EFFICIENTLY RUN A DEEP LEARNING MODEL ON A DISTRIBUTED CPU SYSTEM

D.D.K.K. Gnanasena

218524K

Degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

Faculty of Information Technology

University of Moratuwa

Sri Lanka

April 2024

EFFICIENTLY RUN A DEEP LEARNING MODEL ON A DISTRIBUTED CPU SYSTEM

D.D.K.K. Gnanasena

218524K

Thesis/Dissertation submitted in partial fulfillment of the requirements for the degree

Degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

Faculty of Information Technology

University of Moratuwa

Sri Lanka

April 2024

DECLARATION

I declare that this is my own work and this thesis/dissertation does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date:2024-04-26

The above candidate has carried out research for the PhD/MPhil/Masters thesis/dissertation under my supervision. I confirm that the declaration made above by the student is true and correct.

Name of Supervisor: Dr. Subha Fernando

Signature of the Supervisor: Date:

ACKNOWLEDGEMENT

At the very first I take this opportunity to thank my research project supervisor Dr. Subha Fernando for guiding me throughout my research project by giving advices, opinions & suggestions as well as by encouraging me. Her expertise, knowledge, and unwavering commitment have been instrumental in helping me shape my ideas, refine my research questions, and develop my analytical skills.

In addition, I'm grateful to my friends, family, and coworkers for providing a supportive and cooperative work environment for me. Their enthusiasm, encouragement, and comments have been invaluable in helping me with this research.

I want to thank everyone who has contributed to this work.

ABSTRACT

In Distributed Deep Learning, Data transfer between Computer equipment might act as a bottleneck that prevents the system from growing to its full capacity. To overcome this, people have come with methods like Quantization and Finetuning which work well only on GPUs and TPUs which are costly infrastructure. However, if we could suffice from running deep learning models on CPU based Distributed System, it is a great edge in cost reduction. So, in this research such a system is created and evaluated to find out whether they can compete with GPUs using MultiWorkerMirroredStrategy. MNIST dataset on a linear model, CIFAR10 dataset on a Convolution Neural Network (CNN) model and finally a quantized Llama 2 model was run to summarize and translate an essay. They were evaluated by their accuracy and time taken to train against running them in a Google Colab T4 GPU(For Llama model, BertScore and Rouge Score for summarization and BertScore and BleuScore for Translation was used). Performance of MNIST data concluded, this system is inefficient for small models as the time taken to train on two machines was 200 times of a single machine. When running CIFAR10 dataset CNN accuracy decreases over the number of machines on multi-node and the increasing batch sizes and time taken decreases and increased after a certain batch size showing the bottleneck of communication. In Llama model it was observed that with increasing dataset size the time taken also reduces when compared with GPU time taken. None of the models were fast enough to pass the time taken for the GPU in all cases but some accuracies in Llama model were able to pass GPU accuracies. For future works Model parallelism should also be tested for further development of this CPU Distributed system.

Keywords: MultiWorkerMirroredStrategy, multi-node, CPU, Quantization, Finetunning

TABLE OF CONTENTS

List of Figures

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
DDL	Distributed Deep Learning
CNN	Convolutional Neural Networks

CHAPTER 1

INTRODUCTION

1.1 Prolegomena.

Deep learning (DL) has advanced significantly in recent years, providing cutting-edge outcomes in a number of fields, including natural language processing and image recognition [1]. Large datasets and various model parameters are needed for higher accuracy. Greater precision is achieved with larger datasets. However, significant training time restrictions are seen as dataset sizes increase. It has been noted in the past that using more modern GPUs can result in higher accuracy; nevertheless, When a model is larger than usual or cannot fit into the GPU memory, GPU restrictions may occur. To solve this issue, numerous distributed and parallel training strategies have been employed [2]. However, data transit between computer equipment could act as a barrier, keeping the system from expanding to its maximum potential. Numerous studies suggest lossy compression during communication as a solution to this issue [3]. Particularly for large language models (LLMs), which require a lot of computation and memory, quantization has sped up inference and decreased memory usage [4]. Additionally, these LLMs have been fine-tuned so that the closed-source restriction and hefty development expenses do not hinder them. All of this, though, has only been done in GPUs or TPUs, which are expensive pieces of hardware in and of themselves. Through the use of methods like quantization and finetuning, we have developed a distributed CPU system in this study that is capable of running a deep learning model effectively.

This chapter is organized to present the following: objectives, background, and motivation, Problem in brief, Proposed Solution, Resource requirements and the structure of the thesis.

1.2 Objectives

The following objectives have been identified to develop a CPU Distributed System to address our research problem.

- Critical review of Distributed Deep Learning
- In depth study of technologies used in Distributed Deep Learning
- Design and implement a CPU distributed system to run DL model
- Evaluate the CPU Distributed System.

1.3 Background and Motivation

The research paper by Agarap and Abien Fred explains how they trained deep neural networks on single and dual node (distributed) systems using TensorFlow's MultiWorkerMirroredStrategy. For a single node, they used a configuration consisting of an Intel Xeon E5 v3 (Haswell) processor with eight cores, 30 GB of RAM, and an NVIDIA Tesla V100 16GB HBM2. For multi-node, they only needed two identical machines. They trained a feed-forward neural network with two hidden layers and 512 neurons each with ReLU activation, using a pertubed MNIST dataset. A softmax layer was then added. It takes 2 minutes and 18 seconds to train on one computer, and 1 minute and 57 seconds to train on two. Second, in 62 minutes and 43 seconds on a single machine and 24 minutes and 38 seconds on a multi-machine, they trained a 56-layer version of the ResNet architecture using the CIFAR-10 dataset [1].

In the paper by Hang Xu, Chen-Yu Ho, Ahmed M. Abdelmoniem, Aritra Dutta, El Houcine Bergou, Konstantinos Karatsenidis, Marco Canini and Panos Kalnis, Along with a clear classification, For DNN training, they provide an in-depth examination of the most widely used compressed communication methods. Four tasks of machine learning image classification, segmentation, recommendation, and language modeling were covered by their findings. As a baseline for comparison, they did not utilize any compression. Additionally, they discovered that in language models, spasification can become unfeasible. [2].

A novel adaptive quantization strategy (AdaQS) is presented by Jinrong Guo, Wantao Liu, Wang Wang, Jizhong Han, Ruixuan Li, Yijun Lu, and Songlin Hu to look into how Gradient Quantization might be used to balance model accuracy with quantization level. They conduct experiments on a distributed cluster consisting of four servers with four NVIDIA Tesla M40 GPUs connected by 10Gbps Ethernet, utilizing the Cifar10 and ImageNet datasets. Test results show that AdaQS greatly lowers the communication cost of 32-bit vanilla SGD and improves accuracy on very deep models. Furthermore, AdaQS's speedup performance is further enhanced by the extremely minimal overhead associated with calculating the quantization level thanks to the utilization of moment estimate [3].

. The major obstacles to reducing the ubiquity of instruction models are their expensive development costs and closed-source restrictions. Many suggest fine-tuning an LLM, i.e., LLaMA into an instruction-following model, which is extremely expensive; In order to maintain the model parameter, gradient, and optimizer information, for example, fine-tuning an LLaMA-65B model with AdamW requires more than 1TB of GPU memory. This approach is repeatable and economical. Low-rank adaptation (LoRA) approach can greatly minimize the number of trainable parameters for fine-tuning large language models (LLMs). LoRA is widely used in many different applications because of its shown capacity to perform on par with full-parameter fine-tuning. [4].

1.4 Problem Definition

Though DDL has been efficiently carried out passing the bottlenecks of communication, they still run efficiently only on GPUs which in itself are too costly infrastructure.

1.5 Approach of the CPU Distributed System

We have hypothesized that through Quantization and Finetuning a model we can efficiently run a DL model in a Distributed CPU system. This idea was inspired due to the fact that GPUs are too costly and CPUs are easy to access and even in GPUs to run LLMs we need large GPU clusters which are expensive.

As the input we plan to input the Training data on the LLaMa model.

Our final output is expected to have a CPU Distributed System that can run DL models like LLMs efficiently

Process of the CPU Distributed System is based on several steps. For the training data we are planning to take the data of LLaMa Model. First using Multi Worker Mirrored Strategy of Tensorflow with Keras API, we plan to create a CPU distributed System using several machines either in cloud using Microsoft Azure VMs/Amazon AWS VMs or in a Lab. We will try to run the LLaMa model by Quantizing through Quantization techniques like Gradient Quantization and other Quantization techniques. We then intend to fine-tune the LLaMa model further and attempt to raise the system's efficiency by employing the LoRA fine-tuning method. Finally, we do comparative analysis on the performance on the model with a GPU based on concerning accuracy and speed as well as cost of building such a system.

Users of the CPU distributed system are any student or Researcher plan to run a DL model

Features of the CPU Distributed System is to efficiently run a DL model in the system with accuracy and speed just like GPUs run in DDL.

1.6.1 Resource Requirement

1.6.1 Hardware

- Intel Core i5 Computer CPU 2.4 GHz and 8GB RAM
- Intel Core i7 Computers CPU 3.6GHz and 8GB RAM * 20
- Microsoft Azure VMs
- Amazon AWS VMs

1.6.2 Good Internet Access

1.7 Organizational Structure of the Thesis

Below is the structure of the thesis: Chapter 2 offers a critical analysis of the literature on distributed systems, and Chapter 3 looks at the technologies employed to solve the research problem. Chapter 4 presents the methodology for creating the desired model. The design and implementation are the main topics of chapters 5 and 6, respectively. The evaluation of the developed model is covered in Chapter 7. The study's conclusions and future works are outlined in Chapter 8, which also includes appendices and references.

1.8 Summary

This chapter covered the significance of using distributed CPU systems, as well as earlier research and upcoming difficulties in the field. The research problem we are attempting to solve is Though DDL has been efficiently carried out passing the bottlenecks of communication, they still run efficiently only on GPUs which in itself are too costly infrastructure.

CHAPTER 2

A LITERATURE REVIEW ON DISTRIBUTED SYSTEMS

2.1 Introduction

The research topic and the main goals of our study, which are to answer the research problem, were covered in the overview of this thesis that was provided in the prior Chapter. We also provided an outline of our proposed approach to resolving the issue. This chapter gives a comprehensive literature review in the area of Distributed Systems. The literature review has been structured to cover the overview of Distributed Systems; AI Frameworks used for Distributed Systems; Current applications of Distributed Systems and The Challenge in Distributed Systems. Here we have also summarized the Literature Review and defined our research problem.

2.2 Overview of Distributed Systems

A vast array of practical applications has demonstrated the great potential of deep learning and unsupervised feature learning. In a number of areas, including text processing, speech recognition, and visual object recognition, state-of-the-art performance has been reported [5]. The utilization of more sophisticated software, such deep neural networks (DNNs), and more potent hardware, like GPUs, has been crucial to these astounding developments in machine learning. High accuracy can be achieved with DNN inference, albeit at the expense of longer execution times (latency) and higher energy usage. With deep learning at the edge, these problems are becoming more noticeable [6].

Recent developments in a broad range of Deep Learning (DL) applications have been spurred by the availability of large-scale datasets and abundant computing capacity. Deep Neural Network (DNN) training is a laborious, iterative process. In the absence of a reasonable training time, the slope of improvements reduces. Because of this, a distributed learning framework needs to be able to train the DNN over several nodes and effectively use the resources on each node. The memory demands of modern DNNs can never be met by a single node, not even by the strongest one. So we must take use of every opportunity to scale up (on a single node) and scale out (on multiple nodes) the training process in order to obtain a suitable execution time for training contemporary models on large-size datasets [7].

This presents several difficulties. Initially, it is important to make efficient use of the processing resources; that is, to prevent expensive GPU resources from stalling because of communication constraints. Secondly, in order to save costs and offer flexibility, the network, storage, and compute resources are usually distributed across several users or training procedures. DL offers numerous parallelization opportunities. Here, three popular techniques for DL parallelization were presented: hybrid forms of

parallelism, data, model, and pipeline parallelism. Several workers (computers or other devices, such as GPUs) load duplicate copies of the DL model in data parallelism. To train the worker model copies, the training data is divided into non-overlapping segments. The model parameters are updated by each worker's training on its own subset of training data. It is therefore imperative that all personnel synchronize the model parameters. The key benefit of data parallelism is that it can be used with any kind of DL model architecture and doesn't require any additional knowledge of the model domain. For computation-intensive tasks with few parameters, like CNNs, it scales nicely [8].

In order to implement model parallelism, the model is divided among several computing resources, each of which handles a different portion of the model's computation. When training enormous models that are too big to fit in the memory of a single computing unit, this method is quite helpful. It is possible to build model parallelism with methods like tensor partitioning and pipelining [9].

2.3 AI Frameworks used for Distributed Systems

As a result of the quick development and use of AI technologies, many AI frameworks have been created, and the quantity of cloud services tailored for AI applications has grown. Several popular artificial intelligence (AI) frameworks include TensorFlow, PyTorch, Apache MXNet, Microsoft's Cognitive Toolkit (CNTK), and Caffe. A variety of factors, such as ecosystem, performance, scalability, and user-friendliness, should be taken into consideration while comparing AI frameworks. Right now, TensorFlow and PyTorch are the most widely used options; TensorFlow offers superior performance and a more developed ecosystem, while PyTorch offers more flexibility and a more user-friendly development environment [9].

The research article by Shaohuai Shi, Qiang Wang, and Xiaowen Chu assesses the runtime performance of four cutting-edge distributed deep learning frameworks in single-GPU, multi-GPU, and multi-node scenarios: Caffe-MPI, CNTK, MXNet, and TensorFlow. They start by building performance models of standard protocols for SGD-based DNN training. Subsequently, they evaluate these frameworks' operational performance against three popular convolutional neural networks (ResNet-50, AlexNet, and GoogleNet). Finally, they look into the factors that affect how well these four frameworks perform differently from one another. The overall scaling efficiencies of CNTK in AlexNet, GoogleNet, and ResNet-50 are approximately 55%, 67.5%, and 77.7%; for MXNet, the corresponding efficiencies are 35.625%, 65.625%, and 35.6%; and for TensorFlow, the corresponding efficiencies, when training on four machines, are 50.625%, 75.56%, and 52.187%. [10].

Different approaches were introduced in the form of practical APIs by TensorFlow and PyTorch. Tensorflow contains four distinct strategies: the Mirror Strategy, Multimirror Strategy, TPU Strategy, and Parameter Server Strategy. Similar to this,

PyTorch has three strategies: RPC-Based Distributed Training, Distributed Data Parallel Training, and Collective Communication [11].

In the research paper by Ovidiu-Constantin Novac, Mihai Cristian Chirodea, Cornelia Mihaela Novac, Nicu Bizon, Mihai Oproescu, Ovidiu Petru Stan, and Cornelia Emilia Gordan PyTorch and Tensorflow frameworks were compared using a AMD Ryzen 7 1700X, 3.80 GHz CPU having a ASUS GeForce GTX 1070 Ti A8G, 8 GB GDDR5 GPU with 16 GB 3200 MHz RAM and 500GB SSD running Windows 10 Pro version 20H2 and ned, Using PyCharm Professional Edition with the 2020.1.3 version of the Anaconda plugin and Windows 10 Pro, version 20H2, as the integrated development environment, the system was assessed. It was found that PyTorch was much faster more than 25% than TensorFlow. However regarding accuracy PyTorch performs poor than TensorFlow [12].

2.4 Current applications of Distributed Systems

Nguyen, Tri DT, Jae Ho Park, Md Imtiaz Hossain, Md Delowar Hossain, Seung-Jin Lee, Jin Woong Jang, Seo Hui Jo, Luan NT Huynh, Trong Khanh Tran, and Eui-Nam Huh use data parallelism in their research paper to develop an assistive living Human Activity Recognition (HAR) application using LSTM model. The program can detect and forecast changes in human activity patterns in real time because it is installed in containers inside of a Kubernetes cluster. They specifically assessed the performance of a distributed training system for the HAR application, which runs on several CPUs and GPUs in containers, in contrast to the host system. To make things simpler, they made fixes to the UCI HAR dataset and an LSTM 2 layers network. Furthermore, the container environment and local machine for distributed training were adjusted. The local PC was equipped with an Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz with 4 cores, while the server was equipped with an Intel(R) Xeon(R) Gold 510 CPU @ 2.20GHz with 56 cores and GPU GeForce RTX. There are 4000 epochs in total. The results showed that using the leverage data parallelism strategy instead of containers saves a significant amount of time. The training loss curve was evaluated with a range of dockers. Every container had a set number of epochs, 2000 in all cases. The result showed that the learning model's convergence can be guaranteed by the data parallelism technique. Thus, combining containers with a data parallelism technique expedites training while also enhancing scalability. A sizable dataset will be used in the future to expand the system's capabilities. [13].

The study conducted by Sidi Ahmed Mahmoudi, Saïd Mahmoudi, and Jean-Sébastien Lerat presents an empirical method for evaluating the DDL speedup attained by utilizing various parallelism techniques on the nodes. They assess how much faster training of DL networks with varying levels of complexity and dataset sizes may achieve. Convolutional neural networks (CNNs), which retain their neural network type generality while being perfectly tailored for classification issues involving images and videos are the subject of the current work. The publicly accessible dataset was

provided by the Department of Computer Sciences, Faculty of Engineering, University of Mons. There are 791 images in the dataset's small version and 6003 images in its large form. These can be divided into three groups: smoke, fire, and no fire. A Deep Learning model for image-based fire or smoke detection is constructed using this dataset. The two use cases are as follows: ComplexSmall utilizes the VGG16 CNN architecture for the tiny dataset, while SimpleBig uses the AlexNet CNN architecture for the large dataset. The Spark framework, the Horovod API—which was created specifically for DDL—and the suggested method for dividing the load with MPI are the three ways that the DDL process is carried out. In terms of the faster outcomes produced by distributed computing systems. Since Spark does not support the GPU natively, it can only be run on the CPU. Clearly, a single node parallelized version outperforms the Spark approach in terms of efficiency. The SimpleBig use case even experiences a negative acceleration. It gets more difficult for Spark to process data rapidly the more there are. Horovod can outperform Spark and accelerate the process in every scenario, but it is unable to cut the calculation time in half. The volume of network traffic was the cause of this. The recommended approach avoids network communication, which transfers data more slowly than data exchanged within RAM. Less network traffic is required thanks to computer parallelism optimization, which speeds up the DDL's data processing. The MPI version that is employed outperforms Horovod and offers the best speedup. The findings imply that, for small datasets, DDL might be slower than pure parallelism. Multi-process data parallelism is quicker on the CPU while running on a single machine, while model parallelism is faster on the GPU. The approach of this study will be extended to cloud and edge computing infrastructures in the future, and a proposed distributed deep learning deployment strategy will be made [14].

