Simplifying Usability of Simulation Frameworks using a UI for ASTRA-sim

T2000

for the

Bachelor of Science

from the Course of Studies Computer Science

at the Cooperative State University Baden-Württemberg Stuttgart

by

Anouk de Brouwer

1111

Due Date	1111- 1111
Student ID, Course	8878508, TINF23A
Company	Hewlett Packard Enterprise, Böblingen
Supervisor in the Company	Susanne Helmer
Reviewer	Ph.D Lianjie Cao

Confidentiality Statement

The T2000 on hand

Simplifying Usability of Simulation Frameworks using a UI for ASTRA-sim

contains internal resp. confidential data of Hewlett Packard Enterprise. It is intended solely for inspection by the assigned examiner, the head of the Computer Science department and, if necessary, the Audit Committee at the Cooperative State University Baden-Württemberg Stuttgart. It is strictly forbidden

- to distribute the content of this paper (including data, figures, tables, charts etc.) as a whole or in extracts,
- to make copies or transcripts of this paper or of parts of it,
- to display this paper or make it available in digital, electronic or virtual form.

Exceptional cases may be considered through permission granted in written form by the author and Hewlett Packard Enterprise.

Stuttgart, 1111

Anouk de Brouwer

Author’s declaration

Hereby I solemnly declare:

1. that this T2000, titled *Simplifying Usability of Simulation Frameworks using a UI for ASTRA-sim* is entirely the product of my own scholarly work, unless otherwise indicated in the text or references, or acknowledged below;
2. I have indicated the thoughts adopted directly or indirectly from other sources at the appropriate places within the document;
3. this T2000 has not been submitted either in whole or part, for a degree at this or any other university or institution;
4. I have not published this T2000 in the past;
5. the printed version is equivalent to the submitted electronic one.

I am aware that a dishonest declaration will entail legal consequences.

Stuttgart, 1111

Anouk de Brouwer

Abstract

this is not the abstract, write one instead

Contents

Acronyms

AI	Artificial Intelligence
ASTRA-Sim	Accelerator Scaling for TRAining Simulator
DML	Distributed Machine Learning
GPU	Graphics Processing Unit
HPE	Hewlett Packard Enterprise
LLM	Large Language Model
ML	Machine Learning
NIC	Network Interface Card
NPU	Neural Processing Unit
SGD	Stochastic Gradient Descent
SSP	Stale Synchronous Parallel
UI	User Interface
UX	User Experience

List of Figures

List of Listings

1 Introduction

1.1 Motivation

Machine Learning is used in many aspects of daily life, business and research. To fit demands of high scalability, distribution of the training is used. Distributed Machine Learning (DML) is dependable of configurations such as parallelization strategy, topologies and communication. Important is to find configurations to minimize used training time and optimize computation and communication distribution. Finding such configurations is difficult, as it depends on an expensive hardware base and a huge variety of available system choices. Optimizing choices uses many resources, such as money, time and power. To reduce these costs, simulation can be used. One simulator for research of DML is Accelerator Scaling for TRAining Simulator (ASTRA-Sim). When companies want to train their own Machine Learning (ML) model, they face similar challenges. Here Hewlett Packard Enterprise (HPE) could provide solutions such as the necessary hardware infrastructure, making those companies their customers. Additionally, they should provide associated system choices to use the system efficiently. To gain the companies as their customers, they should provide realistic insights in how long the training would take. To find and help evaluate such information, a simulator like ASTRA-Sim can be used. As it is a research tool and not designed for a sales use case like this, its usage is challenging. It's multiple versions and parameters, that need prior training and expertise, make usage for a non-expert user group like HPE customers difficult.

1.2 Problem Statement

1.3 Objectives

1.4 Structure

2 Literature and State of the Art

In this chapter, the current state of the art and literature get presented. First, the development of machine learning to distributed machine learning gets sketched. Then, ASTRA-Sim gets presented and compared to alternative DML (simulation) approaches. Lastly, User Interface (UI) and User Experience (UX) best practices, with a focus on evaluation criteria and UI for scientific tools get presented.

