OmniFed: A Modular Framework for Configurable Federated Learning from Edge to HPC

Sahil Tyagi*
Oak Ridge National Laboratory (ORNL)
Oak Ridge, Tennessee, USA
tyagis@ornl.gov

Andrei Cozma*
University of Tennessee
Knoxville, Tennessee, USA
acozma@vols.utk.edu

Olivera Kotevska
Oak Ridge National Laboratory (ORNL)
Oak Ridge, Tennessee, USA
kotevskao@ornl.gov

Feiyi Wang
Oak Ridge National Laboratory (ORNL)
Oak Ridge, Tennessee, USA
fwang2@ornl.gov

Abstract

Federated Learning (FL) is critical for edge and High Performance Computing (HPC) where data is not centralized and privacy is crucial. We present OmniFed, a modular framework designed around decoupling and clear separation of concerns for configuration, orchestration, communication, and training logic. Its architecture supports configuration-driven prototyping and code-level override-what-you-need customization. We also support different topologies, mixed communication protocols within a single deployment, and popular training algorithms. It also offers optional privacy mechanisms including Differential Privacy (DP), Homomorphic Encryption (HE), and Secure Aggregation (SA), as well as compression strategies. These capabilities are exposed through well-defined extension points, allowing users to customize topology and orchestration, learning logic, and privacy/compression plugins, all while preserving the integrity of the core system. We evaluate multiple models and algorithms to measure various performance metrics. By unifying topology configuration, mixed-protocol communication, and pluggable modules in one stack, OmniFed streamlines FL deployment across heterogeneous environments. Github repository is available at https://github.com/at-aaims/OmniFed.

CCS Concepts

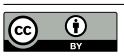
• **Computing methodologies** → **Cooperation and coordination**.

Keywords

Federated Learning (FL), Collaborative Learning (CL), Privacy-Preserving Machine Learning (ML), Edge computing, High Performance Computing (HPC), Deep Learning (DL), Compression

ACM Reference Format:
Sahil Tyagi, Andrei Cozma, Olivera Kotevska, and Feiyi Wang. 2025. OmniFed: A Modular Framework for Configurable Federated Learning from Edge to HPC. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3731599.3767397

*These authors contributed equally to this work.



SC Workshops '25, St Louis, MO, USA

ACM ISBN 979-8-4007-1871-7/25/11
https://doi.org/10.1145/3731599.3767397

1 Introduction

As data becomes increasingly distributed, sensitive, and voluminous, conventional centralized Artificial Intelligence (AI) pipelines are no longer practical today. Federated Learning (FL) and Collaborative Learning (CL) techniques developed over the last decade are crucial to develop Deep Neural Network (DNN) models that learn not just from open, publicly available data, but also from private, sensitive data residing at restricted endpoints. Algorithms like Federated Averaging [50] train over devices with limited compute, skewed data quality/quantity, and slow networks. From a distributed computing perspective, this intuition involves moving the computation instead of moving data [19, 81]. FL emphasizes privacy-preserving training over sensitive or regulated datasets such that data cannot be relocated or reconstructed via model or output inversion attacks. On the other hand, CL focuses on decentralized, in-situ model training where privacy may be less critical, but data locality and institutional boundaries remain key drivers. Together, FL/CL offer a transformational opportunity for the next generation of secure and distributed AI where models are trained or fine-tuned across institutional, geographic, and disciplinary boundaries without relocating the data. Development of novel FL/CL systems and training strategies is thus crucial for scientific collaborations, national security applications, and industrial partnerships. Training models where the data resides: on partner/shared systems, scientific instruments, and secure edge devices, can help advance both open science and protected interests/missions simultaneously.