Agarap and Abien Fred's research article describes how they trained deep neural networks using TensorFlow's MultiWorkerMirroredStrategy on both single and dual node (distributed) systems. gRPC is then used as the communication layer. For their research, they used two datasets for image classification. The MNIST dataset for the feed-forward neural network, and the CIFAR-10 dataset for the 56-layer ResNet. They employed a setup with an Intel Xeon E5 v3 (Haswell) processor with eight cores, 30 GB of RAM, and an NVIDIA Tesla V100 16GB HBM2 for a single node. Only one GPU is used for computing in this configuration. Additionally, because to budgetary and logistical constraints, they only used two similar machines for multi-node. The machines that were utilized were set up on the Google Cloud Platform's Compute Engine. They trained a two hidden-layer feed-forward neural network, each with 512 neurons with ReLU activation, using a pertubed MNIST dataset. A softmax layer was then added. It takes 2 minutes and 18 seconds to train on one computer, and 1 minute and 57 seconds to train on two. Second, in 62 minutes and 43 seconds on a single machine and 24 minutes and 38 seconds on a multi-machine, they trained a 56-layer version of the ResNet architecture using the CIFAR-10 dataset. The variant had batch normalization decay of 0.997, batch normalization epsilon of $1 \times 10{-}5$, and L2 weight decay of $2 \times 10{-}4$. on the CIFAR-10 dataset, and it took 62 minutes and 43 seconds

to train on a single machine while it took 24 minutes and 38 seconds to train on two machines. Ultimately, $62.06 was spent for the training in its whole, including test runs. The complimentary Google Cloud Platform credits were used to cover this expense. In order to further increase the training pace of the models, they propose using different distributed training regimes in future work, such as multiple workers and multiple parameter servers, for training deep neural networks [1].

In their article, Migliorini, Castellotti, Canali, and Zanetti tried to use distributed machine learning to apply to a physics with high energy use case. For Big Data platforms, they provide a distributed deep learning system, which was implemented as Apache Spark libraries. In particular, Analytics Zoo provides an all-in-one platform for analytics and AI that integrates apps like BigDL, Spark, TensorFlow, and Keras into a seamless workflow. and To distribute training among multiple machines, the "Multi Worker Mirrored Strategy" has been used. Oracle Cloud virtual machines (VMs) predicated on Intel Xeon Platinum 8167M CPUs running @ 2.0 GHz are referred to as "VM.Standard2.16" VMs. These computers feature 16 OCPUs, or operational cores, and 240 GB of RAM. The GPU testing was conducted using the virtual machine flavor "GPU 2.1," which has 72 GB of RAM, 12 physical cores (OCPU), and an Nvidia P100 GPU. They discovered that while using the GPU and CPU for training, the Inclusive Classifier requires roughly 2000 seconds to complete 6 epochs of training using 400 CPU cores (spread across 25 virtual machines with 16 physical cores each). This is similar to the training time that they recorded when they divided the training among six GPU-equipped nodes. For training on GPU resources (Nvidia P100), they measured that each batch is processed in about 59 ms, except for epoch 1, which is I/O bound and operates approximately three times slower. Every batch is roughly 7.4 MB in size and has 128 items in it. This corresponds to around 125 MB/sec per node for GPU training data throughput (or 1.2 GB/sec for training with 10 GPUs). During CPU training, the measured processing time each batch is around 930 ms, or 8 MB/sec per node. 90 workers and 480 CPU cores equals 716 MB/sec for the training test. [15].

In their work, Michał Bień, Riccardo Castellotti, and Luca Canali describe how they used cloud resources namely, Spark DL trigger and CMS Big data reduction to successfully implement two high energy physics use cases. They achieved this by utilizing Oracle Cloud Infrastructure (OCI), and in this instance, it has been detailed how TensorFlow was used to utilize both CPU and GPU resources. Additionally, both BigDL and Tensorflow multi worker mirrored strategies have been evaluated with a batch size limit of 128. It was evident that the training times for MultiWorker TensorFlow and BigDL were similar (with a tiny bias towards BigDL since it downloads all the data prior to training, making first-epoch training a worst-case scenario). However, a notable difference in training performance was seen throughout the experiments: the TensorFlow loss function value was plateauing slower and declining at a far faster rate. The backends used by the two frameworks differ greatly. In their Future research will focus on the scalability of TensorFlow

MultiWorkerMirroredStrategy on the cluster of several GPU-enabled cloud instances [16].

In their study, M. H. Anwar, S. Ghafouri, S. S. Gill, and J. Doyle attempted to develop a recommender system to determine the minimum number of clusters required for cloud datacenter Optimal Distributed Deep Learning. For distributed training, they employed the Muti Worker Mirrored Strategy of Tensorflow with Keras and Azure Virtual Machines. They used NetEm to create packet loss and network delays in their clusters, as well as to mimic excessive latency between machines. In our clusters, they replicated latency of 20, 40, 60, 80, and 100 ms. For evaluation, four virtual CPUs with 8GB RAM, 50GB Memory, and vCPUs were employed. Using the MNIST dataset, they assessed the performance of 1, 2, 3, and 4 machine cluster configurations, respectively, with a latency of 0 ms and various batch sizes of 64,000, 32,000, 6400, 2560, 1280, and 192. The result clearly demonstrates that the distributed training time decreases sharply with an increase in the number of machines, so long as the batch size is sufficient to provide for the benefits of distributed training. Clusters with fewer machines perform better than clusters with more machines under smaller batch sizes. They used the CIFAR10 dataset to train the model, mimicking latency times of 20, 40, 60, 80, and 100 ms. Smaller clusters have been shown to outperform bigger clusters under high latency settings since the communication overhead of dispersed training is not offset by the speedup. According to their claims, cluster scaling strategies can speed up the training of deep learning models in scenarios where there are network limits (high latency between computers) caused by machines being geographically apart. These situations are not covered by existing research. They are attempting to look at how to combine their approach with frameworks such as iThermoFog in order to create a comprehensive cluster size recommendation system and train it on different sorts of virtual machines across hundreds of environments. Additionally, they were utilizing several models to gather information on the behavior and training duration of each model in a range of situations. The main goal of this project is to reach global states by parallelizing SGD and utilizing an all-reduce technique. Machine learning algorithms like Random Forest or Reinforcement Learning could be used to train the recommender system's model. Aiming to include accuracy as a factor, the model should suggest a cluster size that maximizes accuracy while requiring the least amount of training time. To make cost-conscious judgments, the model should also take into account the price of each virtual machine. One drawback of distributed SGD is that workers do nothing during the communication phase. They hope to implement decentralized processing in the future. Semantic split neural network models may also be added to the recommender system. These models split weights into a set or hierarchy in order to minimize network overhead. As a result, there would be less network overhead, and a larger cluster might perform better [17].

2.5 Major Challenge in Distributed Systems

Using numerous processors, such as CPUs, GPUs, and Google TPUs, distributed training is becoming increasingly popular as a way to speed up the training process. It makes sense that using several processors to work together to train a single job can shorten the training period overall, but this usually restricts the system's scalability due to communication costs. Even worse, multiple processors may perform worse than a single processor when high-speed computing devices (like Nvidia V100 GPUs) are used to train a deep model over low-speed networks (1/10 Gbps Ethernet, for example) where the computation-to-communication ratio is low. Thus, data communication in distributed deep model training needs to be well-optimized in order to fully leverage the processing capacity of distributed clusters [18].

The study conducted by Anton Sam and Stefan Forsström aims to investigate the variables that affect the distributed training efficiency in Tensorflow. The training time and internet use were then measured by setting up a cluster of three Raspberry Pi 3s and one PC. For evaluation, they employed the mnist dataset with Tensorflow MultiworkerMirrorStrategy. Simultaneously, the accuracy is 94% for 1, 2, 3, and 4 devices. On the other hand, there is an exponential rise in training time. Compared to training on a single machine, training on four machines is 21 times slower. The most obvious explanation for this is that there is too much cost associated with transmitting the training data and updating the model at each training step. All of the data used in this work's tests was sent via WiFi, which is often slower than transferring data via an Ethernet cable and could have resulted in a quicker training period [19].

Because of the high model/gradient aggregation cost and high communication synchronization frequency, the early distributed deep learning models are subjected to a communication bottleneck. In order to overcome communication challenges while ensuring convergence, communication-efficient distributed training algorithms concentrate on four dimensions: communication synchronization, system designs, compression techniques, and parallelism of communication and compute. To mitigate the effects of synchronization and reduce the volume of communications within the same time frame, initial communication synchronization strategies consider reducing the difficult synchronization of Distributed Deep Learning models. Secondly, in order to prevent communication congestion among the PS and workers, some system architectures try to modify the communication topology. Thirdly, the goal of the compression approaches is to shorten communication times by reducing communication traffic and, consequently, communication data. Lastly, using computing and communication parallelism to conceal communication latency and reduce iteration time [18].

To reduce the quantity of data exchanged during communication, several research recommend employing lossy compression. Usually, the stochastic gradient descent (SGD) optimizer is used multiple times throughout the back-propagation phase of DNN training. Training is stochastic by nature, therefore even with the few errors caused by lossy compression, it should be able to converge. The literature discusses

four main categories of compressors: (i) Sparsification, It sends more elements per tensor; (ii) Quantization, which only sends a limited amount of data per tensor (e.g., the top k largest elements); (iii) Hybrid methods, which combine quantization and sparsification; and (iv) Low-rank methods, which break down the gradient into low-rank matrices [2].

In Hang Xu, Chen-Yu Ho, Ahmed M. Abdelmoniem, Aritra Dutta, El Houcine Bergou, Konstantinos Karatsenidis, Marco Canini, and Panos Kalnis' work, they develop 16 lossy compression methods on all four of the aforementioned classes and instantiate their API on TensorFlow and PyTorch. They used datasets from many fields, such as recommendation systems, language modeling, and image classification and segmentation. Additionally, they employed a variety of models, including recurrent neural networks (RNN) and convolutional neural networks (CNN). The dedicated computers ran Ubuntu 18.04.2 LTS with Kernel v.4.15.0-74 and were equipped with a 16-Core Intel Xeon Silver 4112 processor running at 2.6 GHz, 512 GB of RAM, a single NVIDIA Tesla V100 GPU card with 16 GB of on-board memory, and 10 and 25 Gbps network interface cards. They used multiple nodes having a minimum of one NVIDIA Tesla V100 GPU card in the same cluster. They concluded from this that the relative performance of the various approaches is influenced by the DNN architecture. Interestingly, they demonstrate how certain techniques could not be practical in accordance with the communication library that underlies and the computational overhead of decompression and compression (sparsification, for instance, is impractical for language models) [2].

A novel adaptive quantization strategy (AdaQS) is presented by Jinrong Guo, Wantao Liu, Wang Wang, Jizhong Han, Ruixuan Li, Yijun Lu, and Songlin Hu in order to use Gradient Quantization to examine the trade-off between model accuracy and quantization level. They conduct experiments on a distributed cluster consisting of four servers with four NVIDIA Tesla M40 GPUs connected by 10Gbps Ethernet, utilizing the Cifar10 and ImageNet datasets. Test results show that AdaQS greatly lowers the communication cost of 32-bit vanilla SGD and improves accuracy on very deep models. Furthermore, AdaQS's speedup performance is further enhanced by the extremely minimal overhead associated with calculating the quantization level thanks to the utilization of moment estimation [3].

In their work, Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han offer SmoothQuant, a general-purpose, accuracy-preserving, training-free post-training quantization (PTQ) solution. This allows for 8-bit weight, 8-bit activation (W8A8) quantization for LLMs. Serving LLMs requires a lot of energy and money due to their large model size. For example, the GPT-3 model needs at least 350GB of RAM to run in FP16 in order to store its 175B parameters. Accordingly, 5×80GB A100 GPUs or 8×48GB A6000 GPUs are required only for inference. For evaluation, three LLM families are chosen. SmoothQuant: GLM-130B, BLOOM, and OPT. To test the OPT and BLOOM models, they employed seven zero-shot evaluation tasks: WikiText, one language modeling dataset, RTE, HellaSwag PIQA, WinoGrand, OpenBookQA, and COPA. Since some of the aforementioned benchmarks are present

in the GLM-130B model's training set, they employed MMLU, MNLI, QNLI, and LAMBADA to assess the model. They assessed the OPT and BLOOM models using lm-eval-harness, and they assessed the GLM-130B model using its official repository. In the end, they scale up their approach to MT-NLG 530B, allowing a >500B model to be served within a single node for the first time. They reduce the memory footprint and achieve 1.56× inference acceleration by integrating SmoothQuant with PyTorch and FasterTransformer. SmoothQuant provides a turnkey solution to lower the serving cost, democratizing the application of LLMs [20].

A unique fault-tolerant approach for inferencing large language models was introduced by Pavel Samygin, Colin Raffel, Younes Belkada, Max Ryabinin, Artem Chumachenko, Dmitry Baranchuk, Tim Dettmers, and Alexander Borzunov in their work. Additionally, they devised a decentralized methodology that performs substantially better than previous methods for executing inference on consumer-grade hardware while executing LLMs on distributed, unreliable, Internet-connected devices. They evaluate their system with two more realistic tasks: BLOOM (176B) and Llama 2 (70B). They start by taking into account servers operating in a network with regulated latency and bandwidth. They assessed the performance of three different scenarios: (a) Three servers each with a T4 GPU received Llama 2; three servers each with an A100 (80 GB) GPU received BLOOM; and ten servers each with an RTX 3090 GPU received BLOOM. For all evaluations, they employed 8-bit matrix decomposition for BLOOM and 4-bit NormalFloat quantization for Llama 2. And they discovered Performance for inference decreases with increasing latency but is not much impacted by bandwidth or sequence length. In turn, latency and bandwidth have an impact on how well forward passes are tuned for big batches. They showed that the suggested system can grow to include hundreds of billions of trainable parameters, which is the largest language model currently available to the public [21].

2.6 Summary of the Literature Analysis

After conducting a comprehensive review of the problem domain, which is Distributed Systems, we have summarized the key studies analyzing distributed systems in Table 2.1. This table presents the advantages, limitations, and significant contributions of each method to the field.

Table 2.1: SUMMARY OF LITERATURE REVIEW

Paper	Key contribution	CONCLUSIONS	Limitations / future works
[10]	The runtime performance of four cutting-edge distributed deep	Obtained efficiencies for AlexNet, GoogleNet, and ResNet-50 respectively	

	learning frameworks (Caffe-MPI, CNTK, MXNet, and TensorFlow) across single-GPU, multi-GPU, and multi-node settings is assessed	CTNK: 55%, 67.5%, and 77.7%; MXNet:35.625%, 65.625%, and 35.6%; TensorFlow: 50.625%, 75.56%, 52.187% when training on 4 machines	
Ovidiu ,2022	PyTorch and Tensorflow frameworks were compared	PyTorch was much faster more than 25% than TensorFlow. accuracy PyTorch performs poor than TensorFlow	
Nguyen , 2019	develop an assistive living Human Activity Recognition (HAR) application using LSTM model	using containers with a data parallelism method improves scalability while simultaneously speeding up training	a large-scale dataset will be applied to generalize the ability of their system
Mahmoudi, 2022	empirical method for evaluating the DDL speedup attained by utilizing various parallelism techniques on the nodes	for small datasets, DDL may be slower than pure parallelism. Model parallelism is quicker on the GPU while multi-process data parallelism is faster on the CPU when running on a single computer.	methodology will be applied to cloud and edge computing infrastructures, and a distributed deep learning deployment strategy will be suggested
Agarap and Abien Fred, 2019	Evaluate TensorFlow's MultiWorkerMirroredStrategy on both single and dual node (distributed) systems	Decrease in time multi node was observed than single node.	use different distributed training regimes in future work, such as multiple workers and multiple parameter servers,

			for training deep neural networks
M. Migliorini, 2020	use distributed machine learning to a high energy physics use case	GPU is much faster than CPU, spark is slow	
Michał Bień, 2019	used cloud resources namely, Spark DL trigger and CMS Big data reduction to successfully implement two high energy physics use cases. They achieved this by utilizing Oracle Cloud Infrastructure (OCI),	training times for MultiWorker TensorFlow and BigDL were similar, TensorFlow loss function value was plateauing slower and declining at a far faster rate	focus on the scalability of TensorFlow MultiWorkerMirroredStrategy on the cluster of several GPU-enabled cloud instances
M. H. Anwar, 2022	develop a recommender system to determine the minimum number of clusters required for cloud datacenter Optimal Distributed Deep Learning	Clusters with fewer machines perform better than clusters with more machines under smaller batch sizes	implement decentralized processing in the future. Semantic split neural network models may also be added to the recommender system.
Anton Sam and Stefan Forsström, 2021	investigate the variables that affect the distributed training efficiency in Tensorflow.	that there is too much cost associated with transmitting the training data and updating the model at each training step.	
H. Xu *et al, 2020*	evaluate 16 lossy compression methods on all	some methods could not be feasible dependent on the underlying communication	

	four compression methods	library and the computational cost of compression and decompression (for example, sparsification is not feasible for language models)	
Jinrong Guo, 2020	adaptive quantization strategy (AdaQS)	AdaQS greatly lowers the communication cost of 32-bit vanilla SGD and improves accuracy on very deep models.	
Guangxuan Xiao, 2023	SmoothQuant, a general-purpose, accuracy-preserving, training-free post-training quantization (PTQ) solution	allowing a >500B model to be served within a single node for the first time. They reduce the memory footprint and achieve 1.56× inference acceleration by integrating SmoothQuant with PyTorch and FasterTransformer. SmoothQuant provides a turnkey solution to lower the serving cost, democratizing the application of LLMs	
Pavel Samygin, 2023	fault-tolerant approach for inferencing large language models	, latency and bandwidth have an impact on how well forward passes are tuned for big batches. They showed that the suggested system can grow to include hundreds of billions of trainable parameters,	

2.7 Research Problem

In this chapter, we have revealed the current Distributed Systems available in the world. However, though they run efficiently passing the bottlenecks of Communication much of these systems run only on GPUs. And all the CPU distributed

systems found so far are slower than GPU based distributed Systems. GPUs are costly infrastructure itself which sometimes is really a challenge to buy even for startups and small businesses. Even we try to buy a virtual GPU, it should cost more than $68 which is really hard for a student or an individual to buy. Therefore, we define our research problem as though DDL has been efficiently carried out passing the bottlenecks of communication, they still run efficiently only on GPUs which in itself are too costly infrastructure.

2.8 Summary

A thorough review of the literature on distributed systems is given in this section. We talk about the study area, AI frameworks that facilitate DDL, DDL challenges, notable achievements of existing systems, and potential future paths for the field. Furthermore, we formulate the research problem that, even though DDL has been efficiently carried out passing the bottlenecks of communication, they still run efficiently only on GPUs which in itself are too costly infrastructure. We offer a comprehensive examination of the technology employed to solve the research problem in chapter 3.

CHAPTER 3

TECHNOLOGIES ADOPTED FOR DISTRIBUTED SYSTEMS

3.1 Introduction

In the second chapter we did a comprehensive review on Distributed Systems. This chapter will focus on the technologies employed in our research problem domain.

3.2 MultiWorkerMirroredStrategy for CPUs

The all-reduce algorithm is used by the experimental TensorFlow 2.0 multi worker mirrored strategy to maintain neural network variable synchronization throughout training [21]. In other words it's a distributed training strategy provided by TensorFlow for training machine learning models across multiple workers in a synchronous manner.When you have a complicated model that needs a lot of processing power to train, or a big dataset, this method comes in handy. We can combine a number of machines using the MultiworkerMirrored Strategy to increase our total processing power and speed up our training. And in this case the model will be replicated in each and every machine that is connected. It can be CPU or GPU. When you are training a model an amount of data will be send to that device. The gradient will be calculated from it. Then all these gradients from worker will be averaged in each and every devices. Then each worker The gradients from each worker are synchronized across all workers after averaging. By ensuring that each worker is updating their model parameters using the same set of gradients, this synchronization phase allows training to proceed consistently throughout the distributed environment. Communication operations like AllReduce are commonly used in the synchronization of gradients among workers in order to effectively convey information between workers. Once all workers have synchronized their gradient, each worker will change his own copy of the model parameters using the average gradient. Then at each training step, this process of gradient calculation, averaging, synchronization, and parameter updating is repeated until the model converges to an acceptable level of performance [22]. As through this technology we can connect the computers this will be the ideal technology for our research.

