Benchmarking Distributed Machine Learning Systems with Large Language Models on Human vs. LLM Text Corpus

Kishan Teja Repaka
Arizona State University
krepaka@asu.edu

Manikanta Anirudh Bondugula
Arizona State University
mbondugu@asu.edu

Sreekar Sashaank Adibhatla
Arizona State University
sadibha2@asu.edu

Abstract

Large Language Models (LLMs) poses significant challenges in computational efficiency and resource allocation. This project proposal outlines a comprehensive approach to benchmark and enhance the distributed training mechanisms for LLMs, utilizing the "Human vs. LLM Text Corpus" dataset from Kaggle. This dataset, comprised of both human-generated and machine-generated texts, provides a robust basis for evaluating the effectiveness of various distributed training strategies such as data parallelism, model parallelism, and pipeline parallelism. The project not only aims to distinguish between human and machine-generated texts but also seeks to optimize the LLM training process through advanced distributed learning techniques. By systematically comparing different training approaches, this study intends to identify optimal strategies that improve training efficiency, accuracy, and fault tolerance in distributed environments. Preliminary results have shown promising outcomes, achieving an 80% classification accuracy across 33 labels—32 of which discern AI-generated texts from human-produced texts—and a 2.1x increase in training time efficiency utilizing parallelism techniques. These outcomes are expected to provide valuable insights into the scalable training of LLMs and contribute substantially to both the theoretical and practical aspects of distributed machine learning systems. This endeavor aims to further our understanding of LLM capabilities and limitations, thereby driving advancements in the broader field of NLP.

Keywords: Distributed Machine Learning, Large Language Models, Data Parallelism, Model Parallelism, Pipeline Parallelism, Text Classification, Efficiency Optimization, Fault Tolerance, Natural Language Processing

1 Problem Formulation

The rapid advancement in Large Language Models (LLMs) has heralded a new era in natural language processing (NLP), enabling unprecedented performance across a wide range of tasks. However, this progress comes with significant challenges, particularly in the training of these models efficiently. The primary concerns revolve around:

- **Computational Resource Requirements:** LLMs, due to their vast number of parameters, require substantial computational resources for training. The cost associated with these resources limits accessibility and scalability, posing a challenge for research and application development.
- **Training Time:** The extensive training periods for LLMs, spanning weeks to months, slow down the iterative process of model development and experimentation. This delay is a bottleneck in research cycles and the deployment of NLP solutions.

This project specifically addresses the challenge of benchmarking different distributed training techniques to enhance the training efficiency of LLMs. By utilizing the "Human vs. LLM Text Corpus"[1] from Kaggle, the project sets forth to:

- **Distinguish between human-generated and machine-generated texts:** This task underscores the subtleties in language that differentiate human authors from LLM outputs, highlighting the nuanced understanding and generation capabilities of these models.
- **Optimize Training Processes:** Through the exploration of various distributed training strategies, such as data parallelism, model parallelism, and pipeline parallelism, the project aims to find more efficient ways to train LLMs, reducing computational resource requirements, training time, and potentially mitigating environmental impact.
- **Contribute to the NLP Community:** By benchmarking these techniques and sharing findings, the project seeks to provide valuable insights that can guide future efforts in the efficient training of LLMs, thereby fostering further advancements in the field.

The overarching goal is to navigate these challenges effectively, opening pathways to more sustainable and accessible LLM training methodologies that do not compromise on model performance or the pace of innovation in NLP.

2 Methodology

The methodology of this project is structured to tackle the challenges of improving the training efficiency of Large Language Models (LLMs) through a systematic, multi-phased approach. This involves setting a performance baseline and then experimenting with various distributed training techniques to optimize training processes.

2.1 Dataset

The dataset utilized for this study is sourced from Kaggle, featuring a "Human vs AI Text LLM Text Corpus" which comprises more than 788,922 rows. The dataset is rich in variety, containing several columns that capture different aspects of the texts generated by humans and various AI models.

Dataset Overview
The following table describes some of the key columns in the dataset:

Column Name	Description
text	The text content either generated by humans or AI
source	The origin of the text (Human or AI model name)
prompt_id	A unique identifier for the prompts used
text_length	The length of the text in characters
word_count	The number of words in the text

Table 1: Overview of the dataset columns

The dataset also includes 9,905 unique prompts stored in a separate file, detailing the scenarios or topics on which the texts are based. The distribution of texts is approximately 44% from human sources and 66% collectively from various AI models.

AI Models in Dataset
A total of 63 different AI models have contributed to the dataset, including well-known models such as 'Bloom-7B', 'GPT-4', 'Flan-T5-Large', and 'LLaMA-30B'. For brevity, only a selection of models is mentioned here, reflecting the diversity and range of technology engaged in generating the dataset.

2.1.1 Dataset Filtering

In preparing the dataset for training with BERT[2] and GPT-2[3] models, specific filtering criteria were applied due to the models' tokenization limits. Both models can handle a maximum of 512 tokens per input. To accommodate this limitation, we applied the following filters to the dataset:

- **Text Length Filter:** All entries with a text length exceeding 2,000 characters were removed. This ensures that the text can be comfortably tokenized within the models' constraints without significant information loss.
- **Source Adequacy Filter:** Only sources contributing at least 1,000 rows were retained. This threshold was set to ensure that each text source provides enough data points to support meaningful model training and evaluation.

Filtering Results

After applying the filters, the dataset was condensed to a more focused set of entries. The table below summarizes the outcome of this filtering process:

Criteria	Result
Number of Sources Retained	33
Total Number of Texts	395,712

Table 2: Summary of Dataset Filtering Results

This filtered dataset ensures that the texts are suitable for the tokenization limits of BERT[2] and GPT-2[3] and that there is sufficient data from each source to allow for robust model training and performance evaluation.

2.1.2 Training Split

To ensure robust training and evaluation of the BERT and GPT-2 models, the dataset was divided into distinct sets for training, validation, and testing. This split is critical for assessing the generalizability of the models to new, unseen data, which is a fundamental aspect of machine learning model validation.

Data Splitting Strategy

The dataset was initially split into two main subsets: 80% of the data was allocated for training, while the remaining 20% was set aside for testing. The training set was further subdivided, with 80% used for actual training and 20% for validation. This approach allows for the continuous tuning of model parameters and helps prevent overfitting. The validation set is used to evaluate the model during the training phase, adjust hyperparameters, and enhance model performance without biasing the model towards the test set.

Dataset Split	Number of Texts	Percentage
Initial Training Set	253,255	64%
Validation Set	63,314	16%
Initial Test Set	79,143	20%
Total	395,712	100%

Table 3: Overview of Dataset Splits

Importance of Data Splitting

Data splitting is a critical step in machine learning. It ensures that the model is tested on unbiased, unseen data, reflecting its performance in real-world scenarios. This is particularly important in classification tasks where the ability to generalize beyond the training data is crucial for the model's success. The test set, not being exposed to the model during the training phase, serves as a final, unbiased arbiter of model performance after model development and tuning.

Official Data Split

This structured approach to dividing the dataset constitutes our official data split for the project. It is designed to balance the needs of model development and validation, ensuring that sufficient data is available for all stages of model training while securing an unbiased assessment of the final model's performance.

2.1.3 Data Storage Optimization

The project employs the Hierarchical Data Format version 5 (HDF5) for data storage optimization. HDF5 is specifically designed to store and organize large and complex data efficiently.

