https://doi.org/10.54809/ga11a.2025.003

Distributed Machine Learning Algorithms: Related Work

Nashma Taha Muhammed[1], Sarkar Hasan Ahmed[2]*

[1]Department of Information Technology, Technical College of Informatics, Sulaymaniyah, Iraq
[2] Department of Computer Networks, Technical College of Informatics, Sulaymaniyah, Iraq

Abstract — **Artificial intelligence has expanded significantly over the last decade due to growing user demand and has achieved major advancements in managing complex tasks. Processing and analyzing this large volume of data is time-consuming and requires substantial computational resources. To address these limitations, distributed machine learning (DML) has emerged as an effective solution, enabling parallelization of tasks by distributing data, models, or both across multiple servers. This review paper thoroughly examines various strategies and methodologies used in DML, with a particular emphasis on data parallelism and model parallelism. These methods significantly enhance scalability and computational efficiency, which in turn accelerate AI advancements in sectors such as autonomous driving, healthcare, and recommendation systems. Additionally, this paper provides an extensive overview of key DML algorithms and frameworks, exploring their advantages, practical applications, and limitations. Furthermore, it identifies and examines important challenges such as security concerns and communication overhead and offers recommendations for future research to develop DML systems that are more reliable, scalable, and efficient.**

Keywords: **Distributed Machine Learning, Deep Learning, Model Parallelism, Data Parallelism**.

Galla-The Scientific Journal of KISSR Vol. I, No. 1 (2026), Article ID: Galla.12181. 10 pages

DOI: 10.54809/ga11a.2025.003. Received: 12 July, 2025; Accepted: 07 September, 2025 Regular research paper; Published: 25 January, 2026.
*Corresponding author's e-mail: nashma.taha.m@spu.edu.iq.

I. INTRODUCTION

The rapid advancement of digital technologies, particularly the Internet of Things (IoT) and enhanced internet infrastructures, has resulted in exceptional growth in both data volume and complexity (Le et al., 2022a). As a result, methods for artificial intelligence (AI) and machine learning (ML) techniques have become essential tools for gaining knowledge and assisting in decision-making in various fields. Traditional machine learning techniques primarily depend on single-machine architectures, which experience significant drawbacks such as computational bottlenecks, insufficient scalability, and insufficient resources for effectively processing large datasets.

A large number of machine learning (ML) algorithms are being used to classify information and create decision-making systems when the complexity of the issue makes an algorithmic solution difficult. Due to this, the amount of training data needed for complex applications may rapidly reach terabytes, and solution designers are often required to adopt distributed systems because of the lengthy duration of training the models. This increases parallelization and overall, I/O bandwidth. To make these kinds of datasets available for use as training data in machine learning tasks, it is necessary to choose and develop algorithms that facilitate parallel processing, data distribution, and the ability to resist errors (Verbraeken et al., 2020).

Training machine learning models usually requires a lot of time and intensive resources, especially in the case of having an enormous dataset with many features. The basic difficulty encountered by an ML cluster operator is how to optimize the scheduling of submitted training tasks to maximize server resources and speed up training completion (Bao et al., 2018). Distributed training is crucial because it enables researchers and data analysts to deal with large datasets that would be too much for a single machine to handle without optimizing the features.

Despite previous studies having precisely explored different DML techniques, they often lack extensive analyses between different parallelization strategies and generally fail to thoroughly address issues related to security vulnerabilities and communication overhead. This review paper covers these gaps

https://doi.org/10.54809/ga11a.2025.003

by thoroughly reviewing the key DML strategies, particularly data parallelism and model parallelism, providing comprehensive details about their strengths and drawbacks.

The primary objectives of this review are to present a comprehensive overview of crucial DML algorithms and frameworks, analyze their practical uses and essential limitations, as well as examine current challenges, including communication and security issues. Additionally, this paper highlights future research directions, recommending the development of more accurate, scalable, and secure DML systems. Through this comprehensive study, it provides important guidance for researchers who are aiming to utilize distributed machine learning to handle complex data challenges across sectors such as recommendation systems, autonomous driving, and healthcare.

The review article is organized into several sections. A brief introduction to the survey is given in the first section. The second section discusses the literature review. The next section discusses the background theory of Distributed Machine Learning. The paper concludes with a summary of findings and recommendations.