3.3 Summary

This chapter has outlined the technology utilized to answer the problem of this reasearch. The upcoming chapter will delve into the approach taken in our project.

CHAPTER 4

AN APPROACH TO IMPLEMENT A CPU DISTRIBUTED SYSTEM

4.1 Introduction

Chapter 3 justified the suitability of MultiWorkerMirroredStrategy as the technology adapted for developing a CPU Distributed System. This chapter presents our approach to implement a CPU distributed system by covering our hypothesis, input, output, process, features, and users. Among others, the *process* specifically identifies the task within our approach which provides the insight into design of the solution.

4.2 Hypothesis

We have hypothesized that through Quantization and Finetuning a model we can efficiently run a DL model in a Distributed CPU system. This idea was inspired due to the fact that GPUs are too costly and CPUs are easy to access and even in GPUs to run LLMs we need large GPU clusters which are expensive.

4.3 Input

In order to do this research, we have to have the Training data on the Llama model. And also, for testing purposes have to have MNIST dataset and CIFAR10 dataset

4.4 Output.

Our final output is expected to build a CPU Distributed System that can run DL models like LLMs efficiently passing the bottlenecks of communication and much more faster than GPUs.

4.5 Process

Process of the CPU Distributed System is based on several steps.

First using Multi Worker Mirrored Strategy of Tensorflow with Keras API, we plan to create a CPU distributed System using several machines either in cloud using Microsoft Azure VMs/Amazon AWS VMs or in machines in a Lab. We plan to create a system of up to 08 machines.

We will try to run the Large Language model in that system. As Large Language models are huge models we plan to use Quantization techniques like Gradient Quantization and other Quantization techniques and through that we plan to solve the bottleneck of communication and increase the efficiency of the system on one side and on the other make it able to run the model in a single machine.

Next, we intend to further fine-tune the Large Language Model and attempt to boost the system's efficiency using the LoRA fine-tuning method.

Finally, a comparative analysis on the performance on the model with a GPU based on concerning accuracy and speed as well as cost of building such a system was carried out.

4.6 Features

Major features of the CPU Distributed System are to efficiently run a DL model in the system. So, it should have good accuracy and speed just like GPUs run in DDL.

4.7 Users

Users of the CPU distributed system are any student or Researcher plan to run a DL model

4.8 Summary

An approach to designing a CPU Distributed system was discussed within this chapter We have defined input, output, process, features, and users. More importantly, we have identified Three major tasks, namely, create a CPU Distributed System, Run the Llama model and do a comparative analysis when running with GPUs. The next chapter describes the design of this approach with reference to process identified here.

CHAPTER 5

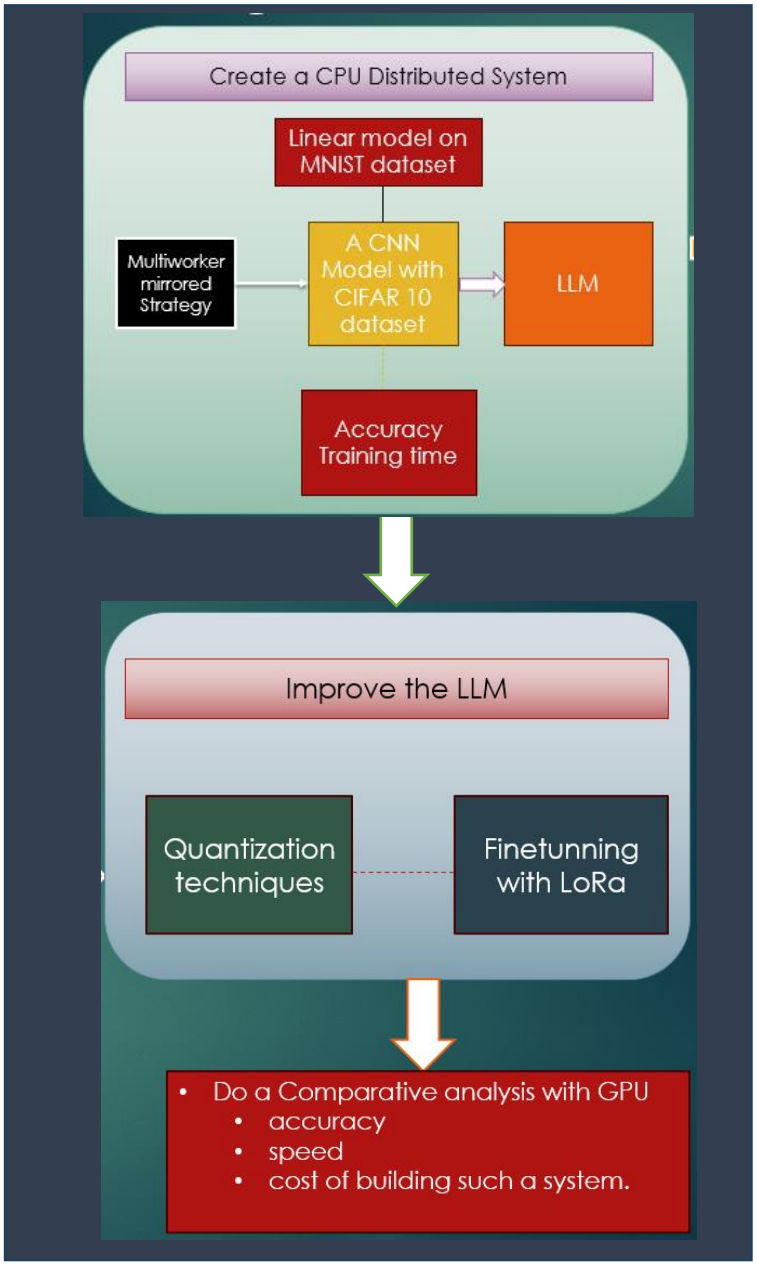
DESIGN OF THE CPU DISTRIBUTED SYSTEM

5.1 Introduction

In the preceding chapter, we presented An approach to implement a CPU distributed system, which involves Creation of a CPU Distributed System that can efficiently run a Llama model. In this chapter, we will provide a more in-depth discussion of the design of our solution. We will focus how to create a CPU Distributed System, Ho to Improve the LLM and How to do the Comparative analysis.

5.2 Distributed CPU system's Architecture

Figure 5.1: Architecture of the CPU Distributed System



5.3 Create a CPU Distributed System

In this step we planned to create a CPU distributed System with MultiworkerMirroredStrategy with Keras API. However here we have to see whether it will work correctly. So, in order to do that and to check whether this works correctly we first run A linear model using MNIST dataset. Then we checked their accuracies and the time taken to train it.

Then we went for a big model. we ran a Convolutional Neural Network Model using CIFAR10 Dataset. Here also we checked the accuracies and the time taken. And here we did a comparative analysis with the result getting from a GPU.

5.4 LLM improving

Then we tried to run a Large Language Model in our CPU distributed system. So here we took the Llama 7b model to run. However it should be noted that large models cannot be run here as the memory of a machine is only 8GB. So, it cannot keep a model that large. So, Quantization was used to run the Llama Model. Then we got the values time taken to get results, accuracies of these models and etc.

Then we finetuned the model using Lora finetuning and again got the values for time taken to generate the results and the accuracy of those results.

5.5 Comparative analysis

After getting the values of the Llama model from the CPU distributed system. We ran the same model on GPU of Google Colab and then the results were compared and did a comparative analysis.

5.8 Summary

The focus of this Chapter was on the designing of a Distributed CPU system, including the creation of the CPU Distributed System, LLM improving and Finally do a comparative analysis on the results got. Next chapter will discuss how we implement our designed solution.

CHAPTER 6

THE IMPLEMENTATION OF THE CPU DISTRIBUTED SYSTEM

6.1 Introduction

In this chapter, we delve into the implementation details of our model design presented in the previous chapter. The chapter is divided into three sections, starting with Creation of CPU Distributed System, LLM improving and Comparative analysis

. All the implementations in this study were conducted in a Python environment using various libraries.

6.2 Creation of CPU Distributed System

In here we created a cluster of IP addresses and Port Numbers and connected through MultiworkerMirroredStrategy. In here we calculated the time taken to train when fitting the model. We were able to connect up to 08 Machines together in the lab.

6.2.1 MNIST dataset

When talking about the MNIST dataset the dataset was loaded from the Keras module. And then it was split into train and test data. X values were divided by 255 to normalize as the pixel value lie in the range 0 - 255 representing the RGB (Red Green Blue) value. Then it was passed through the model and accuracies from test data were evaluated. The time taken was also calculated.

6.2.2 CIFAR10 dataset

The CIFAR10 dataset was also loaded similar to the MNIST dataset. They were also loaded from Keras module and splited to train and test data. The X values were divided by 255 to normalize and then the target variables were transformed to one-hot encoding. Then the CNN model was fitted and ran. The per worker batch size was changed to 32, 64, 128, 256 and the results were taken (accuracy and time taken). This was done by connecting together on 1, 2, 3, 4, 6, and 8 number of CPU machines without any GPUs.

6.3 LLM improving

6.3.1 Llama model Quantization

It was found impossible to quantize the model straight from Llama 2 7b model which is around 13.5GB due Memory constraints as the CPUs in the labs contain only 8GB memories. When talking about Llama 7b Quantized model. They are already there ready to download in huggingface. So llama-2-7b.ggmlv3.q4_0 was selected for the model. Which is the 04 bit Quantization model of Llama 2 7b. These were directly loaded to each and every individual machine.

6.3.2 Llama model evaluation

Unlike image processing data we cannot find the accuracies of LLMs. So two use cases of LLMs were selected for this scenario. Summarization and Translation. At first Wonder_city_Information dataset was selected for both cases. However it seems too large taking more than 04 hours to give all outputs. So, an essay about Elephant was selected for both cases.

In this scenario Summarization was measured by Rouge Score and Bert Score and the total amount of time required to produce the outcome. And Translation was measured by Bleu Score and Bert Score and the total amount of time required to produce the outcome. Translation was done from English to German language. For Generalization the results from the first three machines were selected for the analysis, and from these results the ones giving out the least amount of time taken to from an output was taken out of the three machines for the final comparative analysis.

And also, in order to discover the behavior of these results for these were rerun for three scenarios.

1. For the entire essay: To measure results for a large dataset
2. For some sentences from the essay under maximum number of tokens Llama model can give outputs
3. For part of an essay more than maximum number of tokens.

Finetunning was also tried but a memory error occurred during execution.

6.4 Comparative analysis

The base code for CIFAR10 dataset evaluation was run in Google Colab T4 GPU. And the results got were compared with the results taken from Distributed System.

Regarding the Llama model the best results from the time taken was compared with values when running the base code in Google Colab T4 GPU. Final the behavior of the results was also observed in the above mentioned three scenarios.

6.5 Summary

We have provided a thorough description of our study's implementation in this chapter. We talked in particular about the pre-processing measures we used to clean and prepare our datasets. Next, we talked about loading the Llama model and assessing its output. We concluded by outlining the process of the Comparative analysis. Subsequently, the upcoming chapter will provide a comprehensive analysis of the outcomes we attained and the techniques we employed to assess them.

CHAPTER 7

THE EVALUATION OF THE IMPLEMENTED CPU DISTRIBUTED SYSTEM

7.1 Introduction

In the prior chapter, we spoke about the execution of our study. We report the findings of our research in this chapter and how we have evaluated the generated models. This chapter is divided into two parts, MNIST dataset Results and CIFAR10 dataset results.

7.2 MNIST Dataset Results

Figure 7.1: the training time taken for 01 Machine

```
70/70 [==============================] - 2s 30ms/step - loss: 0.6530 - accuracy: 0.8623
Epoch 17/20
70/70 [==============================] - 2s 30ms/step - loss: 0.6326 - accuracy: 0.8536
Epoch 18/20
70/70 [==============================] - 2s 30ms/step - loss: 0.6068 - accuracy: 0.8587
Epoch 19/20
70/70 [==============================] - 2s 31ms/step - loss: 0.5667 - accuracy: 0.8596
Epoch 20/20
70/70 [==============================] - 2s 32ms/step - loss: 0.5402 - accuracy: 0.8647
>> Finished. Time elapsed: 00 : 00 : 44.
```

Figure 7.2: the training time taken for 02 Machine

```
70/70 [==============================] - 214s 3s/step - loss: 0.6806 - accuracy: 0.8467
Epoch 17/20
70/70 [==============================] - 214s 3s/step - loss: 0.6269 - accuracy: 0.8585
Epoch 18/20
70/70 [==============================] - 216s 3s/step - loss: 0.6106 - accuracy: 0.8554
Epoch 19/20
70/70 [==============================] - 217s 3s/step - loss: 0.5939 - accuracy: 0.8538
Epoch 20/20
70/70 [==============================] - 224s 3s/step - loss: 0.5545 - accuracy: 0.8656
>> Finished. Time elapsed: 01 : 11 : 43.
```

When talking about MNIST dataset results we ran the model for up to 03 machines. And for 01 machine and two machines we got an accuracy of 87.67%. However the time taken for 01 machine is 44 seconds and time taken for 02 machines is 01 hour 11 minutes and 43 seconds. We can observe that when running with 02 machines it has taken more than 100 times the time taken when running with a single machine.

And when running with three machines the training stopped in the 03rd epoch due to a communication error.

Figure 7.3: the training stopped in the 03rd epoch due to a communication error.

```
INFO:tensorflow:Collective all_reduce tensors: 1 all_reduces, num_devices = 1, group_size = 3, implementation = CommunicationIm
plementation.AUTO, num_packs = 1
INFO:tensorflow:Collective all_reduce tensors: 1 all_reduces, num_devices = 1, group_size = 3, implementation = CommunicationIm
plementation.AUTO, num_packs = 1
INFO:tensorflow:Collective all_reduce tensors: 1 all_reduces, num_devices = 1, group_size = 3, implementation = CommunicationIm
plementation.AUTO, num_packs = 1
INFO:tensorflow:Collective all_reduce tensors: 6 all_reduces, num_devices = 1, group_size = 3, implementation = CommunicationIm
plementation.AUTO, num_packs = 1
INFO:tensorflow:Collective all_reduce tensors: 1 all_reduces, num_devices = 1, group_size = 3, implementation = CommunicationIm
plementation.AUTO, num_packs = 1
INFO:tensorflow:Collective all_reduce tensors: 1 all_reduces, num_devices = 1, group_size = 3, implementation = CommunicationIm
plementation.AUTO, num_packs = 1
INFO:tensorflow:Collective all_reduce tensors: 1 all_reduces, num_devices = 1, group_size = 3, implementation = CommunicationIm
plementation.AUTO, num_packs = 1
INFO:tensorflow:Collective all_reduce tensors: 1 all_reduces, num_devices = 1, group_size = 3, implementation = CommunicationIm
plementation.AUTO, num_packs = 1
INFO:tensorflow:Collective all_reduce tensors: 1 all_reduces, num_devices = 1, group_size = 3, implementation = CommunicationIm
plementation.AUTO, num_packs = 1
INFO:tensorflow:Collective all_reduce tensors: 1 all_reduces, num_devices = 1, group_size = 3, implementation = CommunicationIm
plementation.AUTO, num_packs = 1
70/70 [==============================] - 370s 5s/step - loss: 2.2834 - accuracy: 0.1433
Epoch 2/20
70/70 [==============================] - 353s 5s/step - loss: 2.2251 - accuracy: 0.2565
Epoch 3/20
57/70 [=======================>......] - ETA: 1:06 - loss: 6.4883 - accuracy: 1.1719
```

.

7.3 CIFAR10 Results

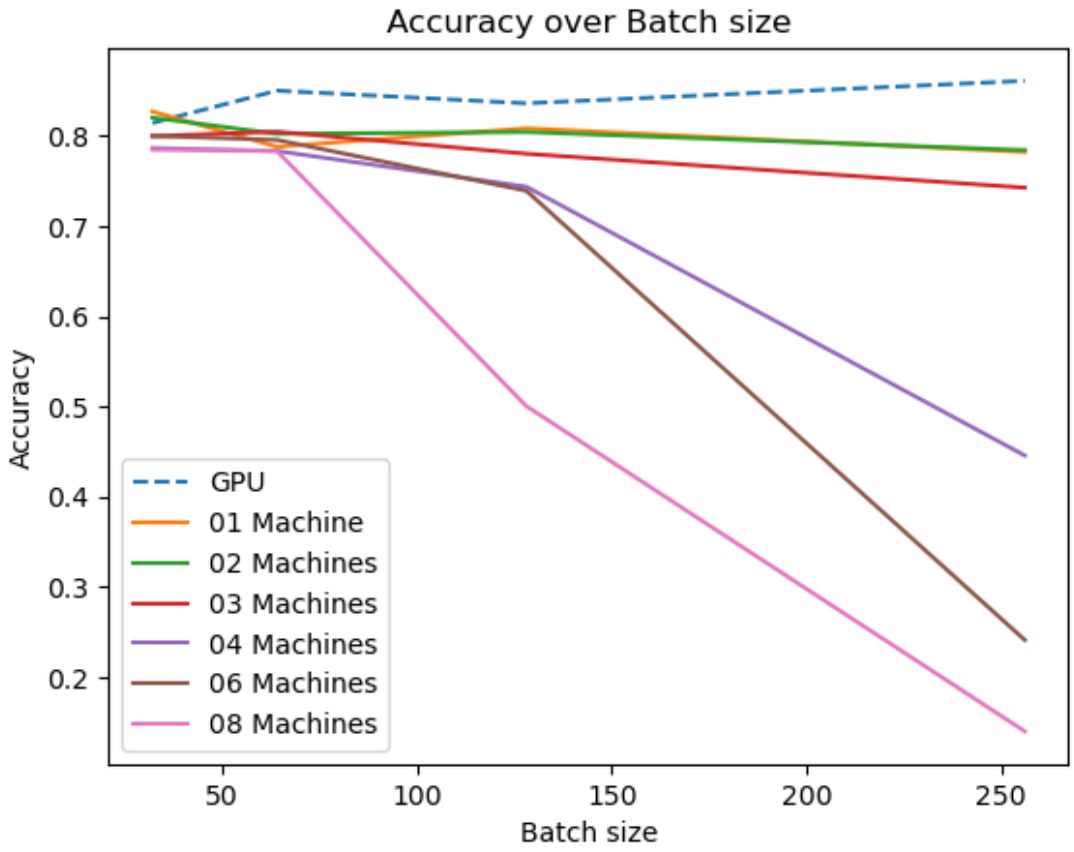
7.3.1 Accuracies

Table 7.1: Accuracies results

No_of_workers_per_batch	32	64	128	256
01_Machine	0.8269	0.7878	0.8083	0.7821
02_Machines	0.8201	0.8022	0.8042	0.7841
03_Machines	0.7996	0.8045	0.7801	0.7425
04_Machines	0.7863	0.7831	0.7436	0.4456
06_Machines	0.7999	0.7954	0.7391	0.2411
08_Machines	0.7846	0.7830	0.5004	0.1399
GPU	0.8142	0.8500	0.8360	0.8609

Figure 7.4: Accuracies over Batch Size

. F



From looking at these results we can observe that When the Batch size increases the accuracy also decreases. But this is not the case in 01 machine and two machines as the accuracy has been decreased when batch size is 64 and increased again when batch size is 128.