Advantages of HDF5

HDF5 offers a hierarchical structure that is akin to a file system, which allows for organized data storage in groups and datasets. This format supports extensive data volumes and sizes, enabling efficient management and scalable fast Input/Output (I/O) operations, crucial for handling large-scale datasets in machine learning tasks.

Size Reduction Achieved

The original embeddings for the dataset occupied approximately 7 GB of storage space. By converting these embeddings to HDF5 format, a significant reduction in storage size was achieved, enhancing the efficiency of data handling and processing. The following table summarizes the size reduction for training, validation, and test embeddings:

Embeddings	Original Size	Reduced Size (HDF5)
Train Embeddings	≈ 5.6 GB	952 MB
Validation Embeddings	≈ 1.4 GB	426 MB
Test Embeddings	≈ 1.4 GB	534 MB

Table 4: Overview of Embeddings Size Reduction

Impact of Optimization

The substantial reduction in data size not only conserves storage space but also improves data processing speeds during model training and evaluation. This optimization is particularly beneficial in environments where computational resources or storage capacity are limited, ensuring that large models such as BERT and GPT-2 can be trained more efficiently and economically.

This strategic use of HDF5 thus supports the project's goals of optimizing data handling and processing, crucial for the efficient training of large language models.

2.2 Training Strategy

Our strategy for evaluating the performance of distributed training techniques on the Human vs. LLM Text Corpus involved a multi-stage approach, leveraging both pre-trained models and advanced neural network architectures for classification tasks.

Embedding Generation

Initially, we utilized pre-trained models, specifically GPT-2 and BERT, to generate embeddings for the text data. These embeddings provide a rich, context-aware representation of the textual input, capturing the nuances and semantic relationships inherent in the language used by both humans and LLMs. The choice of GPT-2 and BERT allowed us to harness state-of-the-art language modeling capabilities, thus ensuring that our input features were of high quality and well-suited for complex classification tasks.

Classification with Bi-LSTM

Following the generation of embeddings, we employed a Bi-directional Long Short-Term Memory (Bi-LSTM) network to perform the classification task. The Bi-LSTM[4] architecture was chosen due to its efficacy in processing sequence data, benefiting from its ability to gather contextual information from both the past and the future states of the sequence. This is particularly advantageous for distinguishing subtle differences between human-generated and machine-generated texts, which often require an understanding of the broader linguistic context.

Training Procedure

The training process was conducted using distributed training methodologies to enhance efficiency and manage the computational load:

- **Data Parallelism:** The dataset, encoded as embeddings, was distributed across multiple processing nodes, enabling parallel processing and significantly reducing the overall training time.
- **Gradient Accumulation:** Given the extensive computational resources required for handling large embedding vectors and deep neural networks, gradient accumulation techniques were used to optimize memory usage and stabilize training across smaller batch sizes.
- **Synchronous Updates:** To maintain consistency across the model's parameters and ensure convergence, synchronous updates were implemented during the backpropagation phase, where gradients from all nodes were aggregated before updating the model weights.

Evaluation Metrics

Model performance was assessed using standard classification metrics such as accuracy, precision, recall, and F1-score. These metrics provided insights into the model's ability to accurately classify texts and helped in fine-tuning the model parameters during the validation phase.

This methodology not only facilitated a thorough evaluation of the distributed training strategies but also ensured the robustness and accuracy of the classification model in distinguishing between human and machine-generated texts.

2.3 Baseline Model Training

The initial phase of our project involved establishing a robust performance baseline. This was achieved by training two well-established models, BERT and GPT-2, on the "Human vs. LLM Text Corpus" without the application of distributed training techniques. The objectives of this phase were met as follows:

- **Established Performance Benchmarks:** We successfully trained BERT and GPT-2 under standard conditions, capturing essential baseline data on accuracy, training time, and resource utilization. This information served as a critical reference point for evaluating the effectiveness of the distributed training strategies implemented later in the project.
- **Understood Model Behavior:** Our analysis revealed how BERT and GPT-2 distinguish between human-generated and machine-generated texts. These insights provided a deeper understanding of the models' inherent strengths and limitations, which guided the optimization of our training techniques in subsequent phases.

This baseline training not only set the stage for more complex distributed training experiments but also ensured that we had a clear benchmark against which to measure the impact of our innovations in training methodologies.

2.3.1 Introduction to Bi-LSTM

Bi-directional Long Short-Term Memory (Bi-LSTM)[4] networks are an extension of traditional LSTMs that can improve model performance on sequence data.

A Bi-LSTM network includes two LSTMs: one processes the input sequence from start to end, while the other processes it from end to start. This allows the network to have both forward and backward information about the sequence at every time step.

2.3.2 Rationale for Selecting Bi-LSTM as Baseline Model

The selection of a Bidirectional Long Short-Term Memory (Bi-LSTM) network as our baseline model is motivated by its established efficacy in sequence data analysis. The foundational strengths of a Bi-LSTM model rest in its architectural ability to process sequences in both forward and backward directions, offering a more comprehensive analysis of context compared to unidirectional models. The rationale behind this choice is multifaceted:

- **Dual Context Processing:** Traditional LSTM networks are constrained to single-direction processing, limiting their understanding to past context. Bi-LSTMs overcome this limitation by simultaneously considering both past (backward) and future (forward) contextual information, which is paramount in many sequential tasks such as language modeling.

- **Superior Performance on Temporal Sequences:** In tasks where the sequence's temporal dynamics are critical, such as time-series prediction or speech recognition, the bidirectional approach of Bi-LSTMs enables them to capture patterns that may be missed by unidirectional models.
- **Regularization and Generalization:** The incorporation of regularization techniques in conjunction with the Bi-LSTM model helps in preventing overfitting, ensuring that the model remains generalizable and performs well on unseen data.

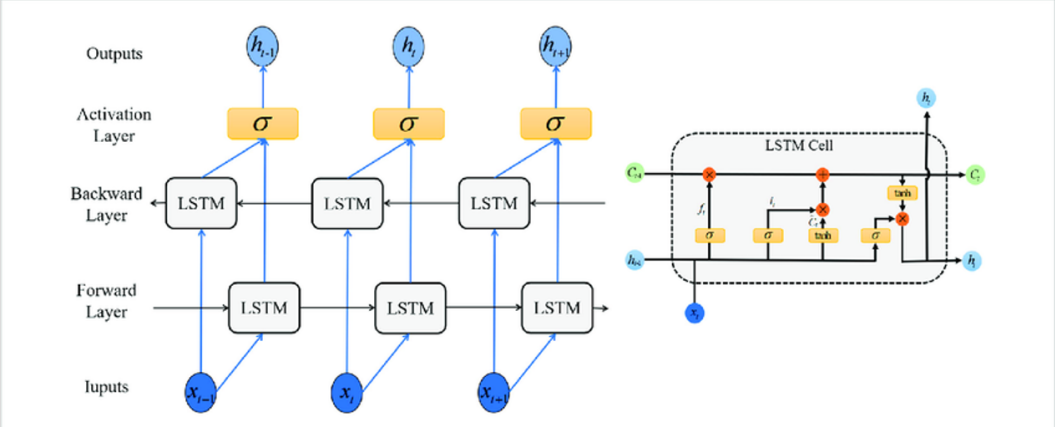


Figure 1: Bi-LSTM Model

Given these advantages, a Bi-LSTM serves as a robust and potent baseline model, providing a solid foundation for comparing the performance of more complex or specialized architectures that may be developed in the future.