We present OmniFed, a Python-based modular, extensible, configurable, and open-source framework built atop PyTorch [56] with the goal of enabling FL/CL from the edge to High Performance Computing (HPC) systems, spanning scientific and healthcare instruments, edge computing environments, cloud resources, and large-scale HPC systems. Built with layered abstractions and clear separation of concerns, OmniFed works in a plug-and-play and override-what-you-need manner via lifecycle hooks. Hiding the complexity of training topology and underlying communication allows rapid prototyping of new FL/CL algorithms without excessive boilerplate. Thus, the end Machine Learning (ML) user focuses

solely on training logic (like algorithm, model architecture, loss computation, optimization strategy, etc.) while the framework handles low-level orchestration details and scaffolding. OmniFed makes it easy to implement and deploy complex, custom training topologies (such as centralized, decentralized, hierarchical, etc.) via Hydra configuration management [79, 80]. It also comes with built-in privacy features like Differential Privacy (DP) [22], Homomorphic Encryption (HE) [27] and Secure Aggregation (SA) [13], gradient compression techniques [71, 75], streaming simulation for real-time learning [74], and 10+ FL algorithms. The layout of this article is as follows: §2 describes prior works in the FL/CL space and the need for secure, distributed AI for collaborative science, healthcare, finance applications, then §3 explains the design principles and core components of OmniFed along with preliminary results, and §4 describes our current and future directions of our work.

2 Challenges and Related Work

2.1 Need for Federated AI/ML Systems

Language and reasoning models already trained on publicly available internet-scale knowledge have reduced the need for pretraining generic DNNs. The next frontier of computing lies in leveraging these models for private, sensitive, restricted, or distributed datasets for tasks like scientific discovery, healthcare, finance, and other critical applications. Further training or fine-tuning models over such datasets allows for personalized, context-aware intelligence for individual users [31], and also help organizations working at the intersection of HPC, security, and scientific discovery. The decentralized nature of FL/CL offers a new approach to scientific workflows that have prior relied on centralized data aggregation and monolithic data pipelines. The ever-increasing volume, sensitivity, and distribution of scientific data, ranging from neutron scattering experiments [65] to genomic repositories [52], have created technical, logistical, and ethical barriers to centralization, thus making it infeasible or prohibited. In the realm of open models and sciences, FL enables collaborative AI/ML across research institutions, laboratories, academia, and industry while preserving data governance. This supports efforts like climate modeling, materials science, cyber-threat detection, and energy infrastructure monitoring, while retaining local data control. The integration of FL/CL with edge, cloud, and HPC systems thus creates a new capability tier of privacy and security-aware distributed intelligence operating across layers of the scientific enterprise.

In the context of national labs and computing facilities, FL opens up new frontiers in scientific modeling by combining federated models with HPC-powered simulations and developing self-updating digital twins [14] for nuclear reactors, electric grids, materials, and climate systems. The convergence of FL and simulations enables adaptive, real-time modeling at scale, offering breakthrough capabilities for monitoring, control, and prediction in complex systems.

2.2 Current Frameworks and Limitations

Many frameworks have been developed in recent years, each with distinct pros and cons. For instance, TensorFlow Federated (TFF) [12] developed by Google is Python-based with tight TensorFlow [2] integration to simulate FL experiments. Although TFF is excellent for research and experimentation, it lacks scalability so it may not be ideal for real-world deployment. NVFLARE [62] developed by NVIDIA is an active and well-documented FL framework supporting TensorFlow and PyTorch that also supports privacy features. However, it faces scalability challenges where performance may drop as the number of active clients rises. Communicating large models between clients and server poses scalability challenges with Google Remote Procedure Call (gRPC) protocol [29], thus demanding high memory/network bandwidth for data exchange. Flower [11] is language and ML-backend agnostic, and supports both simulations and real-world deployments. However, it can require manual orchestration for advanced deployments. Although privacy techniques like SA have been integrated, privacy tools are still in development, and advanced use cases may require extra customization. Flower lacks advanced production tools for monitoring and orchestration, and supports synchronous training by default, which can be prone to straggler slowdowns [72, 73].