We can observe that, When the number of machines are also increasing the accuracy also decreases. We can get a somewhat good accuracy up to using 4 and 6 machines up to a batch size of 128 but after that the accuracy decreases exponentially.

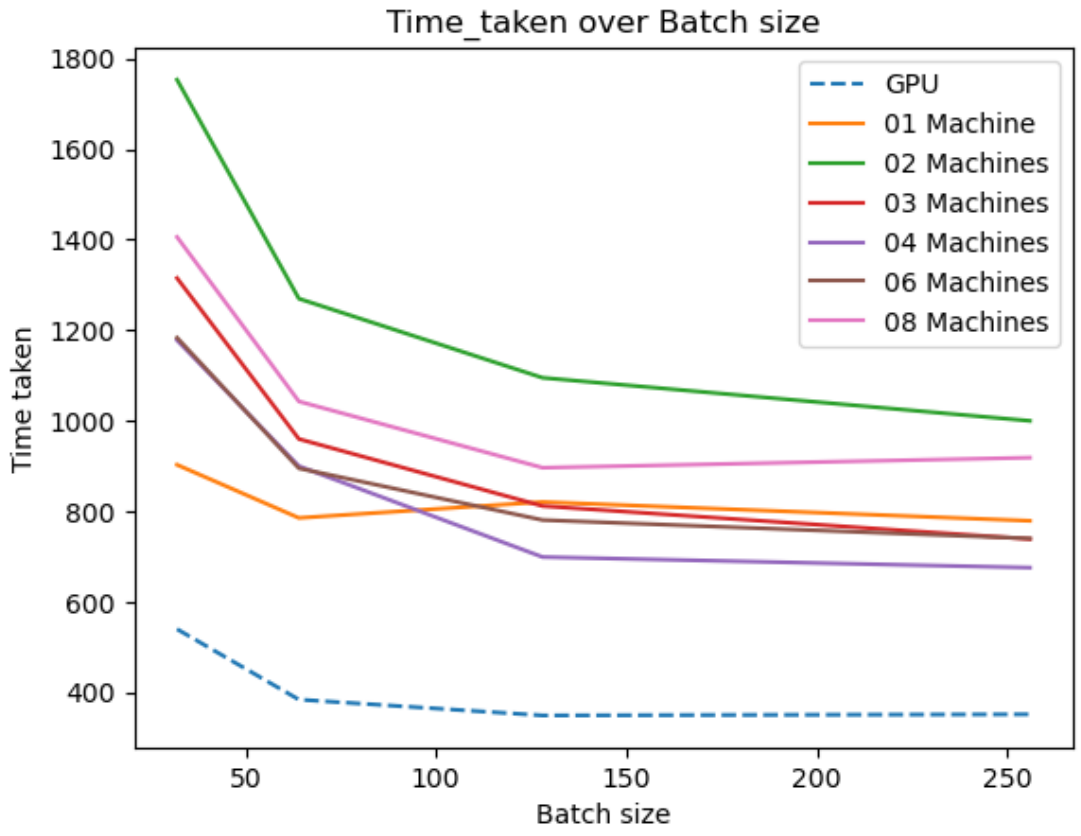
Compared with Google Colab T4 GPU only the batch size of 32 for 01 machine and 02 machines surpassed the GPU's accuracy.

7.3.2 Time Taken

Table 7.2: Time taken results

No_of_workers_per_batch	32	64	128	256
01_Machine	902.517579	785.108801	820.283189	778.751383
02_Machines	1753.415086	1269.632735	1094.825655	999.654525
03_Machines	1315.030171	959.511662	811.394799	738.401128
04_Machines	1177.700255	899.549414	698.685122	675.225406
06_Machines	1183.308959	894.698699	780.424687	739.749271
08_Machines	1405.864068	1042.519577	896.051503	918.240589
GPU	539.337196	383.623174	348.643773	351.058871

Figure 7.5: Time Taken over Batch Size



When looking at the results we can observe that,

- In using 2 machines much more time is taken than the time taken to run on 1 machine

- However, with the increase of Number of machines time taken has decreases surpassing even time taken by 1 machine
- After the number of machines are 4 the time taken increases again for higher number of machines
- When the batch size increases the time taken decreases
- But for higher number of machines like 6 and 8 the time taken decreases up to a certain batch size(128) and then increases again
- However, none of the models were fast enough to pass the time taken for the GPU
- When comparing both results, 04 Machines with 128 batch size give the best output.

7.4 Llama Model Results for Summarization

7.4.1 Time taken for the three machines for summarization

Table 7.3: Summarization Results for the entire essay

Summarization Results for the entire essay

Number of Machines	Machine 1				Machine 2				Machine 3			
	Time Taken	BERTScore			Time Taken	BERTScore			Time Taken	BERTScore		
		Precision	Recall	F1		Precision	Recall	F1		Precision	Recall	F1
1	0:24:04	0.4126	0.382	0.3967								
2	0:21:21	0.3949	0.5124	0.4461	0:19:22	0.5209	0.6058	0.5602				
3	0:20:12	0.5096	0.594	0.5486	0:18:52	0.5738	0.639	0.6047	0:19:47	0.5696	0.63	0.5983
4	0:17:36	0.6124	0.6489	0.6301	0:17:48	0.5223	0.6082	0.562	0:20:23	0.5436	0.6173	0.5781
5	0:19:43	0.3949	0.5124	0.4461	0:19:19	0.5245	0.6077	0.563	0:19:45	0.5447	0.6166	0.5785
6	0:20:30	0.5223	0.6082	0.562	0:19:31	0.5447	0.6166	0.5785	0:17:23	0.5733	0.6392	0.6044
7	0:20:10	0.5738	0.639	0.6047	0:21:28	0.3949	0.5124	0.4461	0:21:33	0.3949	0.5124	0.4461
8	0:18:43	0.5472	0.6172	0.5801	0:17:50	0.544	0.6163	0.5779	0:19:45	0.5472	0.6172	0.5801
T4 - GPU	0:02:59	0.5434	0.6046	0.5724								

Table 7.4: Summarization Results for text under maximum tokens of Llama 2

Summarization Results for text under maximum tokens of Llama 2

Number of Machines	Machine 1				Machine 2				Machine 3			
	Time Taken	BERTScore			Time Taken	BERTScore			Time Taken	BERTScore		
		Precision	Recall	F1		Precision	Recall	F1		Precision	Recall	F1
1	0:07:14	0.6499	0.7783	0.7083								
2	0:07:38	0.6499	0.7783	0.7083	0:10:24	0.6499	0.7783	0.7083				
3	0:05:56	0.6499	0.7783	0.7083	0:08:41	0.6499	0.7783	0.7083	0:08:15	0.6499	0.7783	0.7083
4	0:10:15	0.6499	0.7783	0.7083	0:08:27	0.6499	0.7783	0.7083	0:05:07	0.6499	0.7783	0.7083
5	0:05:43	0.6499	0.7783	0.7083	0:09:00	0.6499	0.7783	0.7083	0:05:45	0.6499	0.7783	0.7083
6	0:04:41	0.6499	0.7783	0.7083	0:08:20	0.6499	0.7783	0.7083	0:05:35	0.6499	0.7783	0.7083
7	0:06:16	0.6499	0.7783	0.7083	0:08:12	0.6499	0.7783	0.7083	0:06:20	0.6499	0.7783	0.7083
8	0:04:42	0.6499	0.7783	0.7083	0:08:26	0.6499	0.7783	0.7083	0:04:58	0.6499	0.7783	0.7083
T4 - GPU	0:00:27	0.9172	0.9684	0.9421								

Table 7.5: Summarization Results for part of the Essay

Summarization Results for part of the Essay												
Number of Machines	Machine 1				Machine 2				Machine 3			
	Time Taken	BERTScore			Time Taken	BERTScore			Time Taken	BERTScore		
		Precision	Recall	F1		Precision	Recall	F1		Precision	Recall	F1
1	0:09:00	0.5309	0.6089	0.5673								
2	0:07:07	0.5309	0.6089	0.5673	0:05:32	0.5082	0.6171	0.5574				
3	0:05:45	0.517	0.593	0.5524	0:05:37	0.5082	0.6171	0.5574	0:06:00	0.5082	0.6171	0.5574
4	0:06:56	0.5469	0.6072	0.5755	0:05:35	0.5082	0.6171	0.5574	0:05:43	0.5034	0.6101	0.5516
5	0:05:45	0.5034	0.6101	0.5516	0:05:04	0.5082	0.6171	0.5574	0:06:09	0.5082	0.6171	0.5574
6	0:05:53	0.5055	0.6096	0.5527	0:05:35	0.5309	0.6089	0.5673	0:06:05	0.5082	0.6171	0.5574
7	0:05:42	0.5082	0.6171	0.5574	0:05:29	0.5163	0.593	0.552	0:05:48	0.5163	0.593	0.552
8	0:05:42	0.5082	0.6171	0.5574	0:05:38	0.5055	0.6096	0.5527	0:05:44	0.5469	0.6072	0.5755
T4 - GPU	0:00:52	0.5483	0.6034	0.5746								

7.4.2 Summarization results selected from the best time taken out of the three machines

Table 7.6: Selected Summarization Results for the entire essay

Summarization Results for the entire essay														
Number of Machines	Time Taken	BERTScore			Rouge Score									
					rouge-1			rouge-2			rouge-l			
		Precision	Recall	F1	r	p	f	r	p	f	r	p	f
1	0:24:04	0.4126	0.382	0.3967	0.0526	0.5789	0.0965	0.0115	0.1429	0.0213	0.0478	0.5263	0.0877
2	0:19:22	0.5209	0.6058	0.5602	0.9761	0.5231	0.6811	0.8908	0.4122	0.5636	0.9761	0.5231	0.6811
3	0:18:52	0.5738	0.639	0.6047	0.9761	0.5862	0.7325	0.8908	0.4559	0.6031	0.9761	0.5862	0.7325
4	0:17:36	0.6124	0.6489	0.6301	0.9952	0.5926	0.7429	0.9454	0.491	0.6464	0.9952	0.5926	0.7429
5	0:19:19	0.5245	0.6077	0.563	0.9761	0.4939	0.6559	0.8908	0.3861	0.5387	0.9761	0.4939	0.6559
6	0:17:23	0.5733	0.6392	0.6044	0.9952	0.619	0.7633	0.9454	0.5101	0.6626	0.9952	0.619	0.7633
7	0:20:10	0.5738	0.639	0.6047	0.9761	0.5913	0.7365	0.8908	0.4552	0.6025	0.9761	0.5913	0.7365
8	0:17:50	0.544	0.6163	0.5779	0.9952	0.5561	0.7136	0.9397	0.4586	0.6164	0.9952	0.5561	0.7136
T4 - GPU	0:02:59	0.5434	0.6046	0.5724	1	0.5471	0.7073	0.9483	0.4558	0.6157	1	0.5471	0.7073

Table 7.7: Selected Summarization Results for text under maximum tokens of Llama 2

Summarization Results for text under maximum tokens of Llama 2														
Number of Machines	Time Taken	BERTScore			Rouge Score									
					rouge-1			rouge-2			rouge-l			
		Precision	Recall	F1	r	p	f	r	p	f	r	p	f
1	0:07:14	0.6499	0.7783	0.7083	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
2	0:07:38	0.6499	0.7783	0.7083	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
3	0:05:56	0.6499	0.7783	0.7083	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
4	0:05:07	0.6499	0.7783	0.7083	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
5	0:05:43	0.6499	0.7783	0.7083	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
6	0:04:41	0.6499	0.7783	0.7083	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
7	0:06:16	0.6499	0.7783	0.7083	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
8	0:04:42	0.6499	0.7783	0.7083	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
T4 - GPU	0:00:27	0.9172	0.9684	0.9421	1	0.9213	0.959	0.9649	0.9066	0.9348	1	0.9213	0.959

Table 7.8: Selected Summarization Results for part of the Essay

Summarization Results for part of the Essay														
Number of Machines	Time Taken	BERTScore			Rouge Score									
					rouge-1			rouge-2			rouge-l			
		Precision	Recall	F1	r	p	f	r	p	f	r	p	f
1	0:09:00	0.5309	0.6089	0.5673	0.3162	0.3814	0.3458	0.0351	0.0411	0.0379	0.2991	0.3608	0.3271
2	0:05:32	0.5082	0.6171	0.5574	0.3504	0.3905	0.3694	0.0292	0.0323	0.0307	0.3333	0.3714	0.3514
3	0:05:37	0.5082	0.6171	0.5574	0.3504	0.3905	0.3694	0.0292	0.0323	0.0307	0.3333	0.3714	0.3514
4	0:05:35	0.5082	0.6171	0.5574	0.3504	0.3905	0.3694	0.0292	0.0323	0.0307	0.3333	0.3714	0.3514
5	0:05:04	0.5082	0.6171	0.5574	0.3504	0.3905	0.3694	0.0292	0.0323	0.0307	0.3333	0.3714	0.3514
6	0:05:35	0.5309	0.6089	0.5673	0.3162	0.3814	0.3458	0.0351	0.0411	0.0379	0.2991	0.3608	0.3271
7	0:05:29	0.5163	0.593	0.552	0.2991	0.3608	0.3271	0.0175	0.0208	0.019	0.2906	0.3505	0.3178
8	0:05:38	0.5055	0.6096	0.5527	0.3248	0.3689	0.3455	0.0234	0.0274	0.0252	0.2991	0.3398	0.3182
T4 - GPU	0:00:52	0.5483	0.6034	0.5746	0.3419	0.3333	0.3376	0.0234	0.0238	0.0236	0.3162	0.3083	0.3122

7.4.3 Time taken on the Summarization results

Table 7.9: Time taken on the selected Summarization results

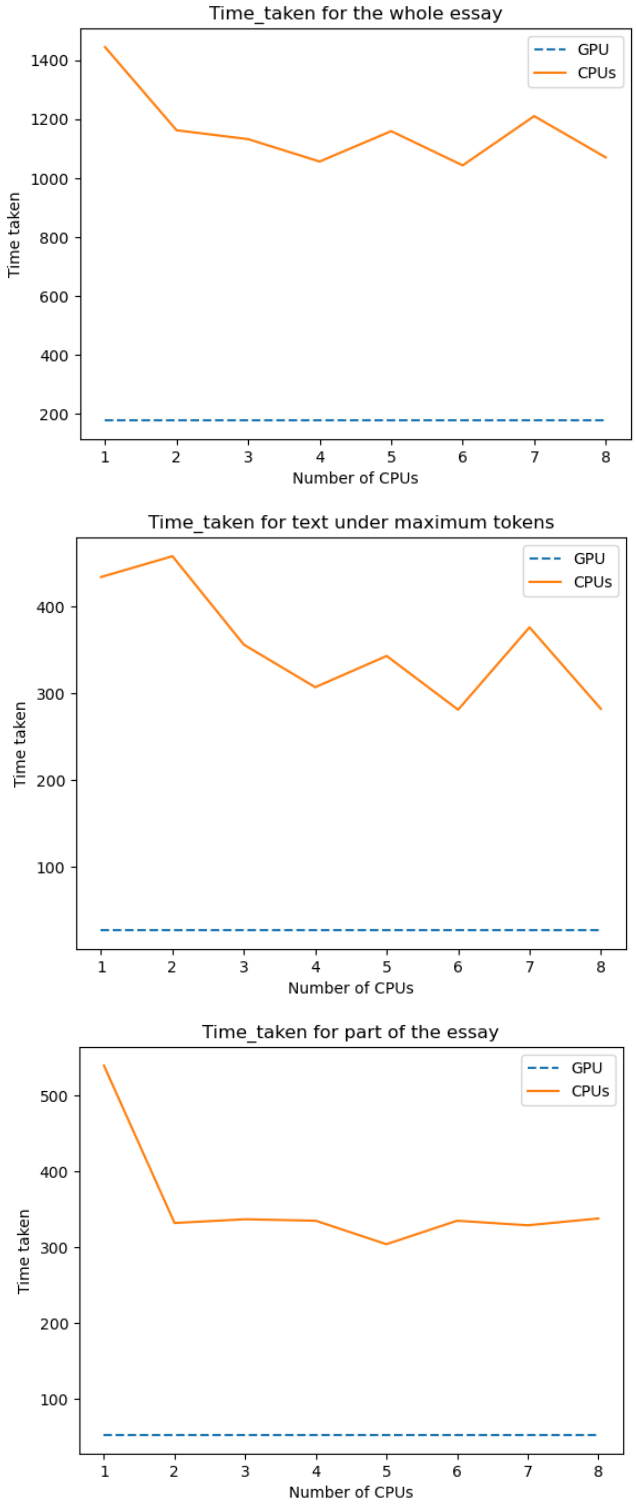
Number of Machines	for the entire essay	for text under maximum tokens of Llama 2	for part of the Essay
1	0:24:04	0:07:14	0:09:00
2	0:19:22	0:07:38	0:05:32
3	0:18:52	0:05:56	0:05:37
4	0:17:36	0:05:07	0:05:35
5	0:19:19	0:05:43	0:05:04
6	0:17:23	0:04:41	0:05:35
7	0:20:10	0:06:16	0:05:29
8	0:17:50	0:04:42	0:05:38
T4 - GPU	0:02:59	0:00:27	0:00:52

When observing the time taken on the summarization results as sown in Table 7.9, the lowest time taken for the entire essay is 17 minutes and 23 seconds while running on 6 machines and it was 2 minutes and 59 seconds when run on T4 GPU of Google Collab. This lowest time by CPU is near 6 times the time taken by the GPU.

When observing results for text under maximum tokens of Llama 2 the lowest time taken is 4 minutes and 41 seconds while running on 6 machines and it was 27 seconds in T4 GPU results. This lowest time by CPU is roughly 11 times the time taken by the GPU.

When observing results for part of the essay the lowest time taken is 5 minutes and 4 seconds while running on 5 machines, and it was 52 seconds when run on GPU. This lowest time by CPU is near 6 times the time taken by the GPU.

Figure 7.6: Time taken on the selected Summarization results



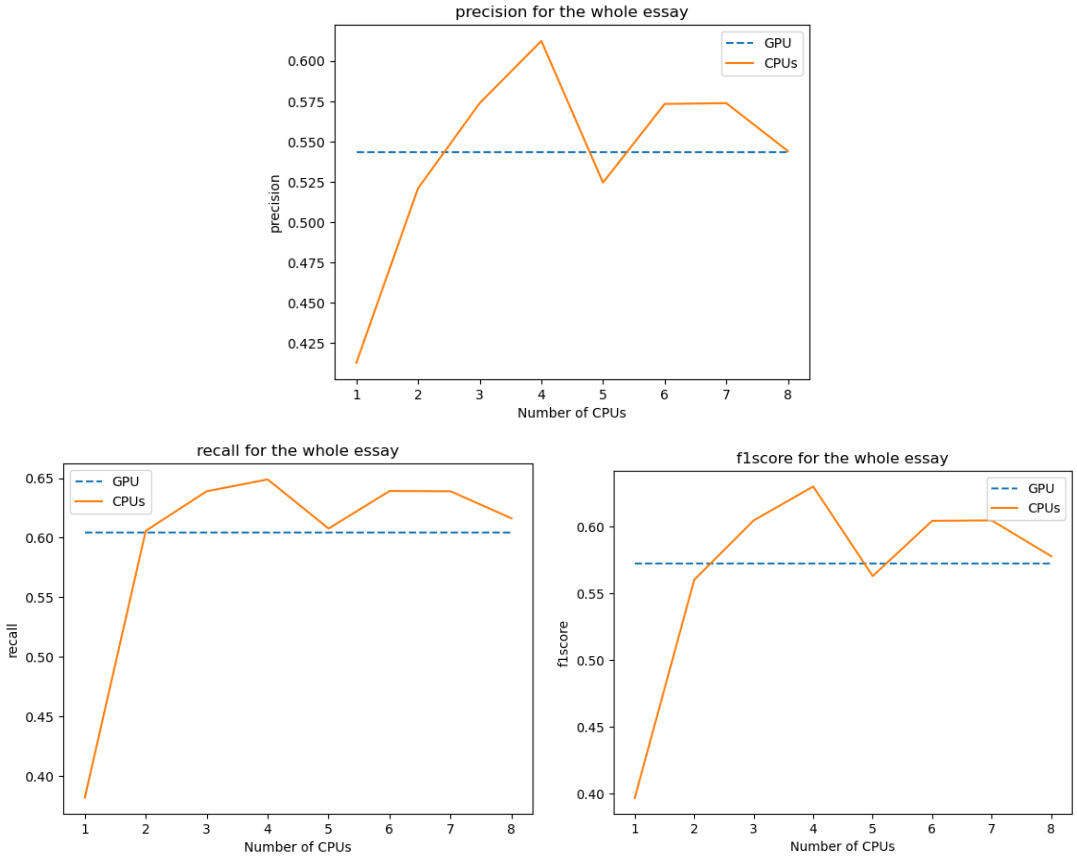
When observing the behavior as shown in Figure 7.6, it was somewhat similar for all three cases and the lowest time taken is in the region of 5 and 6 machines.