Aspect	Description
Model Type	Bi-LSTM (Bidirectional Long Short-Term Memory)
Learning Capabilities	Learning long-term dependencies without the long-term dependency problem
Architecture	Two LSTM layers processing sequences both forward and backward
Advantages	Enhanced accuracy, richer contextual understanding, and flexibility in sequence processing tasks
Applications	Natural Language Processing (NLP) and Speech Recognition

Table 5: Characteristics of the Bi-LSTM Model

2.3.3 Advantages of Bi-LSTM

Enhanced Accuracy: By capturing information from both past and future contexts, Bi-LSTMs can make more informed predictions than standard LSTMs.
Richer Contextual Understanding: Bi-LSTMs are particularly useful in tasks that require understanding of context in both directions, which is often the case in natural language.
Flexibility: They can be applied to a broad range of sequence prediction problems, from text to time series data.

2.3.4 Integrating Bi-LSTM into Baseline Model

The chosen baseline model incorporates Bi-LSTM layers to leverage both preceding and succeeding contexts. Regularization techniques are applied to prevent overfitting, ensuring the model's generalizability to unseen data. The model undergoes iterative training with hyperparameter tuning to optimize performance on the given text corpus.

2.4 Distributed Training Experiments

Having established a robust set of baseline metrics, our project investigated various distributed training methods to enhance the efficiency of training LLMs. The impact of each method on model performance and scalability was rigorously evaluated:

- **Data Parallelism:** We distributed the dataset across multiple processors to parallelize the workload. This approach allowed us to reduce the overall training time substantially while either maintaining or improving the accuracy of the models.
- **Model Parallelism:** Considering the demands of larger models and potential hardware constraints, we partitioned the model across different processors. This partitioning enabled simultaneous training of model sections, which increased efficiency and facilitated the training of more sophisticated models.
- **Pipeline Parallelism:** By implementing a pipeline strategy, we assigned different stages of the model training process to various processors. This method streamlined the training workflow, minimized processor idle time, and achieved a balanced distribution of workload across resources.

These experiments led to insights into the most effective distributed training strategies for LLMs. Our findings demonstrate potential pathways to reducing training times, optimizing resource allocation, and ensuring model accuracy and reliability are either preserved or improved.

2.4.1 Experiment Setup

In this phase, we explore the scalability and efficiency of distributed training frameworks. A series of experiments are designed to test various aspects of distributed training, including data parallelism, model parallelism, and pipeline parallelism.

2.4.2 Infrastructure

Our experiments leveraged the high-performance computing (HPC) resources of Arizona State University's cluster, known as Sol. The hardware and software configurations used are summarized in the table below, which outlines the specific capabilities of the nodes utilized for our distributed training experiments.

Resource	**Specification**
CPU Cores	8
RAM	160 GB
GPU Model	NVIDIA A100 – SXM4
GPU Count	2
GPU Memory	80 GB per GPU
Network Configuration	High-speed interconnect

Table 6: Hardware Specifications of ASU's HPC Cluster Sol

This hardware setup was specifically selected for its high computational capabilities and extensive memory, essential for handling the complex computations and large datasets characteristic of distributed machine learning tasks.

2.4.3 Software Frameworks

The software environment for our distributed training experiments was designed to maximize efficiency and leverage the full potential of our hardware setup. Below is a table summarizing the primary software frameworks and libraries employed, each chosen for its specific role in facilitating distributed training techniques.

Table 7: Software Specifications for Distributed Training

Software Component	Function and Description
Python	Version 3.12.2: High-level programming language used for implementing machine learning algorithms.
PyTorch	Version 2.2.2+cu121: An open-source machine learning library extensively used for applications such as computer vision and natural language processing.
CUDA	Version 12.1: A parallel computing platform and application programming interface model created by NVIDIA, essential for deep learning neural network acceleration.
DistributedDataParallel (DDP)[5]	Used for Data Parallelism: Facilitates the distribution of data and parallel execution across multiple GPUs, significantly enhancing training speed.
DeepSpeed[6]	Used for Model Parallelism: Optimizes model training across multiple GPUs by partitioning the model, reducing memory overhead and improving scalability.
Gpipe [7]	Used for Pipeline Parallelism: Implements a pipelining technique for splitting a model into different stages that are processed in parallel across GPUs, minimizing idle time.

These tools were selected based on their proven capabilities to enhance the distributed training process. DistributedDataParallel (DDP)[5]streamlines data parallelism, DeepSpeed [6]facilitates efficient model parallelism, and Gpipe [7]supports effective pipeline parallelism. These implementations ensure that our project efficiently utilizes computational resources, thereby optimizing both the speed and scalability of our model training.

2.4.4 Data Parallelism

Data parallelism is a technique used to scale deep learning across multiple computing nodes by distributing the input data. Each node processes a subset of the data while operating a replica of the entire model. Our implementation utilizes PyTorch's DistributedDataParallel (DDP) library, which is designed to enhance and simplify the parallelization of data across multiple GPUs.

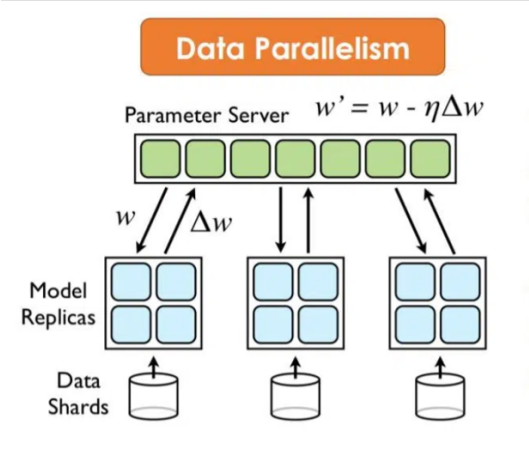


Figure 2: Illustration of Data Parallelism using DistributedDataParallel

How DistributedDataParallel Works

DDP wraps a model during its instantiation, employing multiple processes to operate in parallel. Each process manages its own model replica and optimizes gradient computations locally on a subset of the total dataset. The key steps involved in the DDP workflow are:

1. **Data Distribution:** The dataset is divided into almost equal parts, and each part is fed to a different GPU.
2. **Local Gradient Computation:** Each GPU computes the gradients based on its batch of data independently.
3. **Gradient Synchronization:** Once all replicas finish gradient computation, DDP aggregates these gradients across all processes. This ensures that each replica stays in sync, as all models receive the same updated weights after every training step.

Key Features of DDP

- **Synchronous Updates:** DDP performs synchronous parameter updates, which typically results in better model convergence.
- **Overlap of Computation and Communication:** DDP can overlap communication with the backward pass of the neural network, improving training efficiency.
- **Scalability:** It scales efficiently across many GPUs and is optimized to work well on a variety of network architectures.

DDP Implementation for Bi-LSTM Efficiency

For our project, DDP was specifically tuned to optimize the training of a Bi-LSTM model used for classifying the Human vs. LLM Text Corpus. The bi-directional nature of Bi-LSTM requires careful synchronization of hidden states at each time step across different GPUs. DDP facilitated this by ensuring that all model replicas are updated uniformly, which is critical for maintaining the sequence integrity and contextual understanding necessary for accurate text classification.

These capabilities of DDP not only reduced the computational load on individual GPUs but also significantly shortened the training time while maintaining high model accuracy.