2.1 History of Distributed Machine Learning

DML is a concept based on ML distributed on multiple machines. To deeply understand it, one needs to know basics of ML first.

Development of Machine Learning

The concept of machines learning similar to human learning was first proposed in 1950 by Turing [1]. He proposed the first known of theoretical description of the concept that later became known as Artificial Intelligence (AI). In his paper, he introduces basics such as the idea that machines could simulate human intelligence, if they are given the right data and algorithms. He states that learning, similar to children education is central and supports it with evolutionary algorithms. He claims that machines, which can be seen as discrete-state machines, can simulate anything, which enables them to universal computational capabilities useable for ML.

Following this basic idea, the first working ML program was introduced in 1959 by Samuel [2]. Samuel presents a program that is able to play the game checkers better than its programmer, with only 8 – 10 hours of playing and information such as rules of the game. This program is based on self-improvement, it adjusts its strategy based on previous outcomes. Therefor it is the first documented “self learning” algorithm. The learn process has two approaches; one is memorizing each board positions evaluation and the other one to adjust this evaluation function based on experiences. In this version, it uses the delta between expected and actual result to update expected parameters.

At this time early ML begun and these concepts were expanded, as in 1963 Abramson presents further pattern recognition and machine learning approaches [3]. Here, statistical approaches were introduced and data was first viewed as vectors in a multidimensional space. He framed pattern recognition as a classification problem in vector spaces with two subproblems, being the partitioning techniques and choice of measurements. Also, the first idea of a distinction between supervised and unsupervised training was explained. But there were still gaps in research that only had to be discovered in the following years, like the selection of relevant features.

In the following years, many concepts of ML were revised and newly discovered. Such as backpropagation in the 80s [4], that introduced using gradient calculation for loss correction, or the big data field especially in the 2000s [5].

As ML grew and gained importance, problems regarding growing demands arose. One is the growing availability of datasets. With more data available, training once possible in a few ours might take days or even years to finish, if not optimized. The data is also more complex, forming vectors of increasing dimensions, compared to past data. Also, models are increasing in complexity, so do deep networks have trillions of parameters compared to shallow neural networks in 1990 [6]. These problems demand a solution. Parts of these problems could be solved by scaling with improved hardware as an option, as chips could be made more efficient by scaling with smaller transistors [7]. This approach, based on Moore's law [8], is not upholdable anymore, in the 2010s it was declared "dead" [9]. Naturally, the developed solution was distributing the training. DML is a subset of ML that splits the training process onto multiple Neural Processing Units (NPUs) [10]. In 2024, distributed training became the standard for large scale systems and remained the state of the art [11].

DML Basics

With DML a solution for scaling was introduced, and new challenges have to be solved. Distributing the machine learning process is not straightforward—it depends on many factors, such as the way of splitting, network latencies and communication strategies.

The basic DML process combines individual computation of different workers with a communication between them. That way they can train one model together by separating tasks. This separation can follow two approaches visualized in figure 2.1 by the official PyTorch website (https://docs.pytorch.org/torchrec/high-level-arch.html) [**<empty citation>**]. One, called data parallelism, is splitting the data that the model is supposed to train on. That way, all workers iteratively train a part of the data onto the same model and communicate the models parameters and gradients between computation iterations. While data parallelism has the advantage of being easily implementable and reduce computation times nearly linearly (TODO check if true lol), its communication can create new overheads if workloads are distributed unevenly and a synchronous communication is used. Detailed on that and further advantages and disadvantages get discussed in section **??**. The other parallelization, called model parallelism, a type being for example pipeline parallelism, is focussing on splitting the models parts onto the workers. Here, the layers of the model are divided and trained separately with according communication. This has the advantage of enabling training to include large models, but requires more complicated communication than data parallelism. Common are also hybrid approaches in which both data and models get split. This combines advantages such as a high scalability and the possibility for an efficient training performance, but new introduced challenges are finding the optimal distribution between data and model parallel strategies as well as for load balancing communication and computation efficiently [12].