7.4.4 BERTscores of the Summarization results

Table 7.10: BERTscores of the selected Summarization results

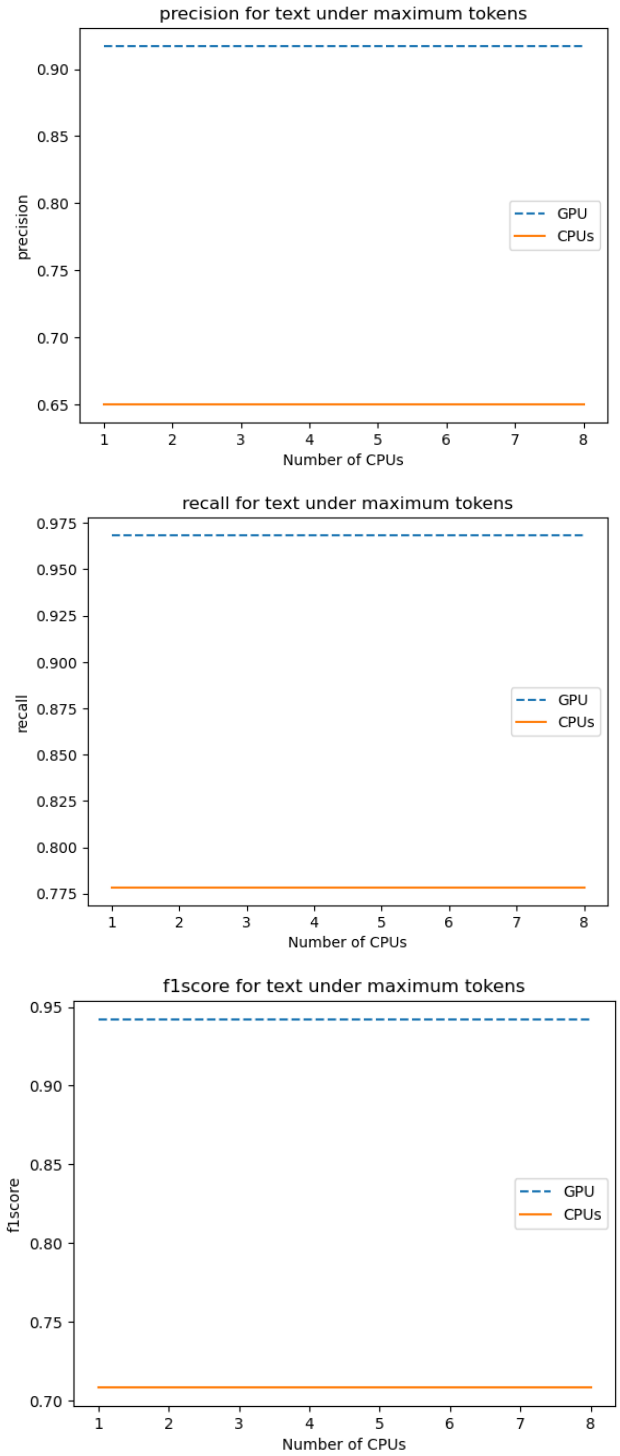
Number of Machines	for the entire essay			for text under maximum tokens of Llama 2			for part of the Essay		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
1	0.4126	0.382	0.3967	0.6499	0.7783	0.7083	0.5309	0.6089	0.5673
2	0.5209	0.6058	0.5602	0.6499	0.7783	0.7083	0.5082	0.6171	0.5574
3	0.5738	0.639	0.6047	0.6499	0.7783	0.7083	0.5082	0.6171	0.5574
4	0.6124	0.6489	0.6301	0.6499	0.7783	0.7083	0.5082	0.6171	0.5574
5	0.5245	0.6077	0.563	0.6499	0.7783	0.7083	0.5082	0.6171	0.5574
6	0.5733	0.6392	0.6044	0.6499	0.7783	0.7083	0.5309	0.6089	0.5673
7	0.5738	0.639	0.6047	0.6499	0.7783	0.7083	0.5163	0.593	0.552
8	0.544	0.6163	0.5779	0.6499	0.7783	0.7083	0.5055	0.6096	0.5527
T4 - GPU	0.5434	0.6046	0.5724	0.9172	0.9684	0.9421	0.5483	0.6034	0.5746

Figure 7.7: BERTscores of the selected Summarization results for the entire essay



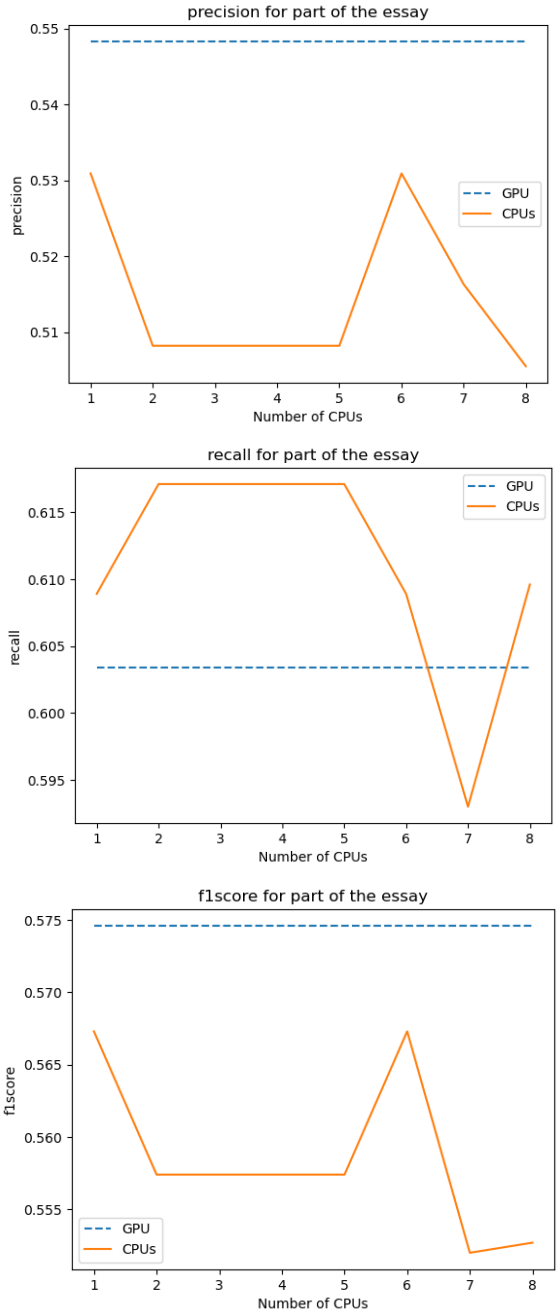
As shown in figure 7.7, the BERTscores of the selected Summarization results for the entire essay, In precision the value increase and surpassed value by GPU after 3 machines except when connecting with 5 machines where a sudden decrease was observed. And in recall all values except 1 machine of CPU has surpassed GPU. However when 5 machines are connected a sudden decrease was observed. All f1 scores apart 1,2 and 5 machines have surpassed GPU value. In 5 machines the same decrease was observed and also the same behavior was observed in all three cases.

Figure 7.8: BERTscores of the selected Summarization results for text under maximum tokens of Llama 2



As shown in Figure 7.8, Same behavior was observed in each case and same amount of precision, recall and f1 scores were obtained for each number of machines (same answer was obtained in each scenario). However, none of the scenarios could surpass values obtained by GPUs.

Figure 7.9: BERTscores of the selected Summarization results for part of the Essay



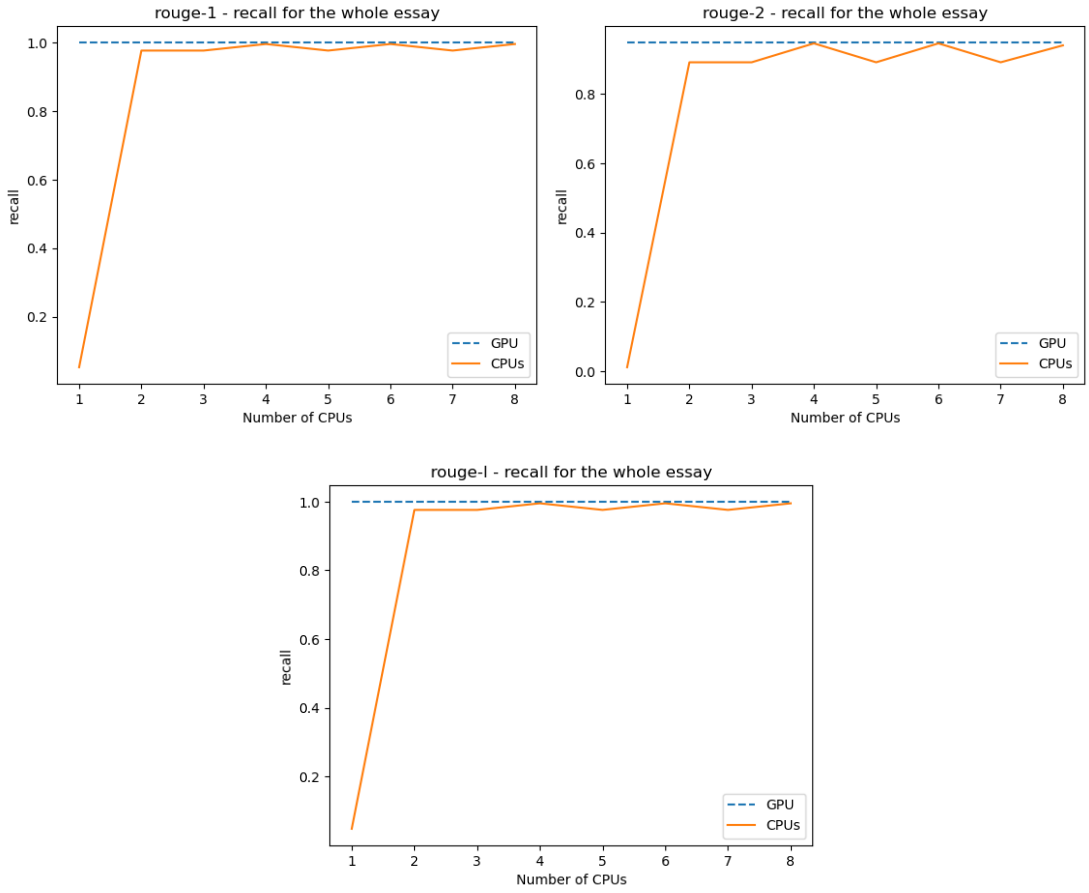
As shown in Figure 7.9, both precision and f1 score were not able to surpass results obtained by T4-GPU and same behavior was observed. However recall differs and except when number of machines are 7, all the rest of the cases surpasses GPU result.

7.4.5 Rouge Scores of the Summarization results

Table 7.11: Rouge Scores of the selected Summarization results for the entire essay

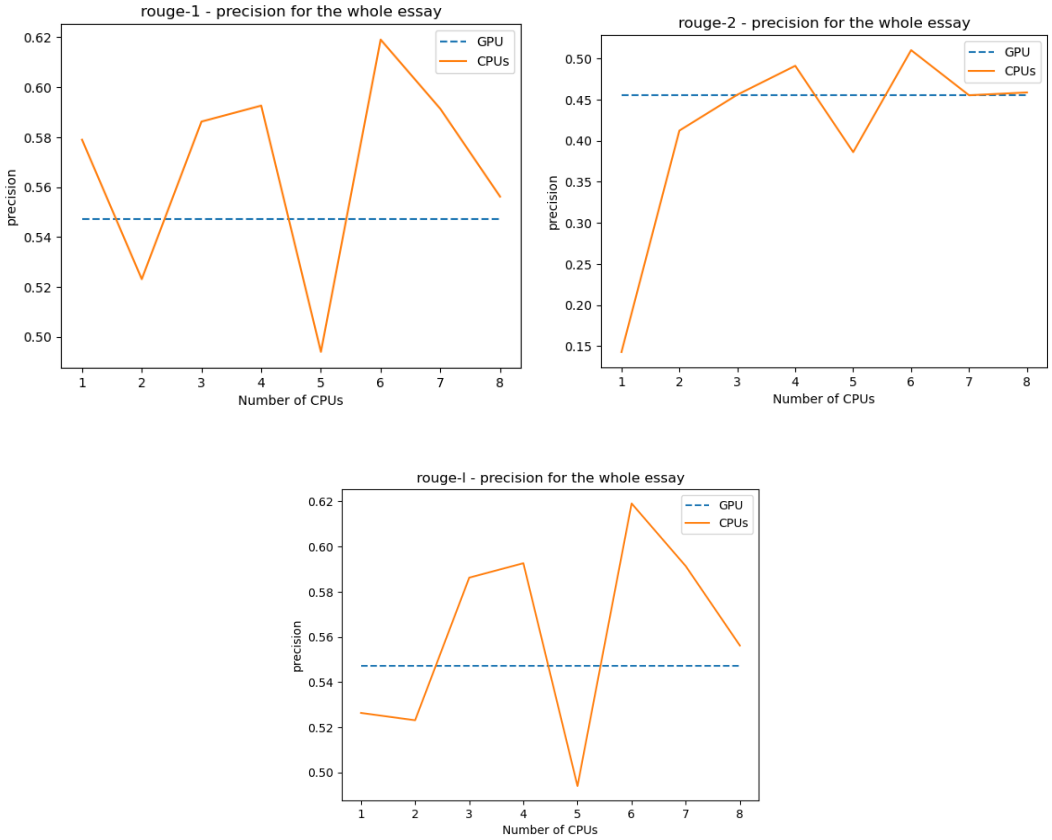
Number of Machines	rouge-1			rouge-2			rouge-l		
	r	p	f	r	p	f	r	p	f
1	0.0526	0.5789	0.0965	0.0115	0.1429	0.0213	0.0478	0.5263	0.0877
2	0.9761	0.5231	0.6811	0.8908	0.4122	0.5636	0.9761	0.5231	0.6811
3	0.9761	0.5862	0.7325	0.8908	0.4559	0.6031	0.9761	0.5862	0.7325
4	0.9952	0.5926	0.7429	0.9454	0.491	0.6464	0.9952	0.5926	0.7429
5	0.9761	0.4939	0.6559	0.8908	0.3861	0.5387	0.9761	0.4939	0.6559
6	0.9952	0.619	0.7633	0.9454	0.5101	0.6626	0.9952	0.619	0.7633
7	0.9761	0.5913	0.7365	0.8908	0.4552	0.6025	0.9761	0.5913	0.7365
8	0.9952	0.5561	0.7136	0.9397	0.4586	0.6164	0.9952	0.5561	0.7136
T4 - GPU	1	0.5471	0.7073	0.9483	0.4558	0.6157	1	0.5471	0.7073

Figure 7.10: Recalls of Rouge Scores of the selected Summarization results for the entire essay



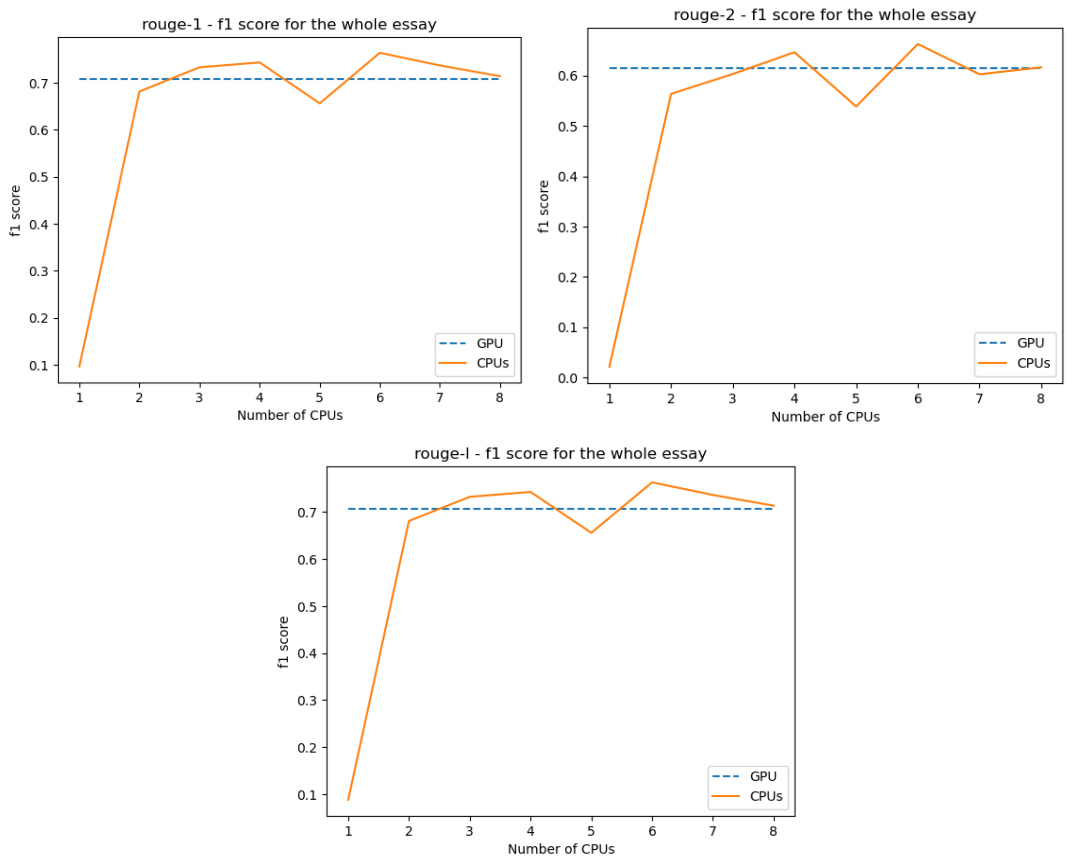
As observed in Figure 7.10, In all recalls of rouge 1, rouge 2 and rouge l a same behavior was observed in recall. Machines 4 and 6 were the only recall values in each 3 cases that surpasses and nears GPU recalls.