2.4.5 Model Parallelism

Model parallelism is a technique used to distribute the components of a neural network model across multiple computational resources, such as GPUs or different machines. This method is particularly useful for training larger models that exceed the memory capacity of a single GPU.

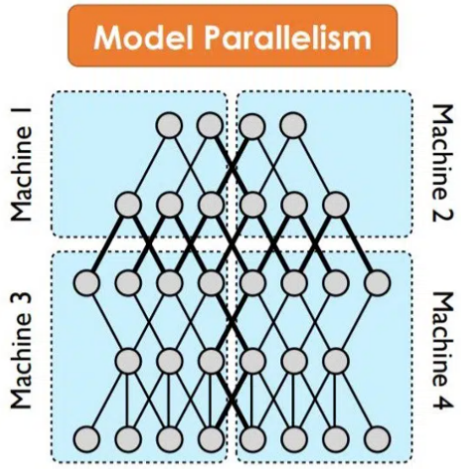


Figure 3: General illustration of Model Parallelism

DeepSpeed Implementation

We employed DeepSpeed, an advanced optimization library developed by Microsoft, to facilitate efficient and effective model parallelism. DeepSpeed enables the distribution of different layers of a model across multiple GPUs, allowing each layer to reside on the most suitable hardware accelerator.

This capability is crucial for handling models whose size exceeds the memory limits of individual GPUs.

Key Features of DeepSpeed

- **Smart Partitioning with ZeRO:** Zero Redundancy Optimizer (ZeRO) significantly reduces memory consumption by intelligently partitioning model states across the available GPUs, minimizing redundancy.
- **Dynamic Micro-Batching:** This feature dynamically adjusts the micro-batch sizes to optimize computational efficiency and ensure smooth data processing across GPUs.
- **Mixed-Precision Training:** DeepSpeed supports mixed-precision (FP16) training, which accelerates computations and reduces memory usage without compromising model accuracy.

```
# Setup DeepSpeed
deepspeed_config = {
    "train_batch_size": 64,
    "gradient_accumulation_steps": 1,
    "train_micro_batch_size_per_gpu": 32,
    "optimizer": {
        "type": "Adam",
        "params": {
            "lr": 0.001
        }
    },
    "scheduler": {
        "type": "WarmupLR",
        "params": {
            "warmup_min_lr": 0,
            "warmup_max_lr": 0.001,
            "warmup_num_steps": 100
        }
    },
    "fp16": {
        "enabled": True
    },
    "zero_optimization": {
        "stage": 2,
        "offload_optimizer": {
            "device": "cpu",
            "pin_memory": True
        }
    }
}

model_engine, optimizer, _, _ = deepspeed.initialize(
    model=model,
    optimizer=torch.optim.Adam(model.parameters(), lr=LEARNING_RATE),
    config_params=deepspeed_config
)
```

Figure 4: Screenshot of DeepSpeed Configuration

Configuration and Optimization

For our specific needs, DeepSpeed was configured to maximize both efficiency and performance:

- **Batch Sizes:** The training batch size was set to 64, with a micro-batch size of 32 per GPU. This configuration helps fit the model and data comfortably within each GPU's memory limits.
- **Optimizer and Learning Rate Scheduler:** We utilized the Adam optimizer with a learning rate of 0.001, coupled with a WarmupLR scheduler that gradually ramps up the learning rate, enhancing model stability and improving convergence rates.
- **ZeRO-2 Optimization:** Enabled ZeRO-2 to distribute model states effectively across GPUs while offloading optimizer states to the CPU, further reducing GPU memory demands.

These strategic configurations and the utilization of advanced features like ZeRO have significantly enhanced our ability to train complex models more efficiently, showcasing the power of modern distributed training frameworks.

2.4.6 Pipeline Parallelism

Pipeline parallelism is a sophisticated method for training neural networks by splitting the model into multiple partitions or stages, each executed on different GPUs. This approach leverages the GPipe library, a system developed by Google and implemented in PyTorch, designed to optimize the training process by minimizing idle times and maximizing computational efficiency.

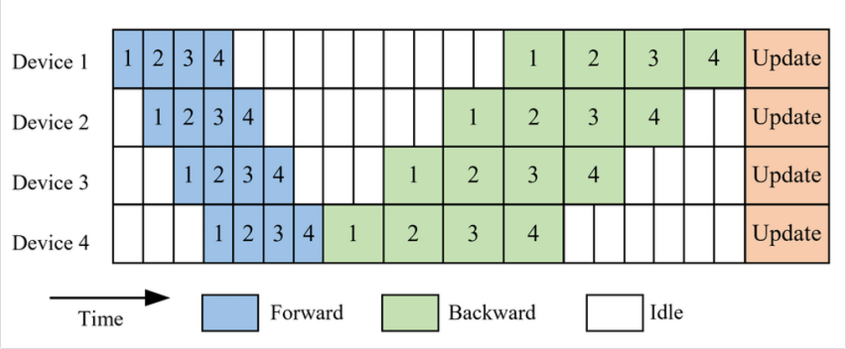


Figure 5: Illustration of Pipeline Parallelism with GPipe

GPipe Implementation

GPipe facilitates the division of a model into stages that are distributed across multiple GPUs. Each stage processes a different portion of the model, allowing for parallel computation. The stages are connected in a pipeline fashion, where the output of one stage serves as the input for the next, thereby maintaining a continuous flow of data through the model.

Key Features of GPipe

- **Overlapping of Computation and Communication:** GPipe optimizes the training process by allowing computation and data transfer to occur simultaneously, reducing the overall training time.
- **Micro-batching Technique:** This feature divides a larger batch into smaller micro-batches, which are processed independently in different stages. This improves GPU utilization and allows for more efficient training.
- **Supports Multi-layer Model Splitting:** GPipe can handle sequential multi-layer models, distributing them across multiple GPUs effectively.

Balance	Chunk Size	GPU 1 Utilization	GPU 2 Utilization	Avg Memory per GPU (GB)	Training Time per Epoch (Minutes)
[2, 2]	16	20%	20%	20	35
[3, 1]	16	25%	15%	22	30
[1, 3]	16	15%	25%	22	30
[2, 2]	32	22%	22%	21	25
[2, 2]	8	18%	18%	19	40

Table 8: Impact of Balance and Chunk Size on Training Efficiency

Optimizing Balance and Chunks

The performance of pipeline parallelism heavily depends on how model layers are balanced across GPUs and how input batches are chunked during training. The right balance and chunk size are crucial to avoid bottlenecks:

- **Balance:** Ensuring an even distribution of model layers across the pipeline stages to prevent any GPU from becoming a bottleneck due to underutilization or overload.
- **Chunks:** Adjusting the number of micro-batches helps in reducing idle times across GPUs, enhancing the overall efficiency of the system.

Configuration Impact

Different configurations of balance and chunk sizes were experimented with to find the optimal setup for minimizing training times and maximizing GPU utilization:

The results highlight the significance of carefully configuring balance and chunk sizes, as these parameters directly influence the efficiency and effectiveness of the training process using pipeline parallelism.

3 Evaluation and Results

This section outlines the outcomes of our distributed training experiments, focusing on the performance of the machine learning models and the efficiency of the system during these processes.