OpenFL [61] by Intel is a Python-based FL framework designed for multi-institution settings, optimized for secure and cross-org FL in regulated industries via Trusted Execution Environment (TEE). However, it only supports centralized server setup (prone to a single point of failure), is built mainly around Intel platforms, lacks built-in privacy tools, and has demonstrated limited mobile/edge use cases. OpenFL does not support decentralized or asynchronous training. Medical Open Network for AI (MONAI) [16] is a PyTorch-based, domain-optimized framework for FL over medical and healthcare applications, and uses NVIDIA-based tools for distributed, mixed-precision training. However, MONAI is specialized for medical imaging and lacks direct support for other healthcare domains (e.g., genomics), optimized primarily for NVIDIA GPUs, and may require additional support for integration with clinical systems. PySyft [82] by OpenMined treats privacy as a first-class citizen for Python-based AI/ML workflows using PyTorch and TensorFlow. It contains a rich privacy toolkit with features like HE, DP, and Secure Multi-Party Computation (SMPC), as well as built-in access controls. However, PySyft requires additional setup and lacks built-in orchestration for FL workflows. As a standalone, PySyft runs in simulation mode only. Running FL at scale requires additional integration with other OpenMined components like PyGrid, which may make real-world deployment non-trivial.

FedML [32] is Python-based, framework agnostic, and optimized for cross-platform FL leveraging cloud and on-device learning over edge/mobile devices. It also encapsulates various privacy-preserving techniques and trusted execution to mitigate data leakage risks. However, FedML can face scalability issues in large-scale training and struggle with data or systems heterogeneity. APPFL [44, 63] developed at Argonne National Laboratory is designed for research institutions running cross-silo experiments. The framework is developed with a modular design, supports advanced FL algorithms, has built-in privacy features like DP, and supports communication backends like Message Passing Interface (MPI) [70], gRPC, and Globus Compute [17], supports synchronous/asynchronous scheduling and lossy compression [71] to reduce communication overhead. However, its dependency requirements may make deployment non-trivial in complex environments. IBM FL [49] is an enterprise-grade framework with visualization dashboards that enables application monitoring, comes with various security tools and advanced aggregation methods. It's also

(a) Centralized (b) Ring (c) Peer-to-Peer (d) Hierarchical Tree

Figure 1: Participants connected under different topological arrangements or configurations (central, ring, p2p, and hierarchical).

framework-agnostic and uses communication protocols like gRPC and WebSockets. However, IBM FL is not entirely open-source, and production-grade access comes through commercial offerings. The integrated tools can be heavyweight and potentially make deployment cumbersome and complex.

Flotilla [9] supports both PyTorch and TensorFlow, works with heterogeneous edge devices, and supports advanced FL algorithms. The system is less focused on privacy and security, and deployment orchestration can be complex in distributed settings. Apple's private FL (pfl) [30, 37] is a privacy-centric framework that supports PyTorch, TensorFlow, and non-neural models like Gradient Boosting Decision Trees (GBDTs). However, the toolkit is designed for simulations instead of direct production deployments. It also has limited portability outside of Apple's ecosystem as it uses proprietary infrastructure like secure enclaves. FATE [47], developed by China-based WeBank, supports industrial-grade FL with secure computation features built into it that make it ideal for finance and enterprise use. The framework runs on standalone systems, Docker [51], Kubernetes [40], and Spark [81]. However, setup can be challenging and complex with all its dependencies initially. These frameworks have different strengths and weaknesses, and may trade off between ease-of-use and scalability, privacy, or feature-rich orchestration. We develop OmniFed to provide users with clear-cut abstractions to choose between these trade-offs in a modular fashion with minimal code change. By hiding away the scaffolding and orchestration details, FL researchers can focus on developing algorithms and optimization strategies to produce secure and quality models.

3 OmniFed Framework

3.1 Motivation

We develop OmniFed with the goal to enable researchers and developers to easily evaluate different FL/CL algorithms, as well as permit rapid prototyping of new learning algorithms without excessive boilerplate. For instance, switching from FedAvg [50] to FedProx [64] should ideally be a single line change that allows users to use different algorithms. Additionally, most frameworks assume a centralized training topology (shown in Figure (1a)), so implementing custom learning strategies across skewed and unbalanced islands of heterogeneous clients or levels of topology (e.g., hierarchical tree in Figure (1d)) can be non-trivial and even complex. Federated nodes/devices may be connected in different topologies, as shown in Figure (1), each offering distinct trade-offs. For instance, a centralized or star topology comprises a single coordinator that communicates with all participants, or a decentralized topology where a node is connected only to its immediate neighbors, or a peer-to-peer topology where every participant can directly communicate with every other node. A hierarchical tree consists of a multi-level structure with parent-child aggregation paths. At the same time, a hub-and-spoke topology may contain islands of densely connected nodes, with sparse connections across regions/facilities. Thus, a flexible FL framework should allow both simulation and real deployment across different clusters and training configurations.

