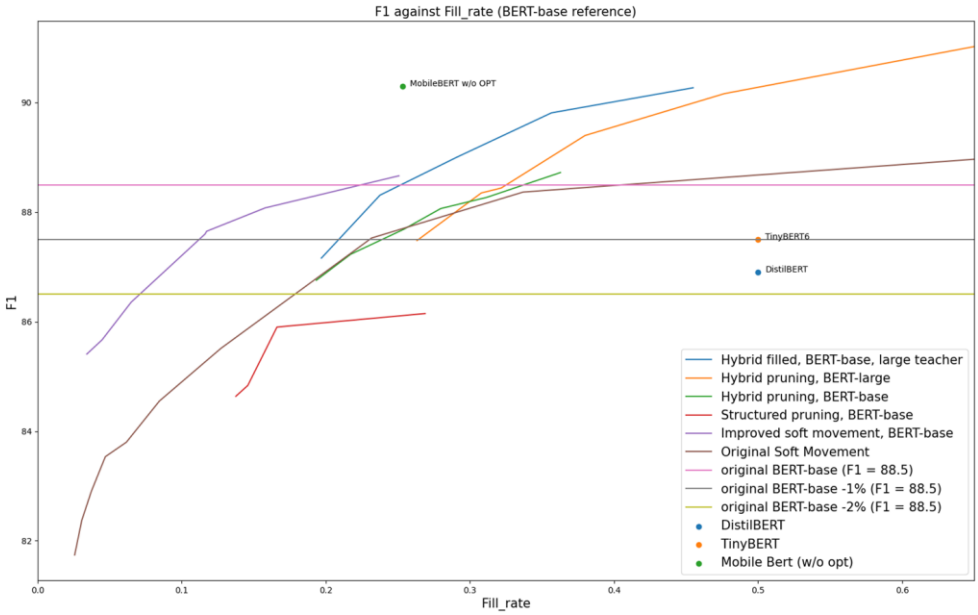

Figure 3: F1 against fill rate for BERT-based models [7, 8].

3.4 Knowledge Distillation

The knowledge distillation technique trains the model in a smaller, basically, a "student" model that helps to mimic the behavior of the larger or a "teacher" model. The teacher model is typically larger and is more accurate LLM as it has been pre-trained on the larger dataset, whereas the student model is typically a smaller model that is being trained to replicate the behavior of the teacher model. Knowledge distillation is the best technique that is being used to reduce the computational resources required to run and store the LLM (large language model)

During training the soft targets are provided to the teacher model, which guides the training of the student model [11]. The soft targets represent the set of probabilities for confidence in the predictions from the teacher's model. After that, the student model is trained to predict the same probabilities as that of the teacher model. In this, the correct labels or the hard targets are being used in traditional training.

Soft targets provide a lot of additional information to the student model that is not present in the hard targets. This allows the student's model to learn more effectively and efficiently. Student models can easily learn to replicate the behavior of the teacher models using very few parameters, this is achieved by using the soft targets. It helps widely in reducing the computational resources required to run and store the model.

Other techniques such as quantization and the pruning technique can combine with Knowledge distillation to reduce the computational resources that are required to run and store the LLM. The student model is less accurate as compared to the teacher's model, as it may not achieve that level of accuracy. Therefore, the tradeoff should be done very carefully between the computational resources and the accuracy while using knowledge distillation.

Here are the steps for performing knowledge distillation:

Train the teacher model The larger model should be trained first, which is referred to as a teacher model, it should be trained using full precision technique, as usual.

Generate soft targets Soft targets are generated by the teacher model, that is the probability distribution over the classes, and are being used as the subset of the training data.

Train the student model Now the simpler and the smaller LLM to be trained is generally referred to as a student model. That is used to mimic the behavior of the larger model which is the teacher model. The student model has been trained in such a manner that the output is similar to probability distributions to the soft targets that are generated by the teacher model.

Fine-tuning After the student model is trained using the soft targets, the fine-tuning should be done using the actual training data, it is just to ensure that the accuracy of the model is not at all compromised.

Validation Finally, at last, the student model on a validation dataset had to be validated to ensure that the accuracy is maintained.

3.5 Approximation Techniques

It is possible to approximate the original LLM with a simpler, smaller, and more computationally efficient model, without compromising the accuracy of the model. This is done by Approximation Techniques. Approximation techniques in LLM (large language models) generally refer to a family of methods that aim to reduce the computational resources that are required to run and store an LLM, without compromising its accuracy.

There are varieties of approximation techniques, some of which are listed below:

Compression-based methods The LLM model can be compressed by lowering the size of its embeddings and its parameters by using techniques like quantization, pruning, and knowledge distillation.

Low-rank approximation The model parameter count can be lowered by approximating the weight matrices of the LLM. This method improves the memory and processing efficiency.

Factorization-based methods In this method, the model parameters are reduced by factorizing the weight matrices of the LLM into smaller matrices. Hence, it would require less computational power to run the model.

Parameter sharing This method allows for a reduction in the number of parameters and the resources needed for computation by sharing the parameters among various LLM components.

Tensor train decomposition In this method, the weight tensors of the LLM model are decomposed into a series of smaller tensor cores, which further can be computed with fewer parameters or independently.

Weight sharing In this technique, the weight of the LLM model is shared across different positions or layers in the input sequence, thus helping in reducing the number of parameters required in the LLM.

Knowledge distillation In this technique, the smaller and more computationally efficient LLM model is trained by using the predictions of a larger and more accurate LLM model as targets during training.

Approximation techniques are one of the techniques that help in reducing the computational resources required to run and store LLMs, thus it becomes possible to train the larger models or to run the heavy models on less powerful hardware. One of the drawbacks is that there will be a significant loss in accuracy as compared to the original.