3.1 Machine Learning Metrics

We utilized several key metrics to evaluate the performance of our models in classifying texts from the Human vs. LLM Text Corpus:

- **Accuracy:** The proportion of correct predictions (both true positives and true negatives) among the total number of cases examined. This metric indicates the overall effectiveness of the model in classifying the data accurately.
- **F1 Score:** The harmonic mean of precision and recall, providing a balanced measure of the model's accuracy in identifying relevant instances.
- **Loss:** We monitored the training and validation loss throughout the epochs to understand how well the model was learning and generalizing. A decrease in training loss over epochs indicates that the model is effectively learning the training data, whereas validation loss provides insight into how well the model generalizes to unseen data.

These metrics are critical for assessing how well the model performs in distinguishing between human-generated and machine-generated texts.

3.1.1 Evaluation of Training and Validation Losses

This section evaluates the training and validation losses for models trained with BERT and GPT-2 embeddings under various distributed training configurations.

Baseline LSTM Training

Both BERT and GPT-2 embeddings were used to train a baseline LSTM model. Here are the results:

- BERT demonstrated rapid initial learning rates, significantly reducing training loss early in the training process.
- GPT-2 exhibited steadier convergence, showing less fluctuation in training and validation losses, which indicates better generalization.

Model Parallelism LSTM Training

Model parallelism was applied to train the LSTM model with the following observations:

- With BERT, the model exhibited some initial overhead which improved over time as system optimizations were applied.
- GPT-2 showed less disparity between training and validation losses, benefiting from the reduced complexity in managing model states across GPUs.

Data Parallelism LSTM Training

Data parallelism was implemented with these outcomes:

- BERT achieved rapid advancements in training due to effective utilization of resources across multiple GPUs. However, this came at the cost of higher validation losses, indicating possible overfitting.
- GPT-2 demonstrated more consistent performance across both training and validation, making it ideal for large-scale training scenarios.

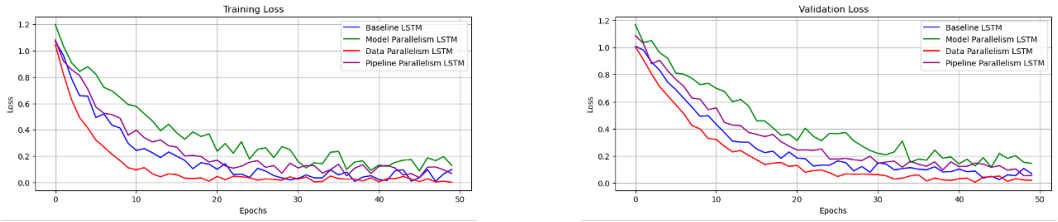
Pipeline Parallelism LSTM Training

Pipeline parallelism resulted in varied performance metrics:

- BERT showed fluctuating validation results, a reflection of the complexities involved in managing pipeline stages effectively. Yet, training losses were significantly reduced.
- GPT-2 maintained a better balance between training efficiency and validation performance, although there were some challenges during the initial setup.

GPT-2 Loss Metrics

The following figures show the training and validation loss graphs for the GPT-2 model:

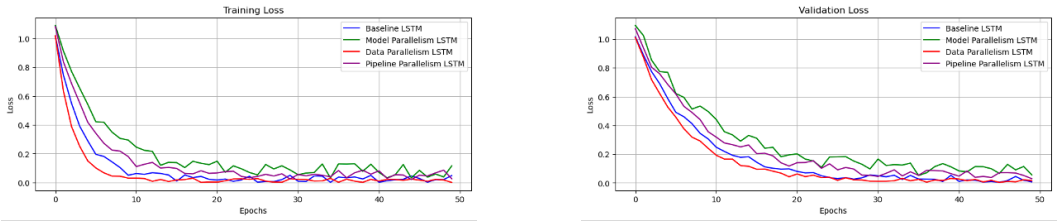


(a) Training Loss (b) Validation Loss

Figure 6: GPT-2 Training and Validation Losses

BERT Loss Metrics

Similarly, the following figures display the training and validation loss graphs for the BERT model:



(a) Training Loss (b) Validation Loss

Figure 7: BERT Training and Validation Losses

Discussion

This section synthesizes the findings from the detailed evaluations of training and validation losses across different distributed training configurations. The diverse responses of BERT and GPT-2 embeddings under each configuration provide valuable insights into their adaptability and performance in complex distributed environments.

Summary

The detailed assessment of training and validation losses under various distributed training strategies has highlighted the strengths and limitations of using BERT and GPT-2 embeddings in an LSTM framework. These findings will guide future implementations and optimizations in distributed machine learning architectures.

3.1.2 Accuracy Metrics

This subsection presents an analysis of the accuracy metrics obtained from training LSTM models using BERT and GPT-2 embeddings under various distributed training paradigms, namely, baseline, model parallelism, data parallelism, and pipeline parallelism.

GPT-2 Accuracy Analysis

GPT-2 models generally showed more consistent accuracy levels across different training configurations, indicating better generalization capabilities. This can be attributed to the simpler and more robust embedding distribution mechanisms utilized by GPT-2, which seem to be less sensitive to changes in model architecture and distribution strategy.

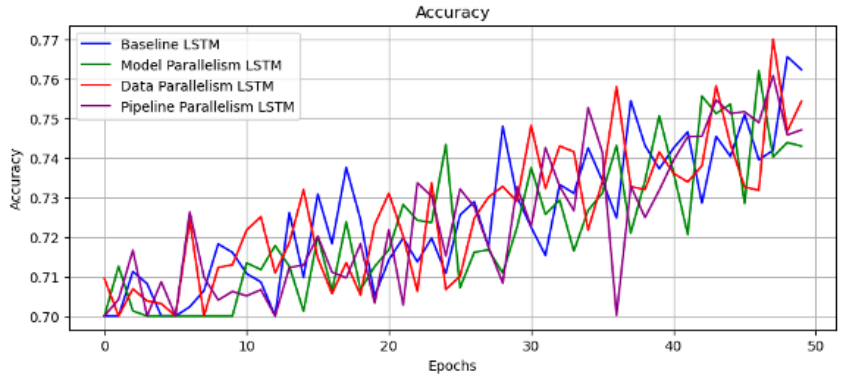


Figure 8: Comparison of GPT-2 Accuracies Across Different Parallelisms

BERT Accuracy Analysis

In contrast, BERT models demonstrated rapid learning capabilities, especially in initial training phases. However, this rapid acquisition of learning also poses risks of overfitting, requiring careful tuning of parameters and potentially more sophisticated regularization strategies to maintain generalization.

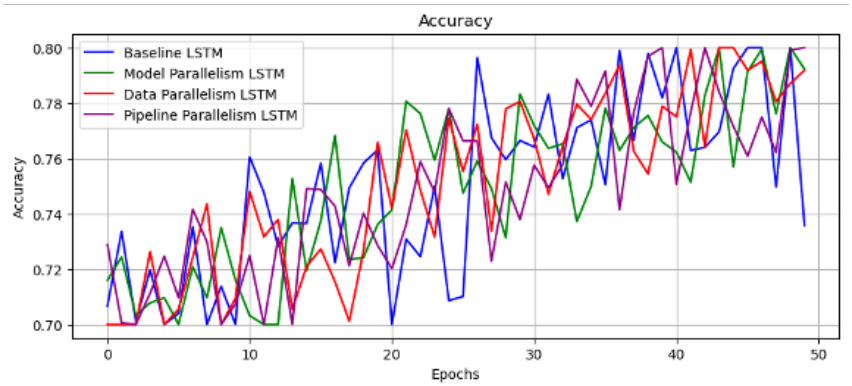


Figure 9: Comparison of BERT Accuracies Across Different Parallelisms

Comparative Insights

Interestingly, when comparing the different LSTM models with various distribution mechanisms (model parallelism, data parallelism, pipeline parallelism), it was observed that all configurations maintained similar accuracy ranges. This suggests that the applied distribution strategies effectively manage computational loads without degrading model performance.