3.2 Design Philosophy and Principles

OmniFed is developed to treat modularity, flexibility, and extensibility as first-class citizens in the FL/CL ecosystem. With precise, layered abstractions, local computation, communication logic, and algorithmic control strategy are well separated and can easily be swapped out with OmniFed's configuration-driven, schema-based workflow that is agnostic to topology (such as those shown in Figure (1)) and communication protocols (e.g., gRPC [29], MPI [70], Message Queuing Telemetry Transport (MQTT) [35]). With distinct and minimal interfaces, plug-and-play components, and an override-what-you-need paradigm, users can add or remove different privacy features and compression techniques with minimal code change. We provide single-file algorithm plugins where existing FL techniques can be evaluated with a line change in a job's YAML configuration file. Additionally, new FL/CL algorithms can be easily prototyped by extending the base algorithm class and overriding abstract methods. With a developer-friendly, consistent, and intuitive usage pattern, OmniFed allows users to focus on core tasks instead of managing the scaffolding and setup around a job.

3.3 Core Components

We implement OmniFed in Python with PyTorch [56] as the core AI/ML backend, and use YAML-based configuration management with Hydra [79, 80], a framework for complex ML and data-science workflows that enables hierarchical composition, command-line overrides, and flexible job deployment. Jobs are deployed for distributed execution over Ray [7, 53], an AI compute engine for orchestrating distributed workloads across heterogeneous hardware. Ray also supports fine-grained parallelism across CPUs and GPUs, allowing users to easily scale jobs from a local machine to multi-node clusters. Based on this stack, OmniFed's core modules are:

- ***Engine*** is responsible for handling the orchestration rounds, resources, metrics, etc.
- ***Topology*** defines the node graph and coordination patterns.
- ***Node*** corresponds to a FL/CL participant that may hold a model, training data, and communicators. Based on the role of the node in executing a specific algorithm, it can either be a client, an aggregator or a relay.

- ***Communicator*** comprises of a unified Application Programming Interface (API) with abstract primitives that use different communication protocols under the hood (for e.g., gRPC [29], MPI [70], MQTT [35], etc.).
- ***Algorithm*** defines the local training logic and overall learning strategy through lifecycle hooks. Users can choose from a suite of built-in ones (see §3.4.1) or implement their own.

The *Engine* is the core orchestrator for all FL/CL experiments. It's responsible for launching and coordinating all distributed experiments, managing node lifecycle and resource allocation, and collecting report metrics and statistics. The OmniFed engine is instantiated via a Hydra-based YAML configuration file, and coordinates with *Topology* definitions to spawn *Node* actors (using Ray [53]) and manage *Algorithm* implementations.

The training *Topology* defines the structure, coordination, and scheduling patterns for distributed nodes. It's responsible for defining the node graph structure and relationships between nodes, supports templates for centralized, decentralized, and hierarchical topologies (as shown in Figure (1)). We are working on developing custom and complex topologies via *Topology*'s graph-based representations from the job's YAML configuration, to be used by the engine to spawn and organize *Node* actors. The edges of the graph will determine which nodes can communicate with each other.

Node can be any participant in the federation. It's responsible for managing local model state, data, and device/client resources, interfaces with the *Communicator* for data exchange, and executes the local *Algorithm* implementations. *Nodes* are instantiated as Ray actors by the OmniFed *Engine* and configured via Hydra YAML files with optional per-node overrides. A *Node* closely coordinates with *Algorithm* components for local compute logic like train, eval, etc. Currently, *Node* can serve the role of a trainer or aggregator. The former performs local training on data, while the latter collects, merges, and redistributes model updates. Every *Node* implements an inner communicator by default for its topology. To implement hierarchical FL, *Nodes* additionally implement an outer communicator to exchange data across sub-topologies.