Figure 7.11: Precisions of Rouge Scores of the selected Summarization results for the entire essay



As observed in Figure 7.11, in Precision Somewhat same kind of behavior was observed in rouge 1 and rouge l except 1 machine. A Sudden decrease in 5 machines was observed in all three scenarios. 3,4,6,7 and 8 machines have all surpassed values obtained by GPUs in all 3 cases in precision.

Figure 7.12: F1-Scores of Rouge Scores of the selected Summarization results for the entire essay



As observed in Figure 7.13, Same behavior was observed in all 3 scenarios of rouge-1, rouge-2, rouge-l. The sudden decrease when connecting with 5 machines was also observed here. Value of 1 machine Is the lowest observed 3,4,6 connected CPUs have surpassed f1 scores of T4 GPU in all 3 cases.

Table 7.12: Rouge Scores of the selected Summarization results for text under maximum tokens of Llama 2

Number of Machines	rouge-1			rouge-2			rouge-l		
	r	p	f	r	p	f	r	p	f
1	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
2	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
3	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
4	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
5	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
6	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
7	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
8	0.6838	0.9195	0.7843	0.5965	0.887	0.7133	0.6838	0.9195	0.7843
T4 - GPU	1	0.9213	0.959	0.9649	0.9066	0.9348	1	0.9213	0.959

Figure 7.13: Recalls of Rouge Scores of the selected Summarization results for text under maximum tokens of Llama 2

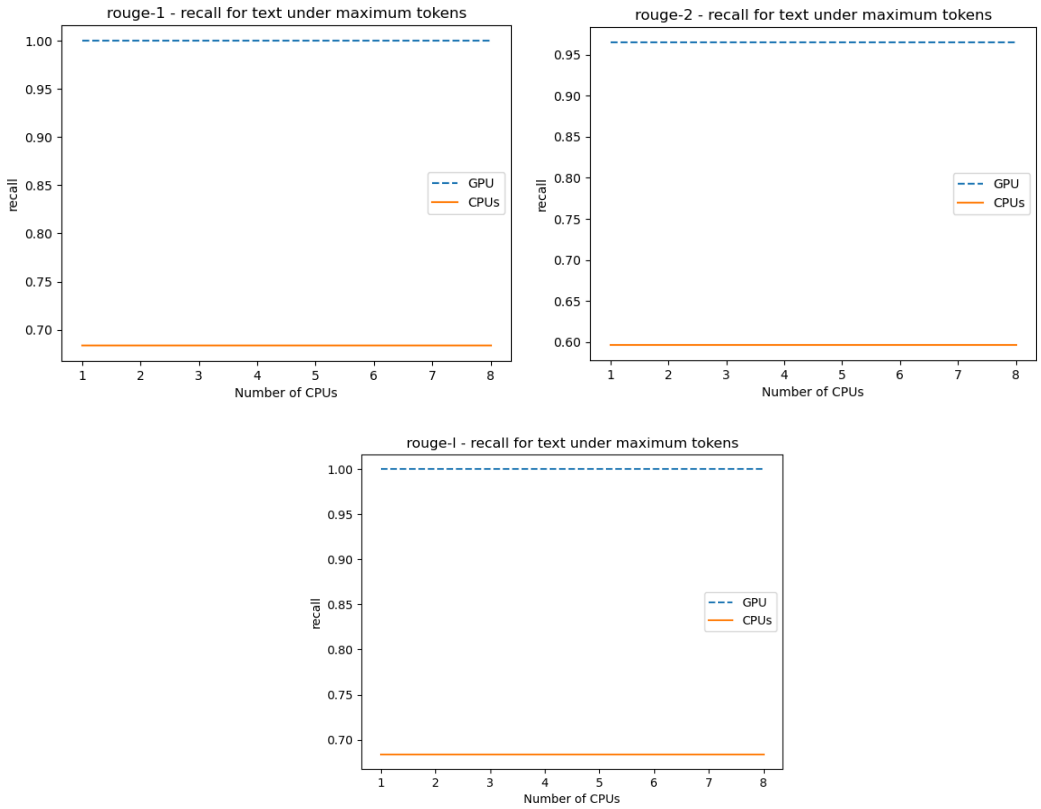


Figure 7.14: Precisions of Rouge Scores of the selected Summarization results for text under maximum tokens of Llama 2

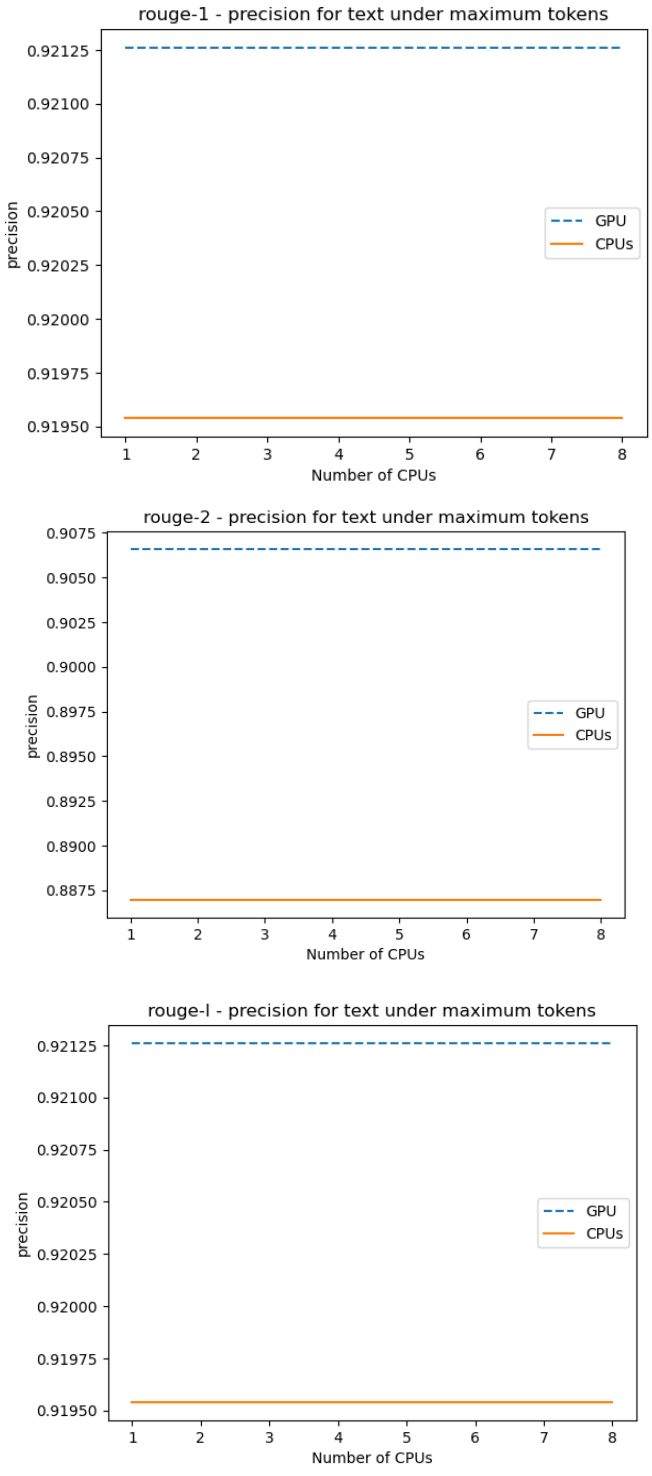
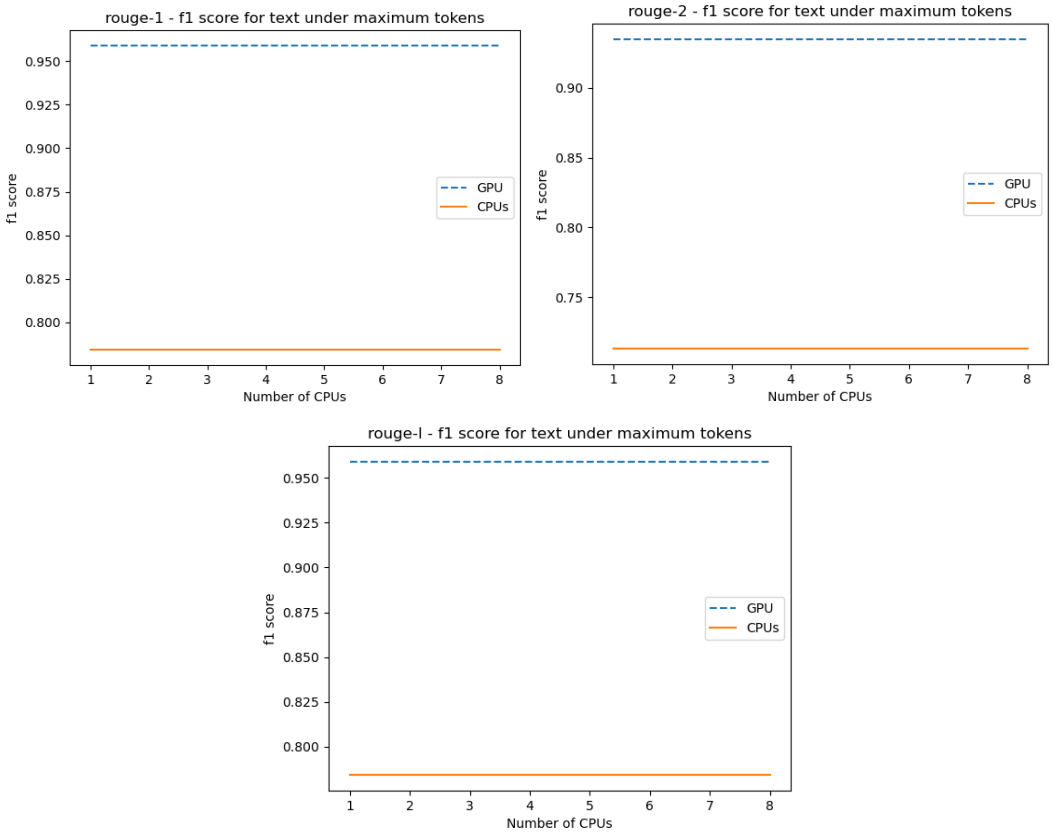


Figure 7.15: F1-Scores of Rouge Scores of the selected Summarization results for text under maximum tokens of Llama 2

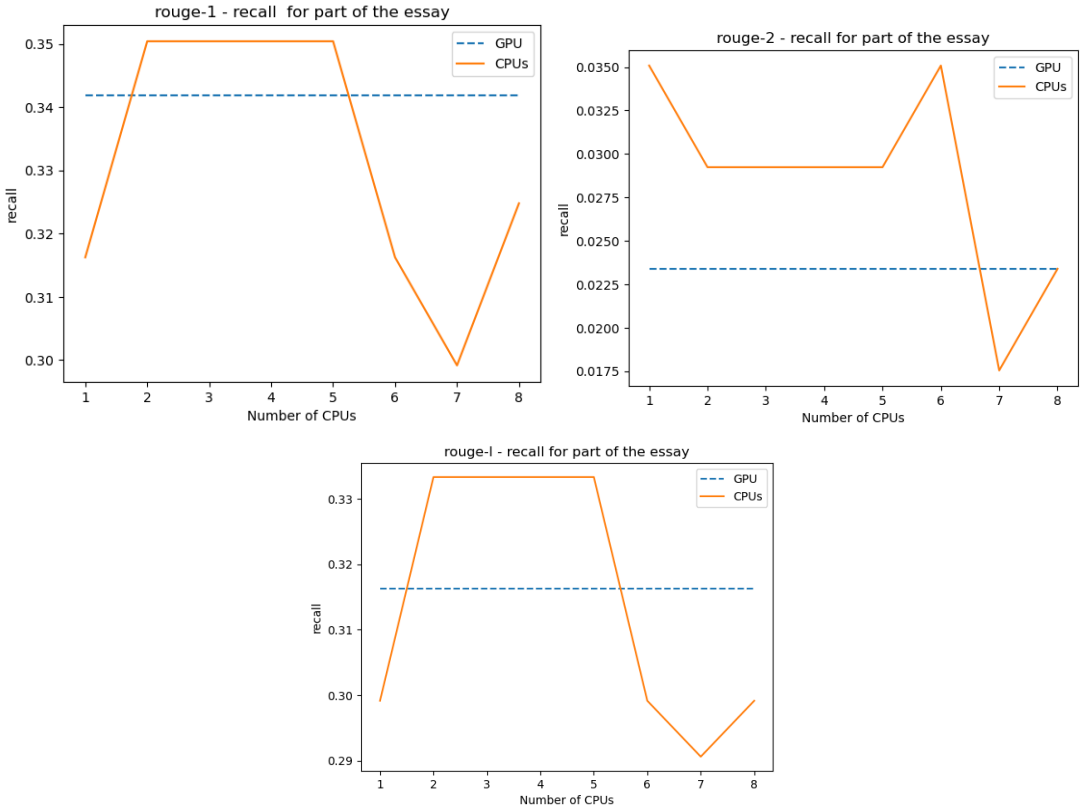


As observed in Figure 7.13, Figure 7.14 and Figure 7.15 all recalls, Precisions and F1 scores of Rouge 1, Rouge 2 and Rouge l show the same behavior. And in each case constant recalls, Precisions and F1 scores have been obtained for each number of machines (same answer was obtained in each scenario). However, none of the scenarios could surpass values obtained by GPUs.

Table 7.13: Rouge Scores of the selected Summarization results for part of the Essay

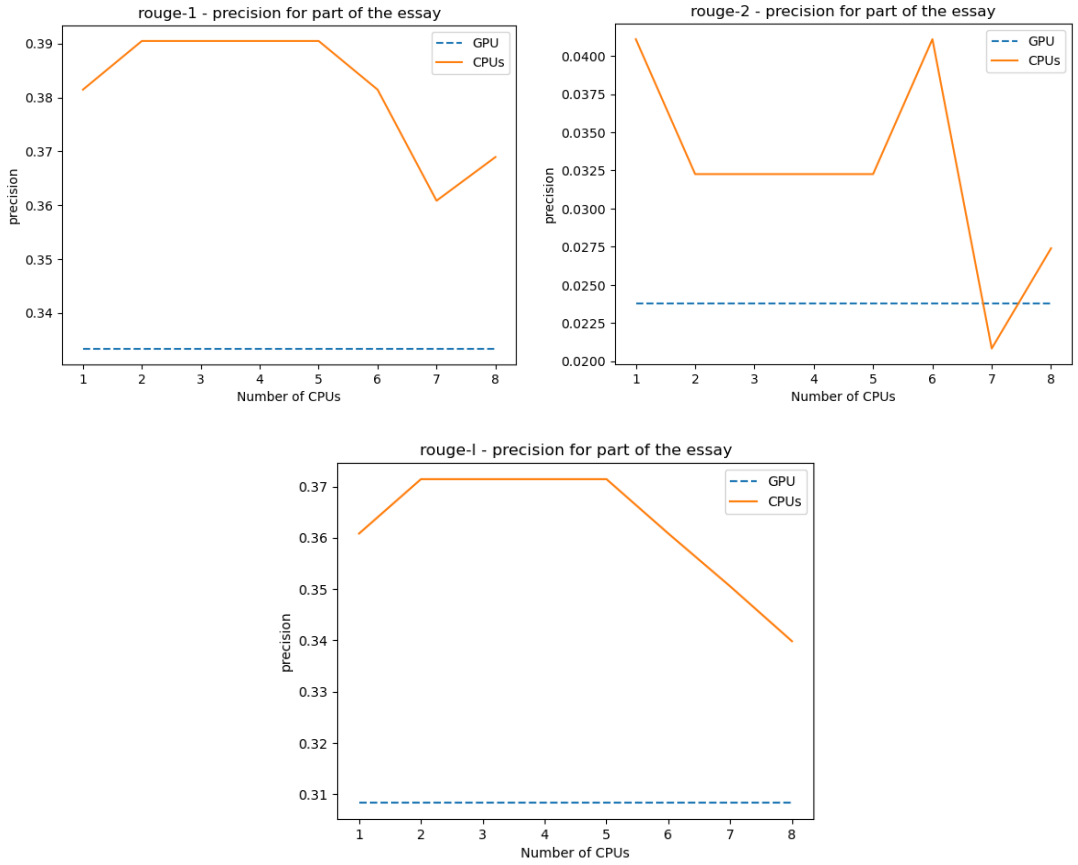
Number of Machines	rouge-1			rouge-2			rouge-l		
	r	p	f	r	p	f	r	p	f
1	0.3162	0.3814	0.3458	0.0351	0.0411	0.0379	0.2991	0.3608	0.3271
2	0.3504	0.3905	0.3694	0.0292	0.0323	0.0307	0.3333	0.3714	0.3514
3	0.3504	0.3905	0.3694	0.0292	0.0323	0.0307	0.3333	0.3714	0.3514
4	0.3504	0.3905	0.3694	0.0292	0.0323	0.0307	0.3333	0.3714	0.3514
5	0.3504	0.3905	0.3694	0.0292	0.0323	0.0307	0.3333	0.3714	0.3514
6	0.3162	0.3814	0.3458	0.0351	0.0411	0.0379	0.2991	0.3608	0.3271
7	0.2991	0.3608	0.3271	0.0175	0.0208	0.019	0.2906	0.3505	0.3178
8	0.3248	0.3689	0.3455	0.0234	0.0274	0.0252	0.2991	0.3398	0.3182
T4 - GPU	0.3419	0.3333	0.3376	0.0234	0.0238	0.0236	0.3162	0.3083	0.3122

Figure 7.16: Recalls of Rouge Scores of the selected Summarization results for part of the Essay



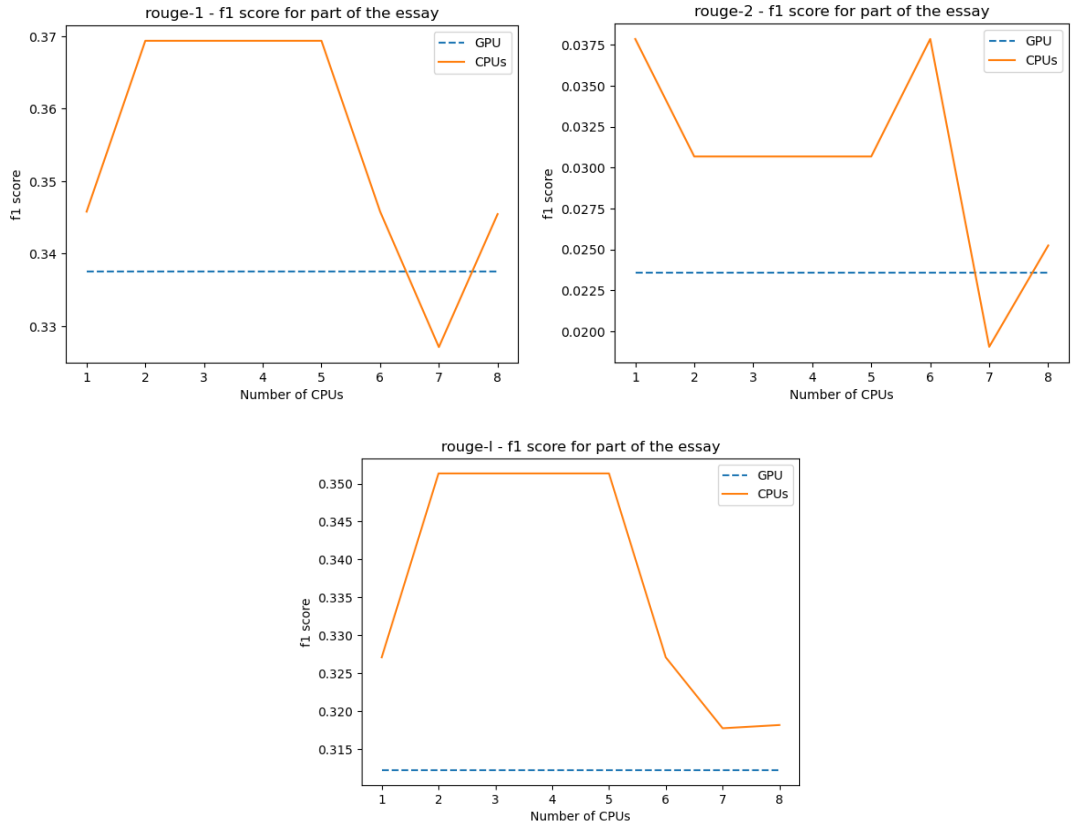
As observed in Figure 7.16, Recalls of Rouge-1 and Rouge-l behave in a similar way and 2,3,4 and 5 machines surpassed values obtained by GPU in both cases For part of the essay . Rouge 2 behave differently and except 7 machines it surpasses GPU results.