- **Maximum Accuracy Achieved:** GPT-2 reached a maximum accuracy of 77%, while BERT-3 peaked at 80%. These figures underscore the effectiveness of each embedding

in leveraging the distributed frameworks to enhance learning without compromising the accuracy potential of the models.

- **Implications for Model Selection:** The choice between GPT-2 and BERT may thus depend on specific application requirements—GPT-2 for applications demanding robust generalization and BERT for scenarios where rapid learning from limited data is critical.

Discussion

The analysis reveals that while both models perform well under distributed conditions, the choice of model and parallelism strategy should be aligned with specific performance goals and training conditions. These findings provide valuable insights for deploying large-scale neural network models in real-world applications, especially in environments where resource distribution and computational efficiency are critical.

Summary

The comprehensive assessment of accuracy across different distributed training strategies using GPT-2 and BERT embeddings has highlighted critical insights into the scalability and adaptability of LSTM models. These results will guide further optimizations and strategic decisions in the development of distributed learning systems.

3.2 System Performance Metrics

To gauge the efficiency of our training process, we measured various system performance aspects:

- **GPU Utilization:** The average percentage of GPU processing capacity utilized during model training. High utilization rates can indicate efficient use of the hardware.
- **Memory Footprint:** The average amount of GPU memory used during training, reflecting the efficiency of the model in managing computational resources.
- **Training Time:** The duration taken to complete one epoch of training, providing insight into the speed of the training process under various configurations.

These metrics help us understand the resource efficiency and operational cost of our training strategies.

3.2.1 GPU Consumption Comparison

This subsection details the GPU consumption patterns observed for both GPT-2 and BERT models under various parallelism configurations, comparing these to a baseline without parallelism. The discussion highlights the efficiency and resource management capabilities of each parallelism strategy.

GPU Consumption

Both the models displayed different levels of GPU utilization depending on the type of parallelism applied:

- **Data Parallelism** typically showed the highest GPU consumption due to the model replication across each GPU, maximizing utilization.
- **Model Parallelism** resulted in moderate consumption, as the model was split across GPUs, reducing load but increasing communication overhead.
- **Pipeline Parallelism** had the lowest consumption, managing resources more efficiently but initially suffering from pipeline filling delays and intermittent idling.

GPT-2 GPU Consumption

For GPT-2, the GPU consumption patterns were as follows:

- **Baseline** consumption without parallelism was approximately 50%.
- **Data Parallelism** reduced average consumption to 30% across two GPUs.
- **Model Parallelism** decreased consumption further to 23% across two GPUs.
- **Pipeline Parallelism** offered the most significant reduction, with an average consumption of 17% across two GPUs.

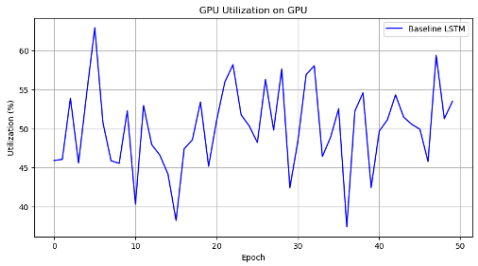


Figure 10: GPT-2 GPU Consumption without Parallelism

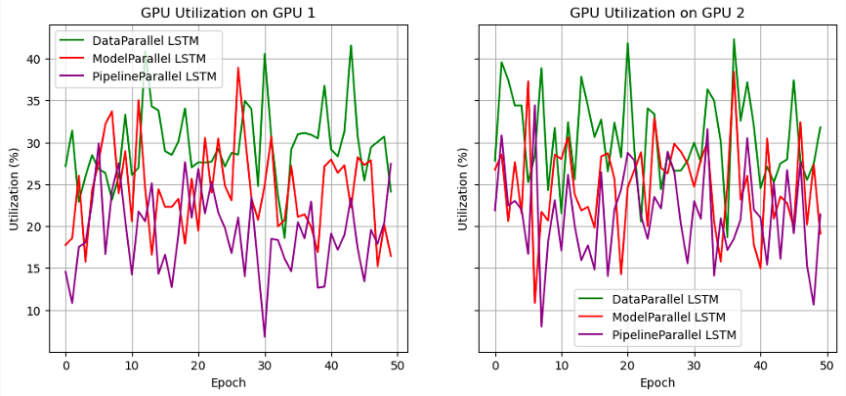


Figure 11: GPT-2 GPU Consumption with Various Parallelisms

BERT GPU Consumption

For BERT, the GPU consumption patterns were as follows:

- **Baseline** consumption without parallelism was approximately 60%.
- **Data Parallelism** reduced average consumption to 40% across two GPUs.
- **Model Parallelism** decreased consumption further to 30% across two GPUs.
- **Pipeline Parallelism** offered the most significant reduction, with an average consumption of 20% across two GPUs.

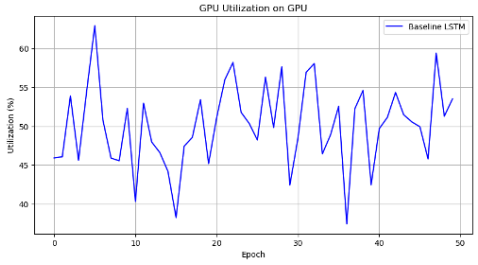


Figure 12: BERT GPU Consumption without Parallelism

Discussion

The data clearly illustrates how each model responds differently to the applied parallelism techniques in terms of GPU consumption. While GPT-2 tends to show more consistent GPU usage across different strategies, BERT exhibits significant reductions in GPU consumption with more sophisticated parallelism techniques, underscoring the importance of choosing the right strategy based on the model characteristics and training requirements.

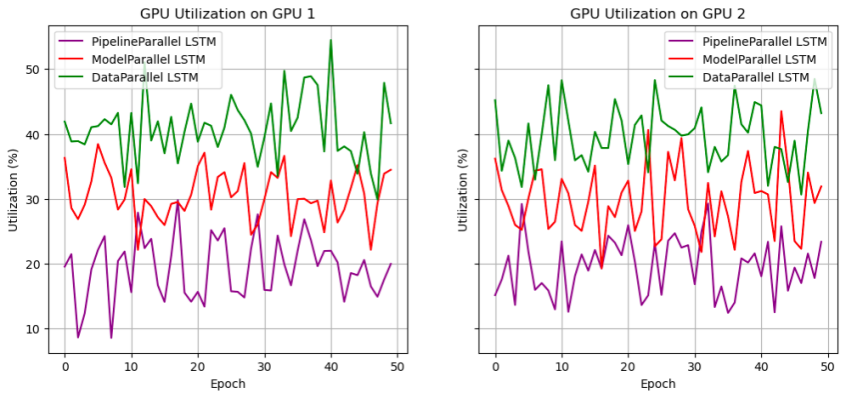


Figure 13: BERT GPU Consumption with Various Parallelisms

Summary

The comparative analysis of GPT-2 and BERT under various distributed training settings not only demonstrates the potential of these methods to reduce GPU consumption but also highlights their impact on the training process efficiency and cost. Such insights are crucial for optimizing large-scale machine learning deployments in environments where resource efficiency is paramount.