II . RELATED WORK

Du et al., 2021, the researchers provided DeCNN, a more efficient inference method that maximizes model parallelism for distributed inference on consumer devices by using a decoupled CNN structure. DeCNN is a new and innovative system that comprises three different schemes. Scheme 1 optimizes at the structural level. It decouples the basic CNN structure for model parallelism by making use of group convolution and channel shuffling. Partition-level optimization is done in Scheme 2. The convolutional layers are divided using a channel group technique, and the fully connected layers are divided using an inputbased approach, which further reveals a high degree of parallelism. Scheme 3 optimizes at the communication level. To improve efficiency and robustness, particularly in cases of poor network connections, it employs inter-sample parallelism to hide communications. Using an ImageNet classification job, they assess how well DeCNN performs on a distributed multiARM architecture. Specifically, their results show that DeCNN uses a lower memory footprint by 65.3% and expedites the inference of large-scale ResNet-50 by 3.21× when employing 1–4 devices, while improving accuracy by 1.29%.

Tang and Stefanov, 2021, authors offered a unique partitioning strategy called the Vertical Partitioning approach, along with a novel methodology, in order to effectively use their partitioning approach for CNN model inference on a distributed system at the edge. This study presents a comparison between their experimental findings on the YOLOv2 CNN model and the results produced by three current approaches. It also highlights the benefits of each methodology in terms of overall system performance and the amount of memory required per edge device. Additionally, their experimental findings on various typical CNN models demonstrate how their unique technique, which makes use of their partitioning strategy, can enable CNN inference while simultaneously improving overall system performance and using a very small amount of memory per edge device. An important factor to consider is that if a CNN partition contains just one CNN layer with a high memory requirement, the overall system performance and memory savings per device will be heavily influenced by this specific bottleneck layer. This is because the partitioning strategy in this paper does not divide individual CNN layers. However, as the number of layers in a CNN model increases, the memory saving rate per device and the overall system performance speedup have the potential to continue improving. This is especially true in a distributed system with a larger number of edge devices, as their methodology allows for the CNN model to be divided into multiple partitions, ensuring that no single layer becomes a bottleneck in the system. Based on the experimental results for YOLOv2, this paper concludes that their partitioning strategy and methodology can successfully enable CNN inference on a fully distributed system at the edge. This approach requires less memory per edge device and/ or offers higher overall system performance compared to other existing partitioning strategies.

Wang, Tong, and Zhi, 2023, introduced a new approach to model parallelism, which involves separating the CNN structure using group convolution and implementing a unique channel shuffling mechanism. The method in this paper can reduce each device's memory footprint while eliminating inter-device synchronization. The authors created a parallel FPGA accelerator for the well-known CNN model ShuffleNet by using the suggested model parallelism technique. Further optimization was done on this accelerator to fully utilize the hardwarelevel parallelism of the FPGA, including features like kernel vectorization and aggregate reads. The research used ShuffleNet to conduct tests on two FPGA boards, each equipped with an Intel Arria 10 GX1150 and 16GB DDR3 memory. According to the testing findings, ShuffleNet demonstrated a 1.42-fold improvement in speed and a 34% reduction in memory use while using two devices, as compared to its non-parallel version. This study presents the empirical impact of CNN model parallelism while using two FPGAs. This approach may be extended to include other devices and perform operations on multiple devices simultaneously. Nevertheless, having numerous devices generally produces better outcomes. An increase in the number of devices was able to decrease the network layer calculation time in a roughly linear proportion to the speed-up ratio, but it was not possible to reduce the time

https://doi.org/10.54809/ga11a.2025.003

required for device synchronization. As a result, the device count will become more important in determining the speed-up ratio rather than the network computation time. This article used two FPGAs to illustrate the experimental impact of CNN model parallelism. Naturally, additional devices could be added and controlled using this approach. However, increasing the number of devices will cause the channel to exchange excessive amounts of data, which might have the opposite effect. Using these strategies, the experimental results demonstrated that the parallel ShuffleNet FPGA accelerator technique used in this work achieved a high level of model parallelism while maintaining accuracy. It produced a 1.42× increase in performance, a power usage of around 20 W, and a 34% decrease in memory footprint when two FPGAs were used. Additionally, it is expected that the experimental results will be most suitable when there are three to four devices.