Figure 7.17: Precisions of Rouge Scores of the selected Summarization results for part of the Essay



As observed in Figure 7.17, Precisions of Rouge-1 and Rouge-l in CPUs have surpassed results obtained by GPUs for part of the essay. And up to 5 machines both Rouge-1 and Rouge-l in precisions show a similar behavior. However Rouge-2 precisions behave differently similar to recall results of Rouge 2 except 7 machines all the rest precisions have surpassed GPU precisions.

Figure 7.18: F1 Scores of Rouge Scores of the selected Summarization results for part of the Essay



As observed in Figure 7.17, similar behavior was observed in rouge-1 and rouge-l except 7 machines in F1 scores. However Rouge-2 f1 scores behave differently similar to recall and precision results of Rouge 2 except 7 machines all the rest f1 scores have surpassed GPU precisions.

7.5 Llama Model Results for Translations from English to German

7.5.1 Time taken for the three machines for Translations

Table 7.14: Translation results for the three machines for Translating the entire essay

Translation Results for the entire essay

Number of Machines	Machine 1					Machine 2					Machine 3				
	Time Taken	BERTScore			BleuScore	Time Taken	BERTScore			BleuScore	Time Taken	BERTScore			BleuScore
		Precision	Recall	F1			Precision	Recall	F1			Precision	Recall	F1	
1	0:09:14	0.3376	0.3659	0.3512	0.2687										
2	0:06:03	0.3414	0.369	0.3547	0.5278	0:06:00	0.3448	0.3728	0.3582	0.5319					
3	0:06:18	0.3448	0.3728	0.3582	0.5306	0:08:05	0.3376	0.3659	0.3512	0.2695	0:07:40	0.3376	0.3659	0.3512	0.2627
4	0:07:44	0.3376	0.3659	0.3512	0.2639	0:04:44	0.344	0.3722	0.3576	0.6576	0:05:56	0.3451	0.3735	0.3588	0.5143
5	0:04:43	0.3443	0.3734	0.3583	0.6574	0:06:37	0.3376	0.3659	0.3512	0.3381	0:06:09	0.3448	0.3728	0.3582	0.5415
6	0:06:38	0.3448	0.3728	0.3582	0.5302	0:06:46	0.3448	0.3728	0.3582	0.5306	0:06:13	0.3448	0.3728	0.3582	0.5299
7	0:08:09	0.3376	0.3659	0.3512	0.2717	0:06:02	0.3448	0.3728	0.3582	0.5319	0:08:11	0.3376	0.3659	0.3512	0.2751
8	0:08:24	0.3376	0.3659	0.3512	0.2692	0:08:03	0.3376	0.3659	0.3512	0.2687	0:08:00	0.3376	0.3659	0.3512	0.2692
T4 - GPU	0:00:58	0.3379	0.348	0.3429	0.80119										

Table 7.15: Translation results for the three machines for Translating text under maximum tokens of Llama 2

Translation Results for text under maximum tokens of Llama 2

Number of Machines	Machine 1					Machine 2					Machine 3				
	Time Taken	BERTScore			BleuScore	Time Taken	BERTScore			BleuScore	Time Taken	BERTScore			BleuScore
		Precision	Recall	F1			Precision	Recall	F1			Precisi	Recall	F1	
1	0:03:16	0.2538	0.3373	0.2896	0.6135										
2	0:03:38	0.2538	0.3373	0.2896	0.6135	0:02:43	0.2538	0.3373	0.2896	0.6135					
3	0:03:02	0.2538	0.3373	0.2896	0.6135	0:02:44	0.3083	0.3521	0.3287	0.8572	0:03:00	0.3093	0.3533	0.3298	0.8588
4	0:03:48	0.3093	0.3533	0.3298	0.8588	0:02:42	0.3093	0.3533	0.3298	0.8588	0:02:46	0.2538	0.3373	0.2896	0.6135
5	0:03:28	0.3093	0.3533	0.3298	0.8588	0:02:46	0.2538	0.3373	0.2896	0.6135	0:03:00	0.3093	0.3533	0.3298	0.8588
6	0:02:47	0.3093	0.3533	0.3298	0.8588	0:02:41	0.3093	0.3533	0.3298	0.8588	0:03:04	0.3093	0.3533	0.3298	0.8588
7	0:02:46	0.2538	0.3373	0.2896	0.6135	0:02:42	0.2538	0.3373	0.2896	0.6135	0:02:51	0.2538	0.3373	0.2896	0.6135
8	0:02:47	0.3093	0.3533	0.3298	0.8588	0:02:49	0.3093	0.3533	0.3298	0.8588	0:02:55	0.2538	0.3373	0.2896	0.6135
T4 - GPU	0:00:26	0.315	0.3696	0.3401	0.9024										

Table 7.16: Translation results for the three machines for Translating part of the essay

Translation Results for part of the Essay

Number of Machines	Machine 1					Machine 2					Machine 3				
	Time Taken	BERTScore			BleuScore	Time Taken	BERTScore			BleuScore	Time Taken	BERTScore			BleuScore
		Precision	Recall	F1			Precision	Recall	F1			Precisi	Recall	F1	
1	0:03:19	0.2928	0.3517	0.3196	0.4944										
2	0:03:59	0.2928	0.3517	0.3196	0.4944	0:03:03	0.2928	0.3517	0.3196	0.4944					
3	0:01:13	0.2788	0.2932	0.2858	0.7205	0:01:12	0.2788	0.2932	0.2858	0.7205	0:01:21	0.2788	0.2932	0.2858	0.7205
4	0:03:45	0.2928	0.3517	0.3196	0.4944	0:03:05	0.2928	0.3517	0.3196	0.4944	0:01:14	0.2788	0.2932	0.2858	0.7205
5	0:03:04	0.2928	0.3517	0.3196	0.4944	0:03:02	0.2928	0.3517	0.3196	0.4944	0:03:19	0.2928	0.3517	0.3196	0.4944
6	0:03:08	0.2928	0.3517	0.3196	0.4944	0:01:11	0.2788	0.2932	0.2858	0.7205	0:01:18	0.2788	0.2932	0.2858	0.7205
7	0:01:12	0.2788	0.2932	0.2858	0.7205	0:01:14	0.2928	0.3517	0.3196	0.4944	0:03:05	0.2928	0.3517	0.3196	0.4944
8	0:03:08	0.2928	0.3517	0.3196	0.4944	0:03:02	0.2928	0.3517	0.3196	0.4944	0:03:05	0.2928	0.3517	0.3196	0.4944
T4 - GPU	0:00:19	0.2744	0.2892	0.2816	0.7175										

7.5.2 Translation results selected from the best time taken out of the three machines

Table 7.17: Selected Translation Results for the entire essay

Translation Results for the entire essay

Number of Machines	Time Taken	BERTScore			BleuScore
		Precision	Recall	F1	
1	0:09:14	0.3376	0.3659	0.3512	0.268667057
2	0:06:00	0.3448	0.3728	0.3582	0.531886479
3	0:06:18	0.3448	0.3728	0.3582	0.530558499
4	0:04:44	0.344	0.3722	0.3576	0.657575595
5	0:04:43	0.3443	0.3734	0.3583	0.657427632
6	0:06:13	0.3448	0.3728	0.3582	0.529910995
7	0:06:02	0.3448	0.3728	0.3582	0.531879269
8	0:08:00	0.3376	0.3659	0.3512	0.269151806
T4 - GPU	0:00:58	0.3379	0.348	0.3429	0.801190376

Table 7.18: Selected Translation Results for Translating text under maximum tokens of Llama 2

Translation Results for text under maximum tokens of Llama 2

Number of Machines	Time Taken	BERTScore			BleuScore
		Precision	Recall	F1	
1	0:03:16	0.2538	0.3373	0.2896	0.613493124
2	0:02:43	0.2538	0.3373	0.2896	0.613493124
3	0:02:44	0.3083	0.3521	0.3287	0.857154306
4	0:02:42	0.3093	0.3533	0.3298	0.858821596
5	0:02:46	0.2538	0.3373	0.2896	0.613493124
6	0:02:41	0.3093	0.3533	0.3298	0.858821596
7	0:02:42	0.2538	0.3373	0.2896	0.613493124
8	0:02:47	0.3093	0.3533	0.3298	0.858821596
T4 - GPU	0:00:26	0.315	0.3696	0.3401	0.902443137

Table 7.19: Selected Translation Results for Translating part of the essay

Translation Results for part of the Essay

Number of Machines	Time Taken	BERTScore			BleuScore
		Precision	Recall	F1	
1	0:03:19	0.2928	0.3517	0.3196	0.494354836
2	0:03:03	0.2928	0.3517	0.3196	0.494354836
3	0:01:12	0.2788	0.2932	0.2858	0.720511595
4	0:01:14	0.2788	0.2932	0.2858	0.720511595
5	0:03:02	0.2928	0.3517	0.3196	0.494354836
6	0:01:11	0.2788	0.2932	0.2858	0.720511595
7	0:01:12	0.2788	0.2932	0.2858	0.720511595
8	0:03:02	0.2928	0.3517	0.3196	0.494354836
T4 - GPU	0:00:19	0.2744	0.2892	0.2816	0.717520285

7.5.3 Time taken on the Summarization results

Table 7.20 Time taken on the selected Translation results

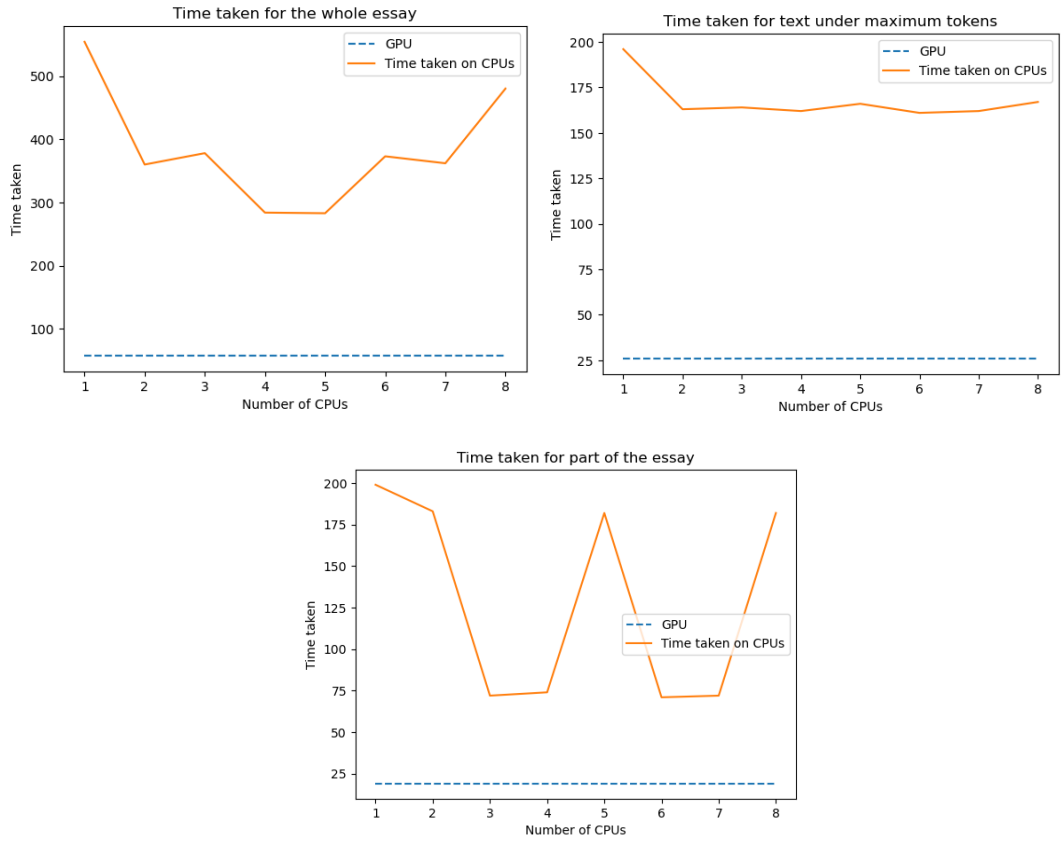
Number of Machines	for the entire essay	for text under maximum tokens of Llama 2	for part of the Essay
1	0:09:14	0:03:16	0:03:19
2	0:06:00	0:02:43	0:03:03
3	0:06:18	0:02:44	0:01:12
4	0:04:44	0:02:42	0:01:14
5	0:04:43	0:02:46	0:03:02
6	0:06:13	0:02:41	0:01:11
7	0:06:02	0:02:42	0:01:12
8	0:08:00	0:02:47	0:03:02
T4 - GPU	0:00:58	0:00:26	0:00:19

When observing the time taken on the translation results as sown in Table 7.18, the lowest time taken for the entire essay is 4 minutes and 43 seconds while running on 5 machines and it was 58 seconds when run on T4 GPU of Google Collab. This lowest time by CPU is nearly 5 times the time taken by the GPU.

When observing results for text under maximum tokens of Llama 2 the lowest time taken is 2 minutes and 41 seconds while running on 6 machines and it was 26 seconds in T4 GPU results. This lowest time by CPU is roughly 6 times the time taken by the GPU.

When observing results for part of the essay the lowest time taken is 1 minutes and 11 seconds while running on 6 machines, and it was 19 seconds when run on GPU. This lowest time by CPU is near 4 times the time taken by the GPU.

Figure 7.19: Time taken on the selected Translation results



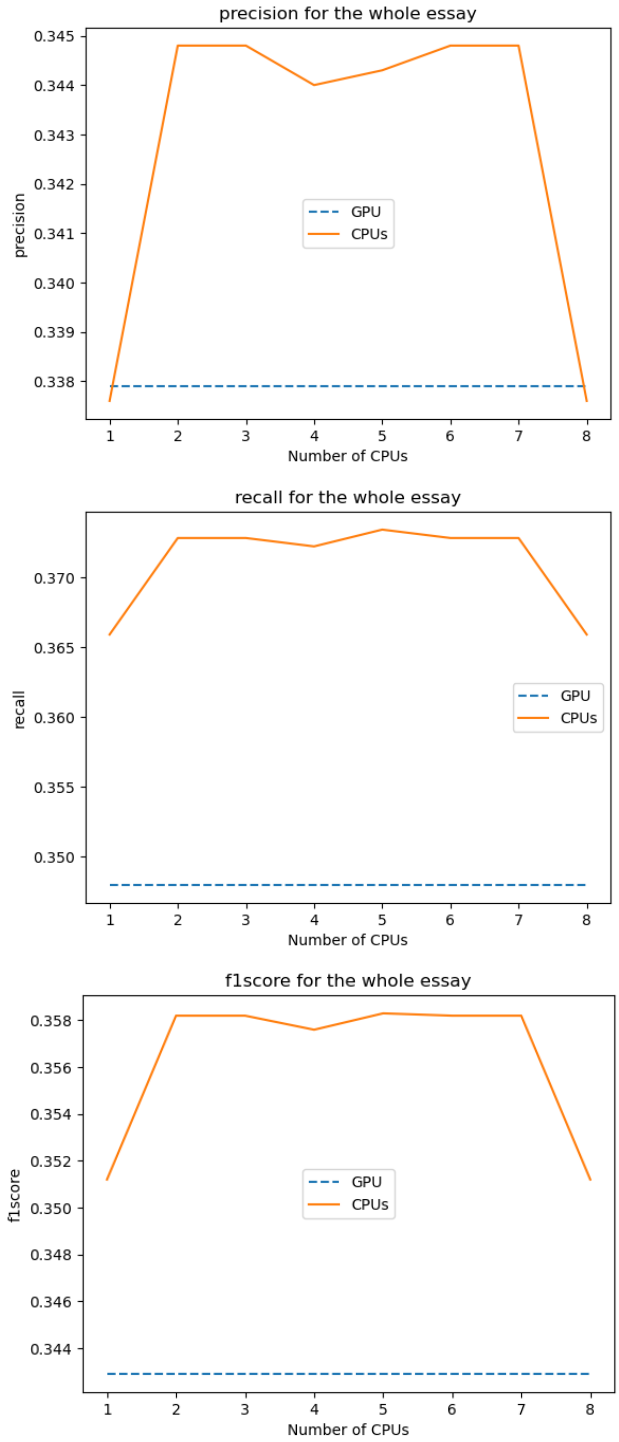
As observed in Figure 7.18, none of the times taken by CPUs were able to surpass time taken by GPUs. And in all three cases except machine 5 rest show a little bit of behavior where time taken by CPUs decreases when machines are increasing and increases again after a certain number of machines.