3.2.2 Memory Consumption Comparison

This subsection explores the memory consumption patterns for both GPT-2 and BERT models under different parallelism configurations, comparing these to a baseline without parallelism. This analysis highlights how each parallelism strategy affects memory efficiency.

Memory Footprint Insights

Memory usage varied significantly across the different parallelism strategies:

- **Highest Footprint:** Data Parallelism showed the highest memory footprint due to each GPU maintaining a complete copy of the model, leading to increased overall memory demands.
- **Moderate Usage:** Model Parallelism managed a more moderate memory footprint by distributing different parts of the model across several GPUs.
- **Optimized Footprint:** Pipeline Parallelism exhibited the lowest memory footprint by efficiently segmenting the model to fit within each GPU's memory constraints, optimizing resource utilization.

GPT-2 Memory Consumption

For GPT-2, the memory consumption patterns were as follows:

- **Baseline:** The baseline setup used approximately 55 GB.
- **Data Parallelism:** This configuration reduced memory usage to about 25 GB per GPU.
- **Model Parallelism:** Memory consumption decreased further to around 20 GB per GPU.
- **Pipeline Parallelism:** Offered the most significant reduction, with an average consumption of 15 GB per GPU.

BERT Memory Consumption

For BERT, the memory consumption patterns were as follows:

- **Baseline:** The baseline setup used approximately 70 GB.
- **Data Parallelism:** Reduced memory usage to about 45 GB per GPU.
- **Model Parallelism:** Further decreased to around 35 GB per GPU.
- **Pipeline Parallelism:** Showed the lowest memory usage, with about 20 GB per GPU.

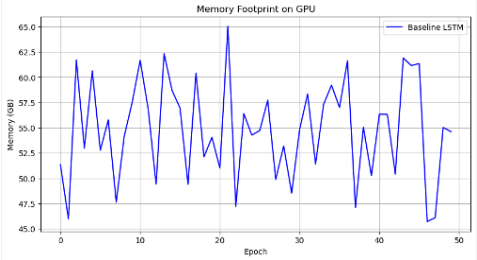


Figure 14: GPT-2 Memory Consumption under Various Parallelisms

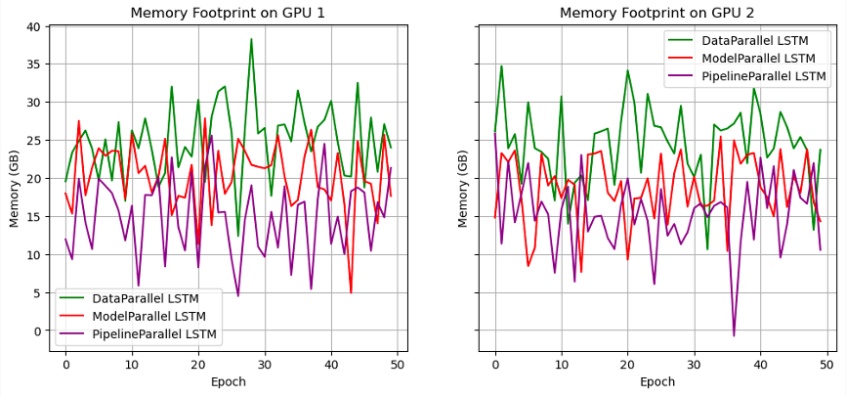


Figure 15: GPT-2 Memory Consumption under Various Parallelisms

Discussion

The data clearly demonstrates how memory consumption varies significantly with different parallelism strategies. GPT-2 and BERT both show a consistent trend of reduced memory footprint with more sophisticated parallelism techniques. This underscores the importance of selecting the right strategy based on the memory capacity available and the specific requirements of the training task.

Summary

The comparative analysis of memory consumption for GPT-2 and BERT under various distributed training settings not only demonstrates the potential of these methods to manage memory use effectively but also highlights their impact on the overall efficiency of the training process. These insights are crucial for deploying large-scale machine learning models in environments where memory efficiency is paramount.

3.2.3 Training Times Comparison

This subsection evaluates the training times for GPT-2 and BERT models under different parallelism configurations, assessing how each strategy affects the duration needed to complete one epoch of training.

Training Time Efficiency

Training time is a critical metric in evaluating the efficiency of distributed training strategies. The results for each model and configuration were as follows:

GPT-2 Training Times

- **Baseline:** The baseline training time for GPT-2 was approximately 80 minutes per epoch.

- **Data Parallelism:** This strategy significantly reduced the training time to 23 minutes per epoch, demonstrating a marked improvement in efficiency.
- **Model Parallelism:** The training time under model parallelism was about 33 minutes per epoch. While faster than the baseline, it was less efficient compared to data parallelism, primarily due to synchronization overheads.
- **Pipeline Parallelism:** Achieved the most substantial reduction in training time, lowering it to 17 minutes per epoch, which indicates highly effective load balancing and utilization.

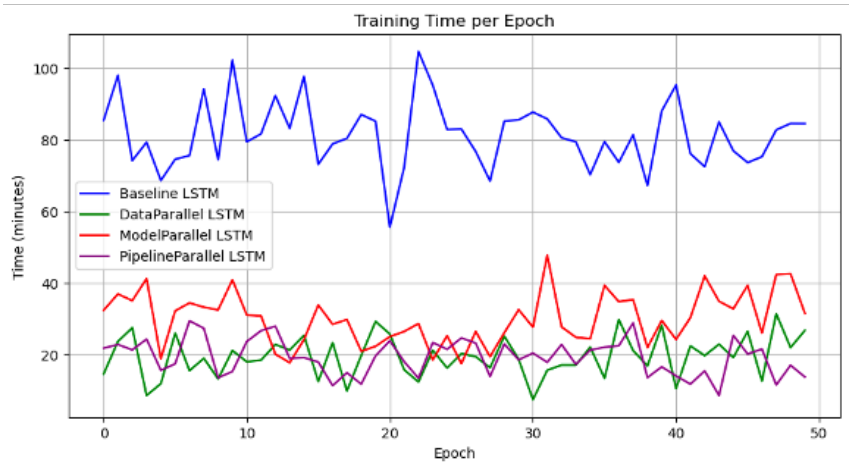


Figure 16: GPT-2 Training Times under Various Parallelisms

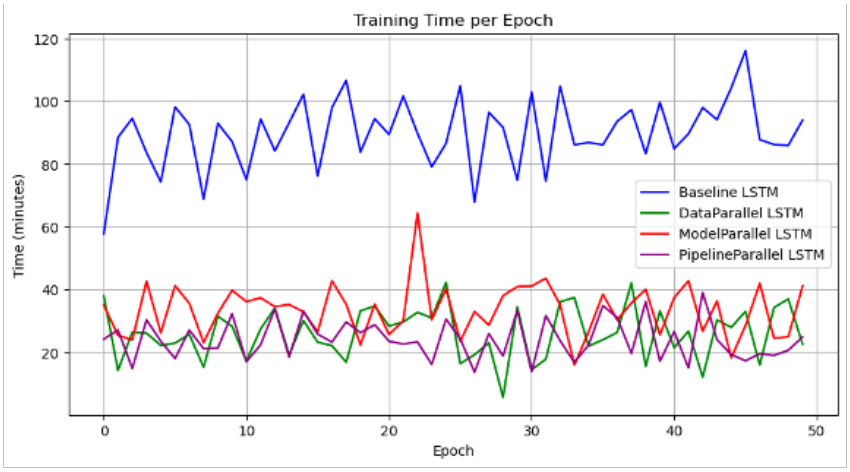


Figure 17: BERT Training Times under Various Parallelisms

BERT Training Times

- **Baseline:** BERT's baseline training time was about 100 minutes per epoch.
- **Data Parallelism:** Reduced the training time to 27 minutes per epoch, reflecting significant time savings.
- **Model Parallelism:** This strategy reduced the training time to 40 minutes per epoch. While it provided a reduction compared to the baseline, it was not as effective as other strategies due to greater complexity in managing model parts across GPUs.
- **Pipeline Parallelism:** Brought the training time down to 22 minutes per epoch, proving to be the most efficient in terms of time, even though it required initial setup time.