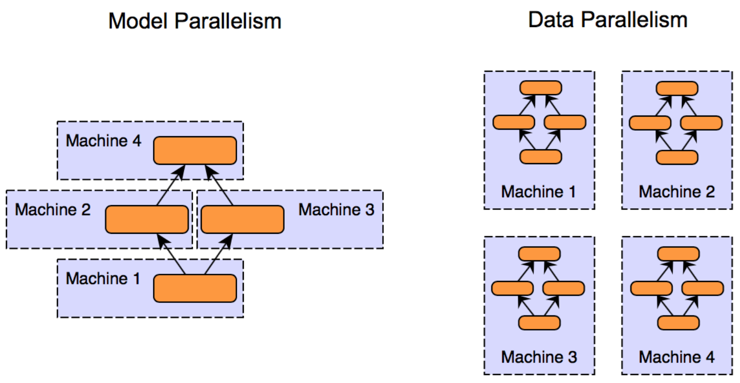

Figure 2.1: Parallelism Strategies

Hardware Details

The workers used in the DML process, generally called NPUs, are usually Graphics Processing Units (GPUs). They can appear in high amounts, from 8 GPUs for research purposes to 1000s for Large Language Models (LLMs) like GPT-5 [**<empty citation>**].

A set of NPUs structured together is called cluster, and each cluster is able to take different amounts of them. How they are connected in each cluster is described by a physical network topology. Every cluster in one machine, or called node, respectively, communicates via intra-node-communication. NPUs communicating between different nodes is possible over Network Interface Cards (NICs), which are connected in an inter-node-network. While intra-node communication has low latencies and high bandwidths, inter-node-communication has rater high latencies and low bandwidths. This would make a distributed training with few machines attractive, but realistic distribution onto many machines is much more scalable [**<empty citation>**]. That is due to the limited amount of GPUs in one server [**<empty citation>**]. Therefore, large distributed systems use and depend on both, intra and inter-node-communication. The topologies of distributed systems can be based on different architectures. Verbreaken presents that there are four types of topologies that can exist for distributed systems [13]. Generally they can be divided into centralized, decentralized and fully distributed topologies, based on the degree of the distribution. A structural overview can be found in 2.2.

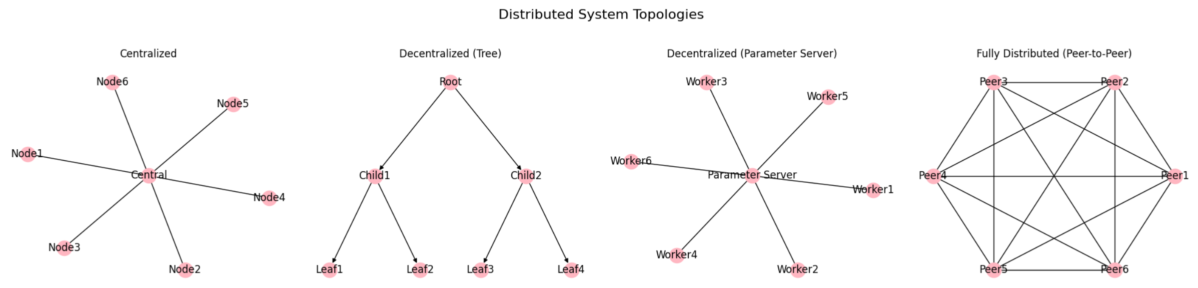

Figure 2.2: Network Topologies

The centralized topology is characterized by a strict hierarchical structure in which all nodes are connected with one central server that orchestrates them and performs the necessary steps for the combination of the machine learning. It serves a simple coordination but is limited in scalability due to having a single point of failure.

The tree topology is a decentralized topology in which the nodes are structured hierarchical. The communication takes place between the nodes in specific directions. Information is communicated upwards and distributed downwards. This is scalable as every node only communicates with children or parents.