Sun, 2018, focused on separate concepts: first, it aims to enhance the cluster's shared resource usage for many distributed MLGP workloads. Employing a cluster management system (CMS) to operate many distributed MLGP applications in a single cluster is becoming popular among businesses. Poor cluster usage results from existing CMSs' inability to assign more than a static partition of the cluster to each application. This work proposes a new content management system (CMS) called Dorm to address this issue. It partitions a cluster using virtualization techniques, runs a single application per partition, and can dynamically resize each partition at runtime to meet various performance constraints and achieve high cluster utilization. Comprehensive performance analyses have shown that Dorm has the potential to boost cluster utilization by as much as 2.32×. Secondly, this study enhances DFSs' metadata lookup performance. Distributed hash tables (DHT) are often used by existing DFSs to maintain their metadata servers. Clients must find the required metadata item using a lookup service before executing a metadata operation. High delay and decreased metadata operation performance might result from the lookup process. To solve this issue, this work creates MetaFlow, a brand-new metadata lookup service. By mapping the actual network topology to a logical B-tree, MetaFlow creates suitable flow tables for SDN-enabled switches and uses software-defined networking (SDN) methods to move metadata searches to the network layer. Comprehensive performance analyses have shown that, compared to DHTbased methods, MetaFlow might boost system throughput by a ratio of up to 6.5 and decrease system latency for metadata management by a factor of up to 5. Third, by using the Parameter Server (PS) architecture, this study reduces the communication cost associated with distributed machine learning (ML). The PS architecture consists of a set of server nodes that store globally shared parameters and a set of worker nodes that execute dataparallel computing. There would be a lot of communication overhead since each worker node would continuously gather parameters from server nodes and submit changes to them. To ackle this issue, ParameterFlow, a communication layer for the PS framework with a dynamic value-bounded filter (DVF) and an update-centered communication (UCC) architecture, was devised in this study. To facilitate data flow between worker nodes and server nodes, UCC presents a broadcast/push architecture. By selectively deleting updates for network transmission, DVF can directly cut network traffic and communication time. According to experiments, PF may accelerate widely used distributed machine learning applications by up to 4.3× when compared to the traditional PS framework. Finally, this work allows large-scale graph processing with excellent performance on small clusters with limited memory.

Le et al., 2022b, provided an optimization strategy for deep convolutional neural networks (FPDCNNs). Initially, redundant parameters are trimmed using a pruning technique based on Taylor's loss (FMPTL), which not only compresses the structure of the DCNN but also lowers the computational cost of training. The next technique described is a glowworm swarm optimization method based on an information-sharing strategy (IFAS), which enhances parameter optimization capability by modifying weight initialization. Ultimately, an equitable distribution of data is achieved, and the cluster's parallel performance is enhanced by the use of a dynamic load-balancing approach based on parallel computing entropy (DLBPCE). Experiments presented in this study demonstrate that this technique achieves both a faster processing speed and a lower computational cost for network training when compared to previous parallelized algorithms. An empirical comparison was undertaken to evaluate the performance of FP-DCNN, MRCNN, BS-CNN, and NFP-DCNN (a variation of FP-DCNN without model compression). The evaluation was based on four datasets: CIFAR-10, Fashion-MNIST, PatchCamelyon, and EMNIST-Bymerge. When processing the CIFAR-10 dataset, the FP-DCNN algorithm takes 52.72% of the time to execute compared to BS-CNN, 35.98% compared to MR-CNN, and 40.23% compared to NFP-DCNN. While the other comparison techniques increase geometrically, the FP-DCNN running time increases only slightly as the amount of data increases. In particular, the percentage of FPDCNN running time is 45.14% of BS-CNN, 34.94% of MR-CNN, and 31.85% of NFPDCNN when working with larger datasets like EMNIST-Bymerge. The primary cause of the reduction is that the FP-DCNN method pretrains the network before classification, which lowers the algorithm's total execution time while simultaneously reducing the computing cost of training. The speed-up ratio, which is defined as follows: Speed-up = Ts/Tp, is often used as an indicator to assess the algorithm's parallel performance, where

https://doi.org/10.54809/ga11a.2025.003

Tp and Ts stand for the algorithm's execution times in parallel and serial cases, respectively.

III. CRITICAL ANALYSIS

Although each reviewed paper highlights significant advancements in distributed machine learning, particularly in CNN model parallelism, their approaches differ substantially in architecture, performance trade-offs, and real-world applicability.

Through all of the works, performance is especially used to evaluate based on metrics like throughput, training acceleration, power efficiency, inference latency, and memory savings.

Du et al. 2021 achieved that the ARM-based multi-device was 3.21x faster and used less memory by about 65.3%. However, it was not tested on GPUs or other types of systems. Tang and Stefanov (2021a) indicated that they achieved 22–35% less memory use and gained a throughput of about 18%, yet devices with uneven loads caused up to 25% of idle time. Wang, Tong, and Zhi (2023) showed that using FPGA clusters could parallel accelerate tasks by 2.6 to 4.1x while using about 40% less power. However, once using more than four devices, synchronization overhead increased by about ~20% per FPGA. Le et al. (2022b) found that adaptive load balancing increased training speed by 1.8 to 2.3×, but in some edge-network scenarios, static scheduling caused latency of >100 ms.

These differences show that, despite the opportunity of significantly increasing efficiency, challenges with workload balancing, scalability, and hardware portability still exist. Tables 1 and 2 highlighting their pros and cons and detailing practical challenges with measurements.

Table 3 provides a methodological categorization of the reviewed algorithms, organized by use case, Throughput, Platform, and algorithm. Showing the difference among the selected works.