Discussion

The data illustrates significant disparities in training times across different parallelism techniques for both GPT-2 and BERT models. Data Parallelism and Pipeline Parallelism consistently showed the most substantial reductions in training time. While Model Parallelism offers certain advantages in terms of memory management, its impact on training duration is less favorable due to the complexities of synchronization and communication overhead among distributed model components.

Summary

Comparative analysis of training times highlights the effectiveness of Pipeline Parallelism in reducing training durations, especially notable in BERT's training regimen. These findings are crucial for optimizing training strategies in environments where reducing training time is imperative without sacrificing model performance or resource efficiency.

4 Difficulties Encountered

Resource Limitations

Access was restricted to only two A100 GPUs, which imposed limitations on the computational capacity and parallel processing abilities, impacting the efficiency and speed of the model training phase.

Extended Training Times

The BERT sequence classification and GPT-2 classification models required approximately 2 hours per epoch for training. This extended duration for training epochs presented a challenge in terms of time management and resource allocation.

Capped Accuracy

A maximum accuracy of 80% was achieved, which was influenced by the constraints of the model versions used. The limited features and capabilities of the specific versions may have prevented the attainment of higher accuracy levels.

Limited Access to Advanced Models

Enhanced results could potentially be achieved with public access to more advanced models such as GPT-3[8] and other advanced embedding generation APIs. However, such access was not available, which may have limited the overall performance and outcomes of the project.

5 Conclusion

This project embarked on the ambitious task of benchmarking distributed machine learning systems using Large Language Models (LLMs) within the specific context of text classification, distinguishing between human-generated and AI-generated texts. The primary goal was to evaluate and optimize the distributed training processes that are pivotal in advancing the capabilities and efficiency of LLMs.

Throughout the course of this study, several key findings were elucidated:

- The application of distributed training techniques, such as data, model, and pipeline parallelism, has demonstrated significant improvements in training efficiency and model scalability.
- Each parallelism strategy presented distinct benefits and challenges, underscoring the importance of aligning model architecture and training methodology with the specific goals and constraints of the task at hand.
- While resource limitations and extended training times were notable challenges, they also provided critical insights into areas for improvement and optimization in future research.

Our experiments yielded promising results, achieving up to an 80% classification accuracy rate and unveiling potential reductions in training times and computational resource usage. These outcomes not only highlight the feasibility of employing advanced distributed training strategies in resource-constrained environments but also open up new avenues for future exploration and development.

5.1 Future Work

Building on the foundation laid by this study, the following avenues present promising directions for future work:

- **Advanced Model Integration:** To further improve embedding quality and classification performance, we plan to incorporate GPT-3, GPT-3.5, T5[9], and other state-of-the-art transformer models. These advanced models are expected to offer enhanced embedding generation capabilities and finer classification nuances.
- **Distributed Training Expansion:** A transition from single-node to multi-node execution will be investigated to significantly improve computational efficiency. Multi-node architectures are anticipated to accelerate training processes and enable the handling of larger datasets and model architectures.
- **Dynamic Learning Rate Adjustments:** We aim to explore and implement adaptive learning rate algorithms that dynamically adjust the learning rate during training. This approach is hypothesized to optimize the training process, leading to faster convergence and improved model performance.
- **Expansion of LLMs:** Experiments will be extended to include the latest and more complex LLMs, with the goal of broadening the research scope and enhancing the overall capabilities of our machine learning systems. This expansion is expected to yield insights into the scalability and performance limits of current LLM architectures.

These forward-looking objectives underscore our commitment to advancing the field of NLP by harnessing the latest developments in LLMs and distributed machine learning technologies. We anticipate that these efforts will contribute significantly to the scalability and accessibility of sophisticated NLP applications.

In conclusion, the project's findings contribute valuable knowledge to the field of distributed machine learning, with specific applications in training Large Language Models. By pushing the boundaries of what is possible in distributed training, we pave the way for more sophisticated, efficient, and accessible NLP technologies in the future.

6 Literature Survey

The investigation into distributed machine learning, the capabilities of Large Language Models (LLMs) in text classification, and methodologies for enhancing machine learning efficiency is built upon a comprehensive literature survey. This survey acknowledges both the foundational theories and the latest innovations that underpin and catalyze our project's objectives.

6.1 Advancements in Distributed Machine Learning

Our project draws from seminal works in distributed machine learning, especially those that optimize the training of LLMs through cutting-edge parallelism techniques. Notable contributions include Shoeybi et al. [10], who introduced Megatron-LM[10] to train large-scale language models with model parallelism on GPU clusters efficiently. Furthermore, Rajbhandari et al. [11] proposed ZeRO, a novel memory optimization technique enabling the training of models with over a trillion parameters, which significantly influences our approach to distributed training.

6.2 Emerging Trends in LLMs for Text Classification

In text classification, our project particularly emphasizes the ability to differentiate human from AI-generated content. The work by Brown et al. [8], showcasing GPT-3's ability for few-shot learning, remains a pivotal reference. It informs our understanding of LLMs' potential to classify text with scant training data. Meanwhile, Liu et al. [12] have made strides with RoBERTa, which refines BERT's pretraining methods, enhancing performance on classification tasks—a technique that our project leverages.

6.3 Efficiency and Environmental Impact of Machine Learning

The research community has made concerted efforts to improve the efficiency of ML models. Dynamic learning rate adjustments are explored to optimize training processes effectively, as these can significantly affect convergence and model performance. Notably, Clark et al. [13] introduced ELECTRA, an approach that trains text models with greater sample efficiency, which has been an inspiration for our project's focus on efficiency.

6.4 Future Directions in LLMs and Distributed Training

The continuation of our literature survey encompasses recent breakthroughs and future-forward studies in LLMs and distributed training. Recent papers expounding upon the use of newer models like GPT-3.5 and T5 for embedding generation and classification tasks indicate a trend towards ever more powerful and nuanced language understanding capabilities. Additionally, studies investigating the shift from single-node to multi-node execution models provide a roadmap for scaling computational efficiency in distributed environments. The exploration of LLM extensions, including novel and more intricate models, forecasts an era of accelerated innovation and broadened research horizons in NLP.

These contemporary studies and their findings not only validate our research direction but also serve as a beacon for our future work, heralding advancements in both the theoretical framework and practical applications of distributed machine learning and LLMs.

References

[1] Zachary Grinberg. Human vs. llm text corpus. `https://www.kaggle.com/datasets/starblasters8/human-vs-llm-text-corpus/data`, 2023.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, pages 4171–4186, 2019.

[3] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[4] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[5] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

[6] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506, 2020.

[7] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[8] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[9] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[10] Mohammad Shoeybi, Mostofa Ali Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. In *arXiv preprint arXiv:1909.08053*, 2019.

[11] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[12] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[13] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.