The parameter server topology is a second decentralized topology that combines centralized parameter servers to store and retrieve gradients with decentralized ML nodes. These servers have a shared memory, so the parameters can be synchronized between multiple parts. The disadvantage of this is that all communication happens at the parameter servers, creating bottlenecks if many workers are combined with few servers.

A fully distributed topology works like a peer-to-peer network, meaning each node holds its own copy of the models parameters and is able to communicate with all other nodes. Communication is possible in every direction and at any time. This is very scalable, as new workers can be added without a high centralized overload, but it has the disadvantage that the communication can create a high overhead if every worker broadcasts its information.

Topologies can be asymmetric or symmetric, depending on if the connections are bidirectional or unidirectional. For fully distributed systems both could be possible, while all other topologies depend on bidirectional connections [14].

Communication Strategies

While physical topologies are showing different ways of organizing and centralizing the connections of machines, logical topologies can be used to specify how the actual communication is practiced. For this two main communication strategies a distinction between two new types of parallelization has to be made. They can either be synchronous, meaning all machines communicate at the same time collaboratively, or asynchronous, where each worker communicates as soon as it is finished with its own computation. Both are not applicable to every physical topology. Asynchronous communication is best suited for decentralized topologies with parameter servers. In that case workers push and pull parameters anytime necessary without the need for waiting for other workers to finish. Possible are also more varying approaches such as a trade-off version, called local Stochastic Gradient Descent (SGD), that allows workers to communicate asynchronous for a set amount of time until needing to synchronize [13] or Stale Synchronous Parallel (SSP), which additionally allows for cached parameters to be used for synchronization for the set amount of time [15].

Depending on the parallelization strategies and split of the training process varying types of communication have to be performed. While data parallelism relies on frequent exchange of calculated gradients, for example tensor parallelism, a type of model parallelism, needs to communicate between steps as early as in the forward pass [16].

Those types of communications can be achieved by using for example communication collectives. That are a set of synchronized communication patterns, which can be used for sharing and combining information across multiple machines in distributed systems, for instance *All-Reduce*, *All-to-All* or *Reduce-Scatter*. These collectives are most common for data parallelism, as they help to synchronize and combine gradients, and share parameters. They are blocking, meaning, nodes have to wait for each other's computation to finish, and they have to be orchestrated by a centralized controller. Generally, collectives can be separated into redistributive operations, that share data and consolidative operations, that aggregate data. The most common collective used for data parallelism is *All-Reduce*. In the following the approach of collectives is explained, based on *All-Reduce* as an example. For guidance, figure 2.3 shows a visualization of the collective. The source for this explanation is the 25th course lecture of UC Berkeley's course *CS 168* in spring 2025 [17].