IV. FUNDAMENTALS OF DML ALGORITHMS

The characteristics of data and the performance of algorithms affect the machine learning solution. The current problem is that learning algorithms are incapable of utilizing all data for the specific purpose of learning in a suitable period. Developing a successful ML model is often difficult and time-consuming, requiring the selection of a suitable algorithm and the development of an optimal model architecture. Over large volumes of data, a single machine's computational capabilities are not sufficient to train ML models. Using distributed machine learning to execute algorithms on clusters, data centers, and cloud providers is one way to address this challenge (Dehghani and Yazdanparast, 2023).

ML creates models from training datasets to predict new data and utilize them. ML models commonly have several parameters. To reduce prediction error, a machine learning application usually employs an iterative convergence algorithm, such as stochastic gradient descent (SGD), to train specific models. Based on the parameter server, different distributed ML systems have been proposed to handle several training datasets, including MxNet, Project Adam, Petuum, TensorFlow, and SINGA (Sun, 2018).

Table 1: Comparative Analysis of Reviewed Approaches.

Pros	Cons	Reference
Memory efficiency Improve accuracy	Limited to multiARM devices	Du et al. 2021
Vertical partitioning Memory and performance balance	Bottlenecks when device is increase not partition within layers	Tang and Stefanov 2021a
Using FPGAs makes speed Memory savings Low power	Overhead grows with device count	Wang, Tong and Zhi 2023
Improvements in throughput and latency	Lacks model-level parallelism	Sun (2018)
Faster training Adaptive load balancing	Use Static MapReduce framework make a limit use in real time	Le et al. (2022b)

Table 2: Research Challenges in Distributed Machine Learning Approaches.

Challenges	Measurement	Reference
Portability to a various hardware architecture	Tested only on ARM CPUs not tested on GPU.	Du et al. 2021
Improve management for those model that are not balance	On small layers Idle time up to 25%	Tang and Stefanov 2021a
Scalability without overhead when using more devices	Sync overhead increase ~20% per FPGA with more than 4 device.	Wang, Tong and Zhi 2023
Suitable for edge or real-time deployment	>100 ms latency under edge network.	Le et al. 2022b

https://doi.org/10.54809/ga11a.2025.003

Table 3: Comparative Table.

Use Case	Throughput	Platform	Algorithm	Reference
Image classification (autonomous driving)	+3.21×	Edge (multi-ARM devices)	DeCNN	Du et al. (2021)
Object detection (YOLOv2)	+18%	Edge	Vertical	Tang and Stefanv (2021a)
CNN acceleration	+2.6–4.1×	Edge / FPGA clusters	FPGA Model	Wang, Tong and Zhi 2023
Image classification	Moderate	Cloud / MapReduce	FP-DCNN	Le et al. 2022b

Distributed machine learning involves multiple nodes and systems designed to improve performance, enhance accuracy, and scale to larger input data sizes. For many algorithms, increasing the input data size significantly reduces learning error and is often more efficient than using more complex techniques. These systems are divided into three main categories: general-purpose, purpose-built, and database systems (Galakatos, Crotty and Kraska, 2017).

In these systems, a set of static workers is maintained in case of failure, and new workers are deployed only on failed machines. For managing ML clusters, most use Borg or YARN-like schedulers (Bao et al., 2018). The classification of these algorithms is shown in Figure 1.

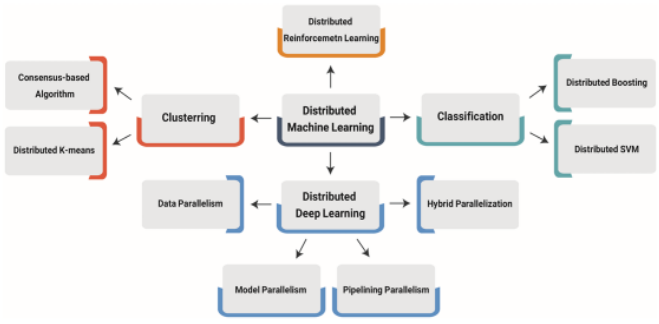

Figure 1: Distributed machine learning algorithms (Dehghani and Yazdanparast 2023).

A. Distributed deep learning

A neural network is a computational model consisting of numerous processing units, known as neurons. These neurons are arranged as interconnected layers, which create the neural network. Within a network, an input parameter is used to activate the input neurons, while the neurons in the subsequent layer are activated by the weight of neurons from the earlier layer (ÓE et al., 2020). Developers can use more than one GPU card for training deep neural networks by using distributed deep learning technology. Various strategies are available for implementing distributed architectures, depending on the abstraction of each node— whether at the GPU or server level—as well as the communication between nodes (Óbudai Egyetem et al., 2020). Different types of deep learning models are commonly used, such as convolutional neural networks, self-coding network models, deep trust network models, and restricted Boltzmann machine models. So, to overcome the challenge of training different method categorized which are model parallelism, data parallelism, pipeline parallelism, and hybrid parallelism (Wang, Fan and Wang, 2021). Table 1 presents a comprehensive overview of these algorithms.