7.5.4 BERTscores of the Translation results

Table 7.21: BERTscores on the selected Translation results

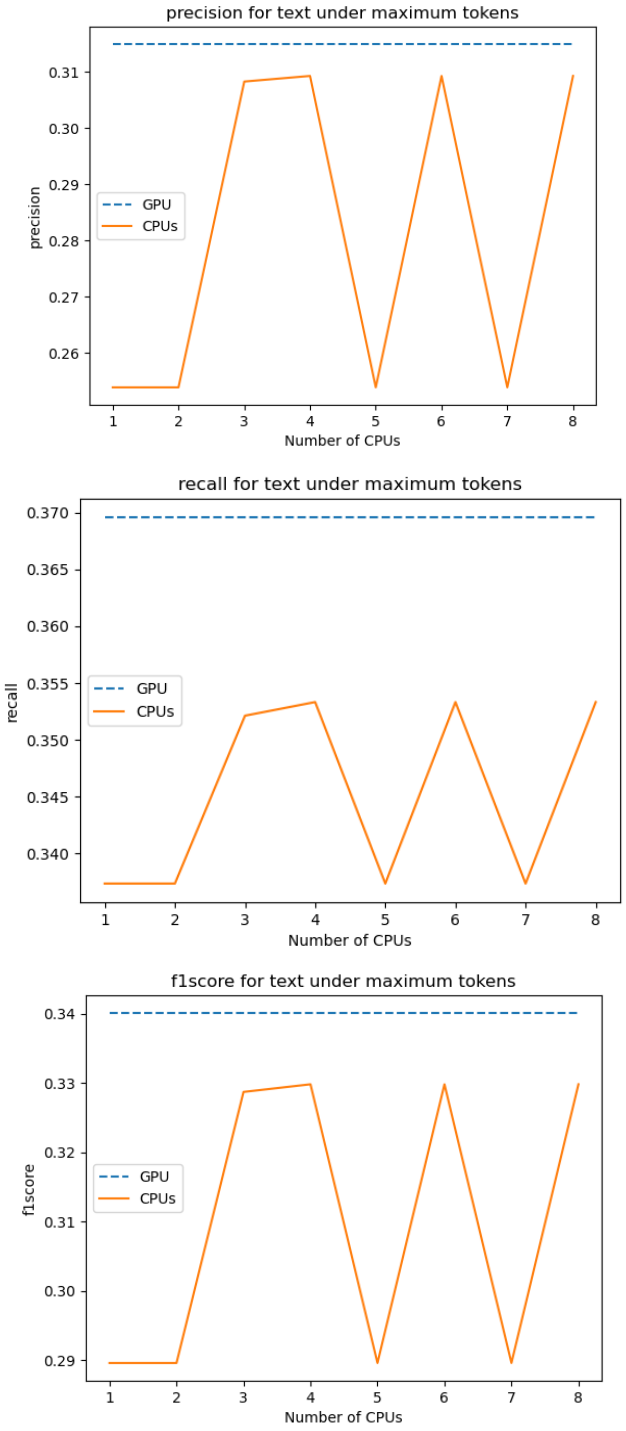
	for the entire essay			for text under maximum tokens of Llama 2			for part of the Essay		
Number of Machines	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
1	0.3376	0.3659	0.3512	0.2538	0.3373	0.2896	0.2928	0.3517	0.3196
2	0.3448	0.3728	0.3582	0.2538	0.3373	0.2896	0.2928	0.3517	0.3196
3	0.3448	0.3728	0.3582	0.3083	0.3521	0.3287	0.2788	0.2932	0.2858
4	0.344	0.3722	0.3576	0.3093	0.3533	0.3298	0.2788	0.2932	0.2858
5	0.3443	0.3734	0.3583	0.2538	0.3373	0.2896	0.2928	0.3517	0.3196
6	0.3448	0.3728	0.3582	0.3093	0.3533	0.3298	0.2788	0.2932	0.2858
7	0.3448	0.3728	0.3582	0.2538	0.3373	0.2896	0.2788	0.2932	0.2858
8	0.3376	0.3659	0.3512	0.3093	0.3533	0.3298	0.2928	0.3517	0.3196
T4 - GPU	0.3379	0.348	0.3429	0.315	0.3696	0.3401	0.2744	0.2892	0.2816

Figure 7.20: BERTscores of the selected Translation results for the entire essay



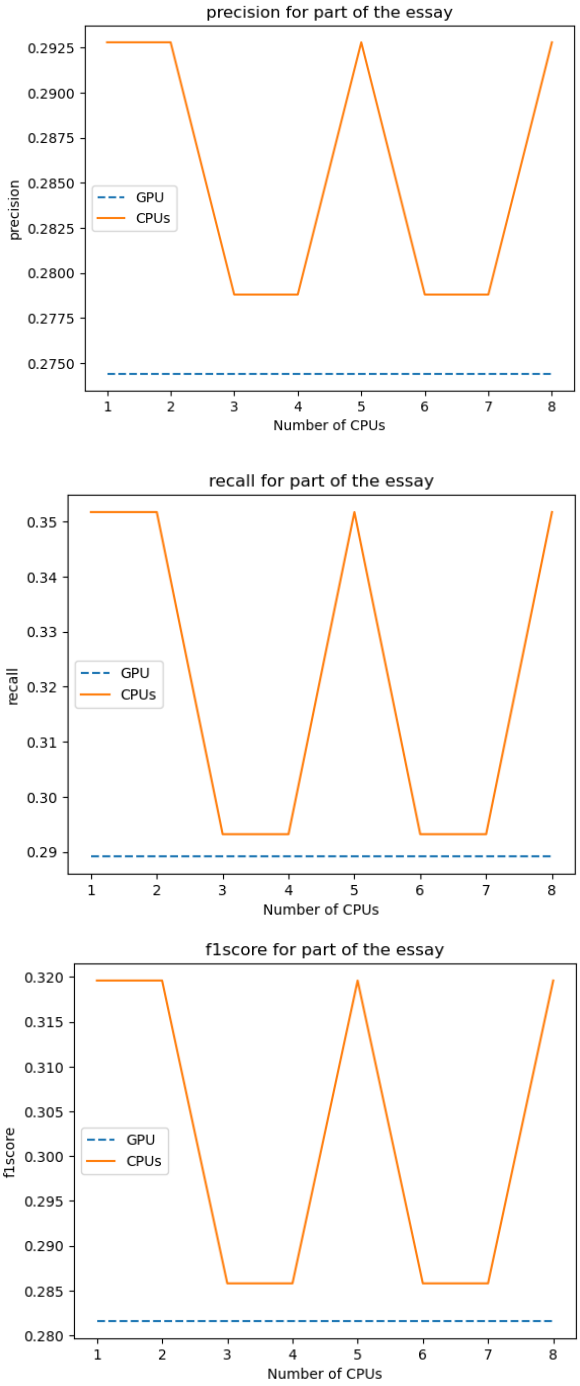
As observed in Figure 7.19, On all precision, recall and f1 scores apart from 1 and 8 machines they were able to surpass the scores obtained when run on GPUs(Except precision) for the whole essay. A little decrease on the BERT scores in 4 machines was also observed in all three cases.

Figure 7.21: BERTscores of the selected Translation results for text under maximum tokens of Llama 2



As observed in Figure 7.20, similar behaviour was observed for all three cases of Bertscores for text under maximum tokens of Llama 2. And none of the CPU results were able to surpass results of GPUs.

Figure 7.22: BERTscores of the selected Translation results for part of essay



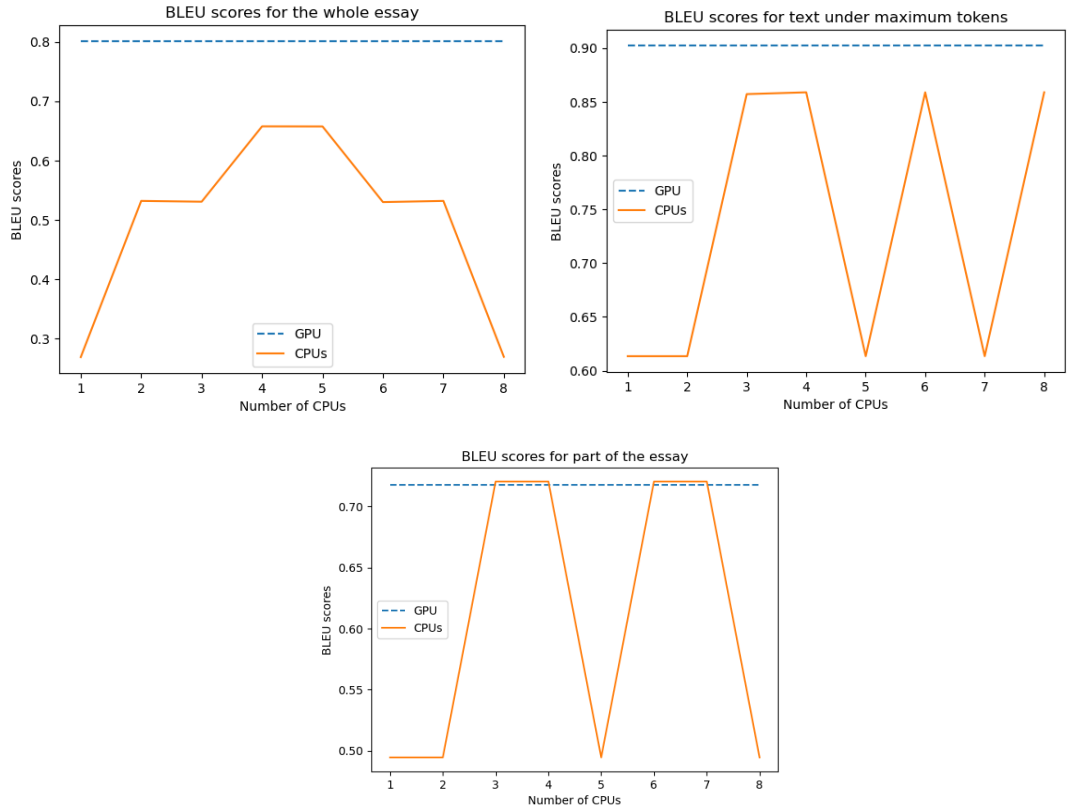
As shown in Figure 7.21, For part of essay all Bertscores precision, recall and F1-scores show a similar behavior and all the CPU results were able to surpass results by GPUs.

7.5.5 BleuScores of the Translation results

Table 7.22: BleuScores on the selected Translation results

Number of Machines	for the entire essay	for text under maximum tokens of Llama 2	for part of the Essay
1	0.2686671	0.6134931	0.494354836
2	0.5318865	0.6134931	0.494354836
3	0.5305585	0.8571543	0.720511595
4	0.6575756	0.8588216	0.720511595
5	0.6574276	0.6134931	0.494354836
6	0.529911	0.8588216	0.720511595
7	0.5318793	0.6134931	0.720511595
8	0.2691518	0.8588216	0.494354836
T4 - GPU	0.8011904	0.9024431	0.717520285

Figure 7.23: BleuScores on the selected Translation results



As shown in Figure 7.21, Except for Bleu scores in part of the essay none of the Bleu Scores were able to surpass values by GPUs.

7.6 Summary

The chapter focuses on the evaluation of three datasets and do a comparative analysis on the time taken and the accuracies. In this scenario though accuracy was overtaken by the Distributed CPU system in many cases, the time taken could not surpass the time taken by GPUs in any case.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Introduction

In the initial chapter, we emphasized the significance of Distributed systems and reviewed previous studies in the problem domain. We defined our problem as though DDL has been efficiently carried out passing the bottlenecks of communication, they still run efficiently only on GPUs which in itself are too costly infrastructure. Chapter 2 provided a thorough evaluation of the available research on the Distributed Systems. We talked about distributed artificial intelligence frameworks, as well as existing and upcoming systems, problems, and directions for distributed deep learning research. In Chapter 3, we provided a detailed description of the technology adopted. Chapter 4 discussed the entire approach of our project, which is divided into six primary sections, including hypothesis, input, output, process, features, and users of the solution, respectively. We explained the underlying principles of the CPU distributed system, data inputs required for the system, expected outcomes, step-by-step procedures for generating the desired outcome, and intended audience and their benefits from the solution. In Chapters 5 and 6, we discussed the design and implementation of the study in detail. We organized this chapter into several sections, including Creating the CPU distributed system, implemented Large Language Model and do a comparative analysis. In Chapter 7, we evaluated our results for the developed system in detail, using various evaluation metrics such as accuracy, time taken and etc. Finally, in the concluding chapter, we summarized our findings and provided recommendations and future works to improve the accuracy and efficiency of the proposed solution.

8.2 Conclusion

Distributed Systems have revolutionized the world today and is being used in various applications like Image Processing, Language modelling etc. However, even though these systems are there almost all of them are based on GPUs. Our objective was to create a CPU based Distributed system that could run a deep learning model just as GPUs thereby reducing our costs and increasing our feasibility. Following the design we build a Distributed CPU system based on the MultiworkerMirroredStrategy by Tensorflow and Keras API. We were able to connect up to 8 machines and we evaluate our system using three different use cases: image processing, Language translation and Summarization.

In this scenario in image processing, if the dataset is small, it showed bizarre outputs in the time taken. Where 02 machines are 100 times slower than a single machine. So according to our study running small datasets in CPU distributed systems is not feasible. However, our second use case CIFAR10 dataset running on a CNN model

showed reasonably good performance such that some accuracies even overcoming the accuracies got from GPUs. When considering the time taken it is still there with difference about 300s. So Due to the communication bottleneck, it is really hard to meet up the speeds of GPUs in all the cases. And also as observed in CIFAR10 implementation, Putting maximum amount of batch size able to process by the CPUs will lead to a more efficient CPU Distributed system.

In our third use case, The Llama model when in both Quantizing and Fine-tunning ran into memory errors, as this is data parallelism the size of the models should be able to fit into the memory. And also as observed in both summarization and translation, when the size of the dataset increases, the best time taken by CPUs decreases when compared with GPUs, For example in summarization time taken decreases from 11 times the GPU time taken for a small dataset that fit maximum token size into the llama model to 6 times of GPU time taken when summarizing the whole essay. And for translation this decreases from 6 times time taken by GPU to 5 times time taken by GPU.

With the bottleneck of communication maximum number of machines connected with good efficiency will be 4, 5 and 6. As for the rest time taken will be much larger

As the case in CIFAR 10 dataset it was unable to reach the accuracies obtained by GPUs. But in the case of Llama 2 precision, recall and f1 scores were able to surpass values obtained by T4 GPU in Google Colab. So we can conclude that when the model is huge the accuracies obtained by Distributed CPU system will surpass the values obtained by GPUs.

The major limitations of this Distributed CPU system are, it is really not possible to change the platform or specifications at your will. In this research Windows 10 Operating System was used with 8 GB of memory for each which is already preinstalled. It will be lot of work if you want to change the platform or increase the RAM to fit the model. And also if 1 machine fail to function properly entire model execution will be stopped and the entire Distributed CPU system fails.

8.3 Further work

We should try the finetuning with Lora and see how they perform. According to our Literature PyTorch performs faster than Tensorflow. But if we could come with a way to parallelize data in multiple machines using PyTorch. It will be a great help for the future. And also, it is a study that is needed to do, to select the optimum number of machines and optimum number of batches per worker to get the best accuracy with the least amount of time taken (04 machines with 128 batch size) which is itself a challenge in this research. Model parallelism should also be tested for further development of this CPU Distributed system

8.4 Summary

Throughout this chapter, we have presented the final thoughts of our study on CPU Distributed Systems. Our study has shown that CPU Distributed Systems have great potential in generating reasonable performance overcoming bottleneck of Communications up to a certain level. We have provided a comprehensive summary of our study findings and discussed the recommendations and future works that can be done to improve our findings. We believe that continued research and development in this area could lead to significant advancements in the field of Distributed Computing, benefiting various stakeholders such as Undergraduates, Researchers and even small companies. We hope that our study's contributions can pave the way for further research in this field and promote the use of Distributed Systems for their large use cases.

REFERENCES

[1] A. F. Agarap, *Training Deep Neural Networks for Image Classification in a Homogenous Distributed System*. 2019. doi: 10.13140/RG.2.2.21940.40321.

[2] H. Xu *et al.*, "Compressed Communication for Distributed Deep Learning: Survey and Quantitative Evaluation," 2020, Accessed: Aug. 25, 2023. [Online]. Available: https://repository.kaust.edu.sa/handle/10754/662495

[3] J. Guo *et al.*, *Accelerating Distributed Deep Learning By Adaptive Gradient Quantization*. 2020, p. 1607. doi: 10.1109/ICASSP40776.2020.9054164.

[4] R. Zhang *et al.*, "LLaMA-Adapter: Efficient Fine-tuning of Language Models with Zero-init Attention." arXiv, Jun. 14, 2023. doi: 10.48550/arXiv.2303.16199.

[5] J. Dean *et al.*, "Large Scale Distributed Deep Networks," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2012. Accessed: Aug. 28, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/hash/6aca97005c68f1206823815f66102863-Abstract.html

[6] H. Mahmud, P. Kang, K. Desai, P. Lama, and S. Prasad, "A Converting Autoencoder Toward Low-latency and Energy-efficient DNN Inference at the Edge." arXiv, Mar. 11, 2024. doi: 10.48550/arXiv.2403.07036.

[7] B. Hasheminezhad, S. Shirzad, N. Wu, P. Diehl, H. Schulz, and H. Kaiser, *Towards a Scalable and Distributed Infrastructure for Deep Learning Applications*. 2020.

[8] R. Mayer and H.-A. Jacobsen, "Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools," no. arXiv:1903.11314. arXiv, Sep. 25, 2019. doi: 10.48550/arXiv.1903.11314.

[9] N. Mungoli, "Scalable, Distributed AI Frameworks: Leveraging Cloud Computing for Enhanced Deep Learning Performance and Efficiency," no. arXiv:2304.13738. arXiv, Apr. 26, 2023. Accessed: Aug. 22, 2023. [Online]. Available: http://arxiv.org/abs/2304.13738

[10] S. Shi, Q. Wang, and X. Chu, "Performance Modeling and Evaluation of Distributed Deep Learning Frameworks on GPUs," no. arXiv:1711.05979. arXiv, Aug. 20, 2018. Accessed: Aug. 28, 2023. [Online]. Available: http://arxiv.org/abs/1711.05979

[11] "(PDF) Performance Analysis and Comparison of Distributed Training Strategies for Deep Learning." Accessed: Aug. 26, 2023. [Online]. Available: https://www.researchgate.net/publication/358552709_Performance_Analysis_and_Comparison_of_Distributed_Training_Strategies_for_Deep_Learning

[12] "Sensors | Free Full-Text | Analysis of the Application Efficiency of TensorFlow and PyTorch in Convolutional Neural Network." Accessed: Feb. 28, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/22/22/8872

[13] "(PDF) Performance Analysis of Data Parallelism Technique in Machine Learning for Human Activity Recognition Using LSTM." Accessed: Aug. 26, 2023. [Online]. Available: https://www.researchgate.net/publication/338946032_Performance_Analysis_of

_Data_Parallelism_Technique_in_Machine_Learning_for_Human_Activity_Recognition_Using_LSTM

[14] J.-S. Lerat, S. A. Mahmoudi, and S. Mahmoudi, “Distributed Deep Learning: From Single-Node to Multi-Node Architecture,” *Electronics*, vol. 11, no. 10, Art. no. 10, Jan. 2022, doi: 10.3390/electronics11101525.

[15] M. Migliorini, R. Castellotti, L. Canali, and M. Zanetti, “Machine Learning Pipelines with Modern Big Data Tools for High Energy Physics,” *Comput. Softw. Big Sci.*, vol. 4, no. 1, Art. no. 1, Dec. 2020, doi: 10.1007/s41781-020-00040-0.

[16] M. Bień, *Big Data Analysis and Machine Learning at Scale with Oracle Cloud Infrastructure*. 2019. doi: 10.5281/zenodo.3550777.

[17] M. H. Anwar, S. Ghafouri, S. S. Gill, and J. Doyle, “Recommender System for Optimal Distributed Deep Learning in Cloud Datacenters,” *Wirel. Pers. Commun.*, vol. 127, no. 2, Art. no. 2, Nov. 2022, doi: 10.1007/s11277-021-08699-3.

[18] Z. Tang, S. Shi, X. Chu, W. Wang, and B. Li, “Communication-Efficient Distributed Deep Learning: A Comprehensive Survey.” arXiv, Mar. 10, 2020. doi: 10.48550/arXiv.2003.06307.

[19] A. Zam, “Evaluating Distributed Machine Learning using IoT Devices”.

[20] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, “SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models,” in *Proceedings of the 40th International Conference on Machine Learning*, PMLR, Jul. 2023, pp. 38087–38099. Accessed: Aug. 28, 2023. [Online]. Available: https://proceedings.mlr.press/v202/xiao23c.html

[21] A. Borzunov *et al.*, “Distributed Inference and Fine-tuning of Large Language Models Over The Internet,” *Adv. Neural Inf. Process. Syst.*, vol. 36, pp. 12312–12331, Dec. 2023.

[22] “tf.distribute.MultiWorkerMirroredStrategy | TensorFlow v2.16.1.” Accessed: Apr. 27, 2024. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/distribute/MultiWorkerMirroredStrategy