Communicator is needed for data exchange among *Nodes* like model parameters, gradients, control signals, etc. The module exposes a unified API that abstracts the underlying protocol while standardizing data exchange operations with consistent behavior and configurable backend selection without code changes. We currently implement MPI collectives via `TorchDistCommunicator` using PyTorch Distributed (so works with various underlying libraries such as NVIDIA Collective Communications Library (NCCL) [54], MPI [69], or Gloo [23]). A `GrpcCommunicator` based on gRPC and protocol buffers [28], has a server that receives, aggregates, and broadcasts updates sent by clients over heterogeneous networks in distributed FL/CL. For middleware systems, we are additionally developing Advanced Message Queuing Protocol (AMQP)-based [76] `AMQPCommunicator` to support publish-subscribe message patterns where clients push updates to a queue, which is subsequently pulled by the aggregator *Node*.

Algorithm provides FL logic through configurable lifecycle hooks, and called by the *Node* at various stages, such as local training, model update, post-aggregation processing, etc. This abstraction separates concerns through well-defined interfaces; *Node* manages resource and data, while *Algorithm* provides the learning strategy. The module is flexible so that different implementations can override only what they need, and extensible to support diverse FL/CL paradigms and topologies. A Hydra-based YAML file describes the simulation or deployment configuration. OmniFed comes with prebuilt *Topology* templates that enable quick and straightforward configurations for common patterns like centralized, decentralized, and hierarchical. We are actively working on implementing custom graphs with explicitly defined nodes and edges. Each *Node* contains the following information: role, *Communicator*, PyTorch model, data, and *Algorithm*, while a *Topology* describes a set of *Nodes* and edges, and an *Engine* is an orchestration point for *Topology*. We also integrate various privacy-preserving techniques into OmniFed, like DP [3, 22], HE [27] and SA [13]. To add DP, we integrate with PETINA [55], a library of privacy-preserving algorithms, while HE is added through Tenseal [10] based on Microsoft SEAL [18]. SA is currently prototyped with Hash-based Message Authentication Code (HMAC) [58] and Hashlib [57] to generate a shared key between any two clients in a deterministic manner. We plan to replace this with Diffie-Hellman key exchange [20].

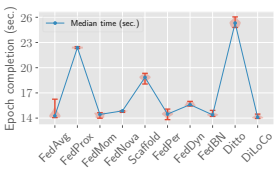
```yaml
defaults:
  - override topology: centralized
  - override algorithm: fedavg
  - override model: resnet18
  - override datamodule: cifar10

topology:
  _target_: src.omnifed.topology.CentralizedTopology
  num_clients: 8
  inner_comm:
    _target_: src.omnifed.communicator.GrpcCommunicator
    master_port: 50051
    master_addr: 127.0.0.1

algorithm:
  _target_: src.omnifed.algorithm.FedAvg
  lr: 0.01

global_rounds: 2
```

Figure 2: Example config for centralized FedAvg on ResNet18. From the YAML config, users can change the algorithm (e.g., FedAvg to FedProx), type of topology (centralized to decentralized), communicator (gRPC or MPI), model, dataset, aggregation frequency/rounds and hyperparameters (step-size).

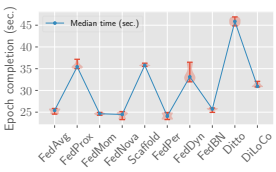
3.4 Configuration and Deployment

We test with 16 clients over NVIDIA DGX server with 8 H100 GPUs to demonstrate OmniFed's features and ease-of-use.

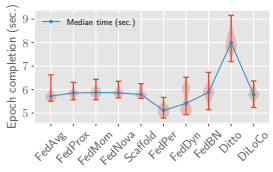
Training setup: ResNet18 [33] over CIFAR10 [1] is trained with initial Learning Rate (LR) 0.01, momentum 0.9, weight decay 1e-4, cross-entropy loss, per-node batch-size 32, and LR decay of 0.1 at 100, 150, and 200 epochs. VGG11 [68] over CIFAR100 [1] uses LR 0.01, momentum factor 0.9, weight decay 5e-4, node batch-size 32, and LR decay 0.2 at 50, 75, and 100 epochs respectively. AlexNet [39] over CalTech101 [25] trained with LR 0.01, momentum 0.9, weight decay 5e-4, batch-size 32, and LR decay 0.1 at 25, 50, and 75 epochs. MobileNetV3 [34] over CalTech256 [25] runs with LR 0.1, momentum 0.9, decay 1e-4 and LR decay 0.1 every 40 epochs.