A.1. Model parallelism

Is a distributed training technique that divides model parameters among different computing machines or workers. Each worker is assigned different parameters or layers of the model by the main machine (Haque et al., 2022).

Memory limitations can be effectively addressed through this approach (Bian et al., 2021). When the model is too large and exceeds the capacity of a single machine, it can be divided across several machines. For instance, one layer can fit into the memory of one machine, and the process of forward and backward propagation involves communicating the output from one worker to another sequentially. Model parallelism is used only when the model cannot be handled by a single worker, and it is generally slower to train compared to other approaches (Hegde and Usmani, 2016). As shown in figure 2 model partitioning has two types: vertical partitioning (splitting between neural network layers) and horizontal partitioning (splitting within layers) (Langer et al., 2020).

Vertical partitioning can be implemented on any deep learning model because each layer is unaffected by the partition (Langer et al., 2020). Tang and Stefanov (2021b) identified the main features of their strategy, noting that item level partitioning within a layer is not handled by this method. In their study, they selected a CNN model and partitioned its layers so that each partition included non-consecutive CNN layers. The memory required for each partition can be significantly reduced because

https://doi.org/10.54809/ga11a.2025.003

both the storage needed for data and weights, and the data exchanged between layers, are reduced when dealing with large partitions. Horizontal partitioning divides the layers into several partitions, and separate parts of each sample are processed in parallel using different devices. Each device handles a distinct part of the sample (Langer et al., 2020). In this strategy, the weights of each CNN layer are partitioned, while the input data to each layer is not. Each partition of the CNN model has all layers of the model, but each layer uses only part of its weight because the weights are already divided. Communication and synchronization between different components of a model layer are important because the output data from each part must be combined with outputs from other parts of the same layer. The memory needed for deployment is reduced because the storage required for weights is decreased, especially when working with large partitions.

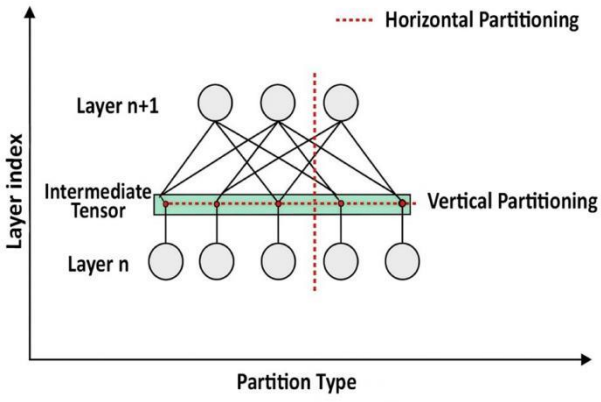

Figure 2: Types of Model Parallelism (Langer et al. 2020).

A.2. Data Parallelism

This strategy distributes the entire dataset among workers, with each worker executing a single replica of the model and communicating with other workers to synchronize their progress at the end of the training process. (Bian et al., 2021). The fundamental idea is to enhance the overall sample rate by duplicating the model on different machines, allowing more information about the loss function to be collected faster, and enabling backpropagation to be carried out in parallel. Data parallelism is conceptually performed as follows: first, every worker downloads the selected model. Next, each worker utilizes its assigned data in parallel to perform backpropagation. Finally, the results are combined and integrated to form a new model (Langer et al., 2020). This technique involves duplicating the model parameters among all workers. Each worker, during a single iteration, performs the local gradient or model updates through sampling several mini batches of data. Then each node exchanges the results with other nodes. After that, to obtain the new global model, aggregation and broadcasting are executed (Haque et al., 2022).

a) Data parallelism can enhance a system's throughput through distributed parallel computing, and datasets that cannot be stored on a single machine can be processed using data parallelism. However, data parallelism also has some challenges, such as the overhead of parameter synchronization, hardware limitations when dealing with large data, and optimization algorithm constraints (Dehghani and Yazdanparast, 2023).

A.3. Pipeline parallelism

Pipeline parallelism partitions training tasks for a model into sequential processing stages. In the context of model parallelism, this means assigning various stages of model training to different machines and transferring intermediate results between machines to reduce training time (Haque et al., 2022).

Pipeline parallel computing divides the layers of a model into various stages, with each stage consisting of a consecutive set of layers. Each stage is assigned to a different GPU, which performs both forward and backward passes for its layers. By combining model and data parallelism, pipeline parallelism enhances the speed of neural network training. This can be done synchronously, as shown in GPipe (Huang et al., 2019), or asynchronously, as in PipeDream (Narayanan et al., 2019).