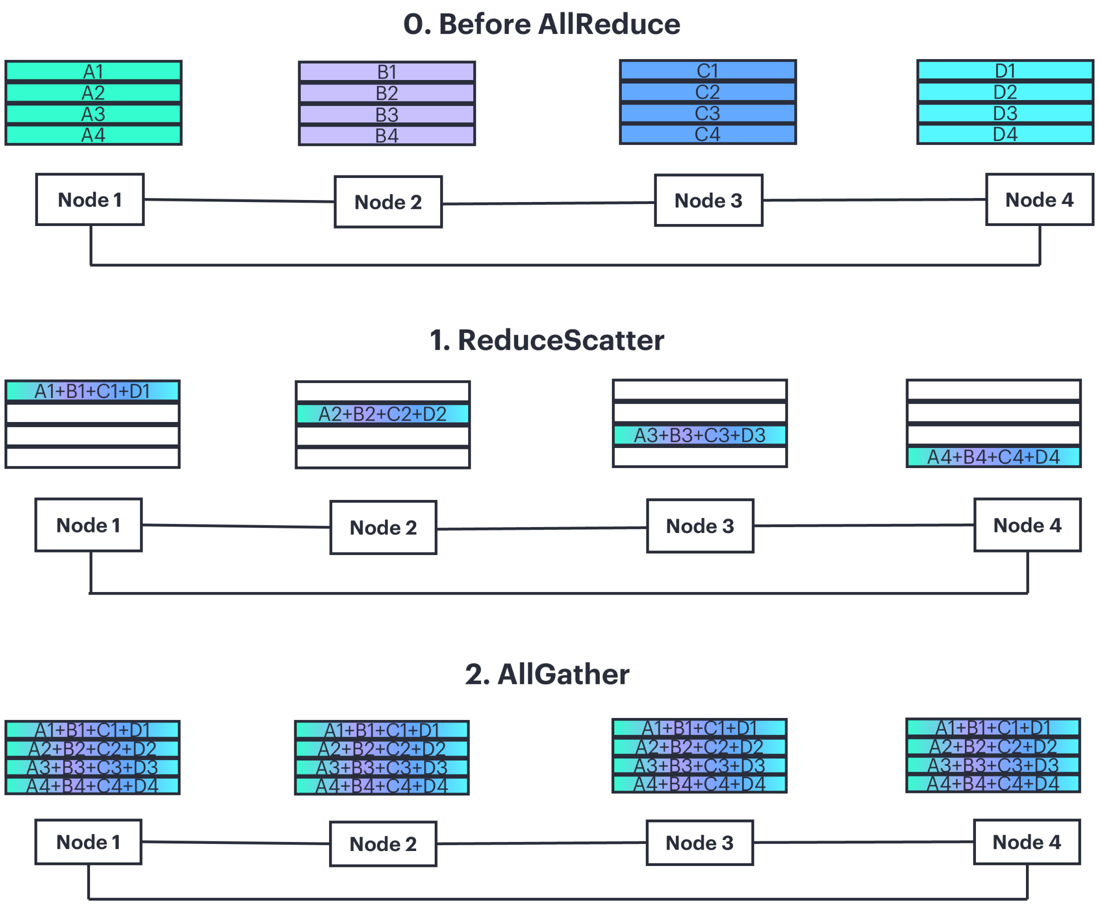

Figure 2.3: All-Reduce

This collective can be used for p nodes with each having a p-sized vector of values. This example uses $p = 4$, four nodes with each having four parameters that the model gets trained on. Every node needs to know the number of p and which index they have, meaning which parameter they are going to be responsible for. Also, this example uses a ring topology for easy visualization. The actual implementation can differ, in the following the theoretical background gets explained, more on that afterward.

In every iteration, the communication starts when all workers, here called nodes, are finished with their computation. Every Node has their four parameters stored. The first step of the *All-Reduce* is a *Reduce-Scatter*, which itself involves two steps, a *Scatter* and a *Reduce*. *Scatter* is the redistributive operation that shares every i-th parameter to the i-th node. With this, the first node receives all first parameters, the second all second ones and so on. *Reduce* is the consolidative operation, that combines the received set of parameters to one single aggregation. Together, they make every i-th node store one value in their i-th vector-place, combining the previous i-th values in every node's vector.

The second step is an *All-Gather*, which is an advanced version of *Gather*. That is an operation that is the reverse of *Scatter*, as it combines every i-th nodes i-th value in one vector. *All-Gather* expands this by a *Broadcast*, which is the reverse of *Reduce* and distributes the resulting vector to all nodes. Afterward, the *All-Reduce* is finished, and every node has a p-sized vector of the combined and shared values.

The used operations can be implemented on top of various topologies, which influences how efficient the operation performs. In figure 2.4, common logical topologies are visualized. They can be for example a *Mesh*, *Tree*, or *Ring*.