3.6 Conversion to ONNX

ONNX is designed in such a manner that they are software and hardware agnostic, which means it can be run on a variety of platforms and devices that includes GPUs, CPUs, and specialized accelerators like ASICs and FPGAs. Converting LLM (large language model) to the ONNX format helps in reducing the use of resources used in computation by enabling the model that can run efficiently on different software and hardware platforms. The accessibility of the hardware-specific optimizations, which are not available in other forms, is one benefit of converting LLM to ONNX format. For example, Some hardware accelerators may support specialized operations or data types that can improve the performance of different types of models. Without having to rewrite the model code, these benefits can be obtained using the ONNX approach. Additionally, The models in ONNX are typically more compact as compared to other formats, which can help in reducing the amount of storage and memory required to run and store the model.

4. Conclusion

Using the model optimization techniques, it is possible to leverage the capabilities of LLM in a resource-constraint environment. Quantization in most of the cases reduced the model size by atleast half. It achieves a compression ratio of atleast 50% with only a minimal impact on the performance of the LLM. Converting the quantized model to ONNX made the inference time almost 1.5 times faster, enabling efficient execution across different hardware platforms. Pruning reduces the computational requirements of large language models, resulting in faster inference without significant loss in accuracy. Knowledge distillation illustrates that it can reduce the computational demands of LLM, while maintaining comparable performance to the teacher model. Model architecture optimization techniques, such as efficient attention mechanisms and parameter sharing, improved computational efficiency without sacrificing model performance.

However, there are a few limitations as too much of the model can be optimized. There is a trade-off between model size reduction and maintaining desirable performance, as techniques like quantization or pruning may lead to accuracy degradation if not calibrated properly. Additionally, certain optimization methods can increase model complexity, making interpretation and explainability more challenging. Ensuring models generalize well to diverse data and exhibit robustness is an ongoing concern. Ethical considerations, including biases and fairness, must be taken into account throughout the optimization process. Overcoming these limitations demands

continuous research, collaboration, and a comprehensive approach that addresses computational, data, interpretability, generalization, and ethical aspects.

5. Future of LLM

After the release of ChatGpt in 2022, people's interest are increasing a lot towards LLM. A lot of industries have been transformed a lot by LLM, because of the ability of LLM, such as generation of human text, and addressing a variety of applications. Due to the massive quantity of data that LLM has been trained on, its present performance is impressive and they have significantly altered the way that natural language processing behaves.

Because of how well LLM performs on a variety of tasks, including text summarization, question answering, code generation, emotion analysis, translation, etc., it has recently attracted a lot of interest. LLMs have shown an impact in different fields. Here are some examples:

- **Advancement of Healthcare:** Large language model has impacted the healthcare industry a lot by providing personalized human-centric care, and also by promoting patient adherence and engagement. It can also help in analyzing the patient data as well as the medical record.
- **Enhancement of Customer Service:** Large Language models can help in generating a good analysis of the feedback received from the customers, and can also help in improving the overall experience of the customers.
- **Improving Education:** Large language models can help in providing a proper structure for discussions, personalized guidance to the students at the time of discussions, and also in generating real-time feedback. Overall, this will help in improving the engagement and participation of the students.
- **Enhancement in Business Operations:** As amazing insights and analytics can be generated by the large language model from the feedback provided by the customer, it helps the business to take proper decisions at the correct time.
- **Advancement of scientific research:** It can help in suggesting better experiments and also helps in generating the correct hypothesis, also can be used in analyzing different kinds of patterns, analyzing the scientific data.
- **Focus on Ethical Issues and Social Impact:** LLM programmes may place a higher emphasis on ethical issues and social impact in an era of rising social consciousness and emphasis on corporate social responsibility. As the legal profession and society's interests and values change, courses on legal ethics, sustainability, human rights, and social justice may become increasingly prevalent.

Acknowledgments

We would like to extend our sincere gratitude to each and everyone who has made a contribution to the field of Machine Learning and Artificial Intelligence. Their diligence and commitment considerably improved the study's findings.

References

[1] Nina Shenker-Tauris, "Do Large Language Models (LLMs) reason?". Internet: https://www.shaped.ai/blog/do-large-language-models-llms-reason, Feb. 2023.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, et al., "Attention is All You Need", *arXiv arXiv:1706.03762,* Jun. 2017.

[3] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, John Guttag. "What is the state of neural network pruning?." *arXiv preprint arXiv:2003.03033,* Mar. 2020.

[4] Han, Song, et al. "Learning both weights and connections for efficient neural networks." *arXiv arXiv:1506.02626,* 2015.

[5] Xinhe, Chen Suyue, et al. "Neural Compressor". Internet: https://github.com/intel/neural-compressor/blob/master/docs/source/smooth_quant.md, Mar. 29, 2023.

[6] Molchanov, Pavlo, et al. "Pruning convolutional neural networks for resource efficient inference." *arXiv preprint arXiv:1611.06440,* 2016.

[7] Babaeizadeh, Mohammad, Paris Smaragdis, and Roy H. Campbell. "Noiseout: A simple way to prune neural networks." *arXiv preprint arXiv:1611.06211,* 2016.

[8] François Lagunas, Ella Charlaix, "NN pruning". Internet: https://github.com/huggingface/nn_pruning, Aug. 30, 2021.

[9] Elias Frantar, Dan Alistarh, "SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot" *arxiv:2301.00774,* 2023

[10] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, Dan Alistarh, "OPTQ: Accurate Quantization for Generative Pre-trained Transformers", *ICLR 2023 poster,* 2023

[11] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, et al. "Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes", *arXiv arXiv:2305.02301,* May 2023.

[12] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, Dan Alistarh, "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers", *arXiv arXiv:2210.17323,* Oct. 2022.