A.4. Hybrid parallelism

Hybrid parallelism uses data and model parallelism at the same time (Langer et al., 2020). It optimizes performance by taking advantage of data parallelism with low overhead for weak scaling, as well as model parallelism for more compute resources in strong scaling. Hybrid parallelism may resolve memory issues caused by data parallelism, as well as communication and scaling limitations related to model parallelism (Kahira et al., 2021).

Different research works have shown that hybrid approaches can outperform current parallel methods in training time and scalability while maintaining similar accuracy. Table four shows a comparative overview of research papers which

https://doi.org/10.54809/ga11a.2025.003

organized by the algorithm that they used, year of publication, datasets used, and evaluation metrics.

B. Comparison with traditional machine learning.

Researchers are always trying to develop new techniques to reduce processing times because of the increasing amount of data and the need for good and fast processing. Deep learning, primarily using CNN, automates extracting features as well as learning complicated hierarchical patterns based on new data such as images. Traditional machine learning operates only on one machine, which causes several bottlenecks when dealing with a large scale of data or a complex model. However, in distributed machine learning several machines participate in parallelizing the training process, reducing the time that is required to train models on large datasets. This method includes different strategies such as data parallelism, which includes distributing datasets among workers and each of them training a local model, and model parallelism which involves distributing a model to the different parts that are training on different machines. Distributed machine learning allows more efficient resource utilization, increases fault tolerance as well, and includes privacy-preserving techniques such as federated learning. These services make it very suitable for modern applications that need to process large volumes of data efficiently.

V. ARCHITECTURE AND FRAMEWORK

The architecture of distributed machine learning affects performance as well as ease of use. Different frameworks are available depending on the architecture. These frameworks are designed to ensure resource usage and handle the complexity of distributed computing, data management, as well as model training.

PyTorch provides various tools to facilitate distributed training, such as DataParallel and DistributedDataParallel. The first enables single-process multi-thread data parallel training, which means utilizing multiple GPUs on the same node. The second enables multiprocess data parallel across GPUs and nodes, also including Remote Procedure Calls for general distributed training such as parameter server (Li et al. 2020).

The aim of SlipStream is to provide efficient distributed training even if a fault occurs. It does not need a spare server and does not affect the accuracy of the model in contrast to faultfree training. This framework can tolerate several hardware failures and guarantee that training throughput remains proportional to the number of operational servers. SlipStream is optimized for fast recovery from faults because failures do not require extensive re-shuffling of model parameters across functional nodes. Profilers, Executors, and critical Planners are the key components of SlipStream. When a large training task is submitted, SlipStream performs a short profiling job to gather key performance statistics, including the average latency of micro-batches for both the forward and backward passes, memory needs for activations and gradients, and the bandwidth for inter-node communication. The profiling job performs a small number of training iterations, typically 100 by default, and normally takes a few minutes to complete. The planner utilizes these statistics. Runtime Executors use the plans, which are saved in distributed fault-tolerant storage. SlipStream and Executor operate together on every GPU node to handle training plans specified for that specific node (Gandhi et al. 2024).

Horovod is an open-source distributed training framework for different deep learning frameworks that addresses various problems. By applying custom reduction to resolve interworker communication, it requires only a few additional lines of code from users and enables the distribution of computation across multiple CPUs via MPI. Paper (Alonso-Monsalve et al. 2021) identified that Horovod is better than distributed TensorFlow in terms of the number of images processed per second.

The problems solved by Horovod which identified in this paper include:

1.The developer must have many GPUs to execute their deep learning algorithms in a distributed way because the distribution of training computation concentrates on the GPUs rather than the CPU.

2.The user is usually required to make significant changes to their source code to distribute computation based on the training API.

3.Distributed models often fail to fully take advantage of available hardware resources because of inter-GPU communication, which in terms of execution time may result in significant overhead.

VI. CHALLENGE AND LIMITATION

Unlike machine learning on a single device, distributed machine learning faces several challenges. This section talks about the two most common problems, which are communication overhead and security with unreliable machines:

1. Communication overhead is one of the System's performance bottlenecks. When the number of workers is

https://doi.org/10.54809/ga11a.2025.003

increased, the overall communication overhead is also increased in a distributed system, regardless of the synchronization method. This leads to a considerable burden on the network and makes it difficult to achieve optimal capacity in a distributed system. Additionally, when the machines are located in different locations, communication is slowed by - the increasing number of machines, so the total time used by computation will be reduced, hence reducing the training time. Therefore, it is crucial to minimize communication overhead to achieve better scalability and speed up training.