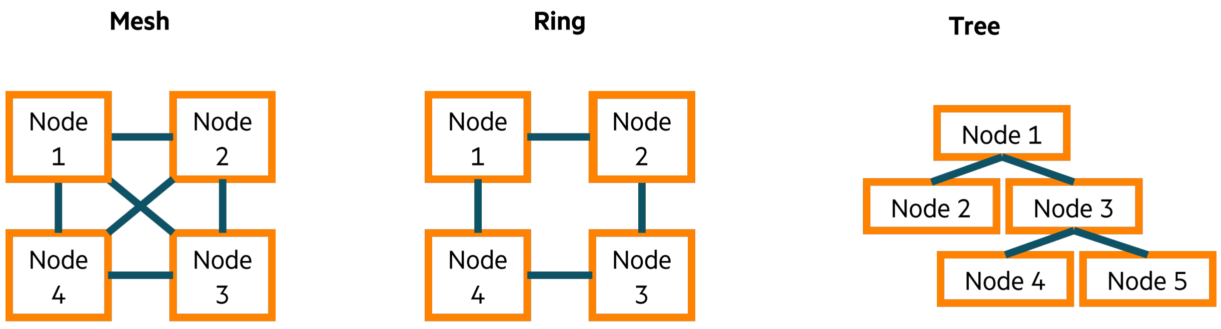

Figure 2.4: Logical Topologies

Depending on the underlying physical topology, different logical topologies can be useful for the different collectives, as different complexities of bandwidths are relevant. Also, overlay topologies can be used to create virtual links to support exchanging data between unconnected nodes.

DML Characteristics

??

2.2 Performance Modeling and Simulation Tools

2.3 User Interface Design for Scientific Tools

3 Design

3.1 Requirements

3.2 Requirements Analysis

3.3 Technology Selection

figures/methodology_flowchart.png

Figure 3.1: Methodology flowchart

3.4 UI/UX Design Process

figures/cli_vs_ui_workflow.png

Figure 3.2: Concept diagram: CLI workflow vs UI workflow

4 Implementation

4.1 Optional: Overall Architecture

`figures/system_architecture.png`

Figure 4.1: System architecture diagram

4.2 Frontend Design

figures/final_ui_screenshot.png

Figure 4.2: Final UI screenshot

4.3 Backend Integration

5 Evaluation

5.1 Feedback Integration

5.2 Challenges

5.3 Solutions

6 Conclusion and Future Work

6.1 Summary of Findings

6.2 Limitations

6.3 Next Steps

6.4 Future Work

Bibliography

[1] A. M. Turing. “Computing machinery and intelligence (1950).” In: *Perspectives on the computer revolution*. USA: Ablex Publishing Corp., June 1989, pp. 85–107. ISBN: 978-0-89391-369-4. (Visited on 08/16/2025).

[2] A. L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers.” In: *IBM Journal of Research and Development* 3.3 (July 1959), pp. 210–229. ISSN: 0018-8646. DOI: 10.1147/rd.33.0210. URL: https://ieeexplore.ieee.org/document/5392560 (visited on 08/16/2025).

[3] N. Abramson, D. Braverman, and G. Sebestyen. “Pattern recognition and machine learning.” en. In: *IEEE Transactions on Information Theory* 9.4 (Oct. 1963), pp. 257–261. ISSN: 0018-9448. DOI: 10.1109/TIT.1963.1057854. URL: http://ieeexplore.ieee.org/document/1057854/ (visited on 08/16/2025).

[4] Raúl Rojas. “The Backpropagation Algorithm.” en. In: *Neural Networks: A Systematic Introduction*. Ed. by Raúl Rojas. Berlin, Heidelberg: Springer, 1996, pp. 149–182. ISBN: 978-3-642-61068-4. DOI: 10.1007/978-3-642-61068-4_7. URL: https://doi.org/10.1007/978-3-642-61068-4_7 (visited on 08/17/2025).

[5] Ignacio Perez Karich and Simon Joss. “Emergence and Evolution of ‘Big Data’ Research: A 30-Year Scientometric Analysis of the Knowledge Field.” en. In: *Metrics* 2.3 (Sept. 2025). Publisher: Multidisciplinary Digital Publishing Institute, p. 15. ISSN: 3042-5042. DOI: 10.3390/metrics2030015. URL: https://www.mdpi.com/3042-5042/2/3/15 (visited on 08/17/2025).

[6] Věra Kůrková. “Limitations of Shallow Networks.” en. In: *Recent Trends in Learning From Data*. Ed. by Luca Oneto et al. Vol. 896. Series Title: Studies in Computational Intelligence. Cham: Springer International Publishing, 2020, pp. 129–154. ISBN: 978-3-030-43882-1 978-3-030-43883-8. DOI: 10.1007/978-3-030-43883-8_6. URL: http://link.springer.com/10.1007/978-3-030-43883-8_6 (visited on 08/18/2025).

[7] Jaime Sevilla et al. “Compute Trends Across Three Eras of Machine Learning.” en. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. arXiv:2202.05924 [cs]. July 2022, pp. 1–8. DOI: 10.1109/IJCNN55064.2022.9891914. URL: http://arxiv.org/abs/2202.05924 (visited on 08/18/2025).

[8] Gordon Moore. “Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.” In: *Solid-State Circuits Newsletter, IEEE* 11 (Oct. 2006), pp. 33–35. DOI: 10.1109/N-SSC.2006.4785860.

[9] Thomas N. Theis and H.-S. Philip Wong. “The End of Moore’s Law: A New Beginning for Information Technology.” In: *Computing in Science & Engineering* 19.2 (Mar. 2017), pp. 41–50. ISSN: 1558-366X. DOI: 10.1109/MCSE.2017.29. URL: https://ieeexplore.ieee.org/abstract/document/7878935 (visited on 08/18/2025).

[10] R. Bekkerman, M. Bilenko, and J. Langford. *Scaling up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press, 2011. ISBN: 978-1-139-50190-3. URL: https://books.google.com/books?id=9u0gAwAAQBAJ.

[11] Wenxue Li et al. “Understanding Communication Characteristics of Distributed Training.” en. In: *Proceedings of the 8th Asia-Pacific Workshop on Networking*. Sydney Australia: ACM, Aug. 2024, pp. 1–8. ISBN: 979-8-4007-1758-1. DOI: 10.1145/3663408.3663409. URL: https://dl.acm.org/doi/10.1145/3663408.3663409 (visited on 08/18/2025).

[12] Matthias Boehm et al. “Hybrid parallelization strategies for large-scale machine learning in SystemML.” In: *Proc. VLDB Endow.* 7.7 (Mar. 2014), pp. 553–564. ISSN: 2150-8097. DOI: 10.14778/2732286.2732292. URL: https://doi.org/10.14778/2732286.2732292 (visited on 08/16/2025).

[13] Joost Verbraeken et al. “A Survey on Distributed Machine Learning.” In: *ACM Comput. Surv.* 53.2 (Mar. 2020), 30:1–30:33. ISSN: 0360-0300. DOI: 10.1145/3377454. URL: https://dl.acm.org/doi/10.1145/3377454 (visited on 08/16/2025).

[14] William Won et al. “TACOS: Topology-Aware Collective Algorithm Synthesizer for Distributed Machine Learning.” In: *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. ISSN: 2379-3155. Nov. 2024, pp. 856–870. DOI: 10.1109/MICRO61859.2024.00068. URL: https://ieeexplore.ieee.org/abstract/document/10764470 (visited on 08/16/2025).

[15] Zhenheng Tang et al. *Communication-Efficient Distributed Deep Learning: A Comprehensive Survey*. arXiv:2003.06307 [cs]. Sept. 2023. DOI: 10.48550/arXiv.2003.06307. URL: http://arxiv.org/abs/2003.06307 (visited on 08/16/2025).

[16] Harry Dong et al. *Towards Low-bit Communication for Tensor Parallel LLM Inference*. arXiv:2411.07942 [cs]. Nov. 2024. DOI: 10.48550/arXiv.2411.07942. URL: http://arxiv.org/abs/2411.07942 (visited on 08/21/2025).

[17] : Sylvia Ratnasamy, Rob Shakir, and Kao Peyrin. *Collective Operations*. en. Lecture. Berkeley, CA, 2025. URL: https://textbook.cs168.io/beyond-client-server/collective-operations.html (visited on 08/21/2025).