Table 4: Distributed Deep Learning Technique

Algorithm	Year	Dataset	Evaluation metrics	Reference
Model parallelism	2023	MNIST, CIFAR-10	Speed Increase Memory Footprint Reduction Accuracy Resource Utilization Power Consumption Performance Improvement	(Wang et al. 2023)
	2021	ImageNet	Memory Footprint Reduction Accuracy Improvement	(Du et al. 2021)
	2023	GPT-3 like language model, U-Net Transformer	Throughput Metric Speedup	(Zhuang et al. 2023)
	2021	Not Mentioned	Speedup Execution Time Average Step Time Weak Scaling Performance Strong Scaling Performance Memory and Communication Cost	(Bian et al. 2021)
Data parallelism	2020	Not Mentioned	Latency Scalability Communication Overhead	(Li et al. 2020)
	2018	Not Mentioned	Training Speed	(Sergeev and Del Balso 2018)
Pipeline Parallelism	2019	ImageNet-2012, Multilingual Corpus	Accuracy Performance	(Huang et al. 2019)
	2020	U-Net Memory Benchmark, AmoebaNet-D Spee Benchmark	Throughput Memory Usage	(Kim et al. 2020)
	2021	Transformer-based Language Models (GPT-2)	Training Throughput Memory Consumption	(Shigang Li and Torsten Hoefler 2021)
Hybrid Parallelism	2021	CosmoFlow, 3D U-Net	Weak and Strong Scaling Memory Usage	(Oyama et al. 2021)
	2021	2D and 3D Datasets	Prediction Accuracy Accuracy of Analytical Model Performance and Scalability Memory/ Computational pressure	(Kahira et al. 2021)

2. Security is one of the most challenging problems in a distributed system. In some cases, it is complex to identify identities or workers' behavior, particularly in federated learning. It is also possible for some workers to be attacked and injected with poisoned data, or for the message during transmission to be manipulated. In the worst situations, some machines may behave arbitrarily or change their data. Apart from that, it is also a common problem for workers to have software or hardware failures, for example, bit-flipping in the memory or communication media. In this case, it is necessary to assume that the machines are not reliable and to defend the system against possible attacks and failures.

CONCLUSION

Distributed machine learning (DML) is a crucial technique for addressing the complexity, performance demands, and scale requirements of modern AI applications. By using techniques such as data, model, pipeline, and hybrid parallelism, DML enables faster training, handles large datasets, and enhances resource utilization. This review paper has analyzed different DML approaches, algorithms, and frameworks, focusing on their practical applications, limitations, and strengths. While some challenges remain especially communication overhead and security vulnerabilities these cause significant limitations to further advancements. Performance trade-offs often depend on the context, and improvements in one metric could compromise another. Researchers should focus on enhancing the efficiency of inter-node communication, designing and developing security protocols to protect against vulnerabilities, and increasing fault tolerance. The combination of edge–cloud collaborative systems and federated learning offers privacy protection. Still, a number of problems must be addressed, including deployment across diverse edge devices, latency requirements, and ethical issues. By contributing to continued innovation in these areas, distributed machine learning can become more robust, scalable, and reliable. Addressing these limitations provides helpful directions for future research, which can lead to more secure and efficient DML systems in fields such as healthcare, large-scale scientific computing, and autonomous systems.

REFERENCE

Alonso-Monsalve, S. *et al.* (2021) 'Analyzing the distributed training of deep-learning models via data locality', *2021 29th Euromicro International Conference on Parallel, Distributed and Network-BasedProcessing(PDP)*,pp.117–121.doi:10.1109 /pdp52278. 2021. 00026.

Arora, A. and Basu, N. (2023) 'Machine learning in modern healthcare', International Journal of Advanced Medical Sciences and Technology,3(4),pp.12–18. doi.org/10.54105/ijamst.D3037.063423

Bao, Y. et al. (2018) 'Online job scheduling in distributed machine learning clusters', in Proceedings of IEEE INFOCOM 2018 – IEEE Conference on Computer Communications [Preprint]. https://doi.org/10.1109/INFOCOM.2018.8486422

Bian, Z. et al. (2021) 'Maximizing parallelism in distributed training for huge neural networks', arXiv preprint [Preprint]. https://doi.org/10.48550/arXiv.2105.14450

Dehghani, M. and Yazdanparast, Z. (2023) 'From distributed machine to distributed deep learning: a comprehensive survey', Journal of Big Data, 10(1). https://doi.org/10.1186/s40537-023-00829-x

Du, J. et al. (2021) 'Model parallelism optimization for distributed inference via decoupled CNN structure', IEEE Transactions on Parallel and Distributed Systems, 32(7), pp. 1665–1676. https://doi.org/10.1109/TPDS.2020.3041474

Galakatos, A., Crotty, A. and Kraska, T. (2017) 'Distributed machine learning', in Encyclopedia of Database Systems. New York: Springer, pp. 1–6. https://doi.org/10.1007/978-1-4899-7993-3_80647-1

Gandhi, S. et al. (2024) 'SlipStream: Adapting pipelines for distributed training of large DNNs amid failures', arXiv preprint [Preprint]. Available at: http://arxiv.org/abs/2405.14009

Haque, S. et al. (2022) 'Communication-efficient data parallel distributed deep learning: a comprehensive survey', in IEEE International Conference on Program Comprehension. IEEE Computer Society, pp. 36–47.

Hegde, V. and Usmani, S. (2016) 'Parallel and distributed deep learning', Stanford University Technical Report, pp. 1–8.

Huang, Y. et al. (2019) 'GPipe: Efficient training of giant neural networks using pipeline parallelism', in Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Vancouver, Canada.

Kahira, A.N. et al. (2021) 'An oracle for guiding large-scale model/hybrid parallel training of convolutional neural networks', in Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing (HPDC 2021). ACM, pp. 161–173. https://doi.org/10.1145/3431379.3460644

Kim, C. et al. (2020) 'torchgpipe: On-the-fly pipeline parallelism for training giant models', arXiv preprint [Preprint]. Available at: http://arxiv.org/abs/2004.09910

Langer, M. et al. (2020) 'Distributed training of deep learning models: a taxonomic perspective', IEEE Transactions on Parallel and Distributed Systems, 31(12), pp. 2802–2818. https://doi.org/10.1109/TPDS.2020.3003307

Le, Y. et al. (2022) 'FP-DCNN: a parallel optimization algorithm for deep convolutional neural networks', Journal of Supercomputing, 78(3), pp. 3791–3813. https://doi.org/10.1007/s11227-021-04012-y

Li, S. et al. (2020) 'PyTorch distributed: experiences on accelerating data parallel training', arXiv preprint [Preprint]. Available at: http://arxiv.org/abs/2006.15704

Maboud, Y.E. et al. (2024) 'Accelerating recommender model training by dynamically skipping stale embeddings', arXiv preprint [Preprint]. Available at: http://arxiv.org/abs/2404.04270

Narayanan, D. et al. (2019) 'PipeDream: generalized pipeline parallelism for DNN training', in Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP 2019). ACM, pp. 1–15. https://doi.org/10.1145/3341301.3359646

Óbudai Egyetem, IEEE Hungary Section and IEEE Industrial Electronics Society (2020) 'Parallel and distributed training of deep neural networks: a brief overview', in Proceedings of the IEEE 24th International Conference on Intelligent Engineering Systems (INES 2020). Reykjavík, Iceland, p. 231.

Oyama, Y. et al. (2021) 'The case for strong scaling in deep learning: training large 3D CNNs with hybrid parallelism', IEEE Transactions on Parallel and Distributed Systems, 32(7), pp. 1641–1652. https://doi.org/10.1109/TPDS.2020.3047974

Sergeev, A. and Del Balso, M. (2018) 'Horovod: fast and easy distributed deep learning in TensorFlow', arXiv preprint [Preprint]. https://doi.org/10.48550/arXiv.1802.05799

Li, S. and Hoefler, T. (2021) 'Chimera: efficiently training large-scale neural networks with bidirectional pipelines', arXiv preprint [Preprint]. Available at: https://arxiv.org/abs/2107.06925

Sun, P. (2018) Performance optimization for distributed machine learning and graph processing at scale over virtualized infrastructure. PhD thesis. Nanyang Technological University. https://doi.org/10.32657/10356/73229

Tang, E. and Stefanov, T. (2021) 'Low-memory and high-performance CNN inference on distributed systems at the edge', in ACM International Conference Proceeding Series. ACM. https://doi.org/10.1145/3492323.3495629

Verbraeken, J. et al. (2020) 'A Survey on Distributed Machine Learning', ACM Computing Surveys. Association for Computing Machinery. https://doi.org// 10.1145/3377454.

Wang, J., Tong, W. and Zhi, X. (2023) 'Model Parallelism Optimization for CNN FPGA Accelerator', Algorithms The 2023 MDPI, 16(2). https://doi.org 10.3390/a16020110.

Wang, Pin, Fan, E. and Wang, Peng (2021) 'Comparative analysis of image classification algorithms based on traditional machine learning and deep learning', Pattern Recognition Letters, 141, pp. 61–67. https://doi. org/10.1016/j.patrec.2020.07.042.

Zhuang, Y. et al. (2023) 'On Optimizing The Communication of Model Parallelism', Proceedings of Machine Learning and Systems, 5, pp. 526–540.