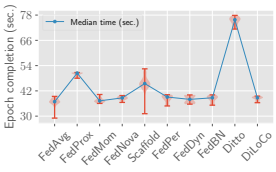
(a) ResNet18



(b) VGG11



(c) AlexNet



(d) MobileNetV3

Figure 3: Epoch completion time for different FL algorithms.

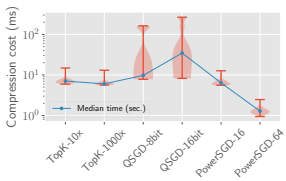
Algorithm	ResNet18	VGG11	AlexNet	MobileNetV3
FedAvg	99.32%	**86.6%**	87.9%	81.35%
FedProx	99.26%	86.31%	87.98%	**82.96%**
FedMom	99.14%	66.39%	63.85%	48.98%
FedNova	91.18%	14.1%	58.1%	22.27%
Moon	**99.46%**	81.67%	87.28%	81.4%
FedPer	90.9%	26.93%	82.94%	14.59%
FedDyn	99.31%	86.18%	**88.78%**	79.15%
FedBN	99.33%	86%	88.7%	78.65%
Ditto	73.64%	5.5%	40%	9.84%
DiLoCo	84.88%	5.1%	45.17%	15.47%

Table 1: Convergence quality of various FL algorithms.

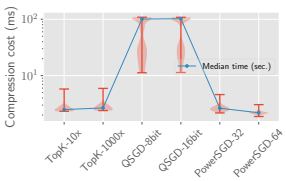
3.4.1 Switching between different algorithms. We have currently implemented the following in OmniFed's *Algorithm* module: FedAvg [50], FedProx [64], FedMom [36], FedNova [78], Scaffold [38], Moon [41], FedPer [8], FedDyn [5], FedBN [43], Ditto [42] and DiLoCo [21]. An example Hydra-based YAML configuration for training ResNet18 on CIFAR10 over a centralized topology with gRPC communicator and FedAvg algorithm is illustrated in Figure (2). The training algorithm can easily be swapped at line 16, and the specific hyperparameters can subsequently be defined in the config. For e.g., we can change FedAvg to FedProx and add the proximal-term related parameter μ under the algorithm section. This makes comparing different FL/CL algorithms fairly easy with minimal code change, with results shown in Table (1) and Figure (3). The latter logs the average epoch completion time across the four models while former denotes the final test accuracy of various algorithms. FL developers/researchers can thus quickly deploy their applications with different configurations and compare both the systems' and statistical performance of different algorithms. Some algorithms like DiLoCo are configured for specific settings (e.g., large language models with AdamW [48] for local optimization and momentum Stochastic Gradient Descent (SGD) for cross-client aggregation) and may exhibit sub-optimal performance in other settings. We do not perform extensive hyperparameter tuning as

```
1  inner_comm:
2      _target_: src.omnifed.communicator.TorchDistCommunicator
3      port: 28670
4      compression:
5          _target_: src.omnifed.communicator.compression.TopK
6          k: 1000x
```

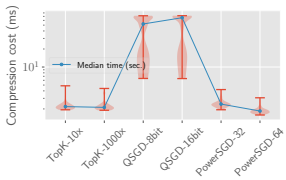
Figure 4: Configuring Communicator with Compression. Different compressors (and its related parameters) can be specified from YAML config as part of the communication module.



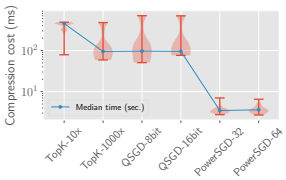
(a) ResNet18



(b) VGG11



(c) AlexNet



(d) MobileNetV3

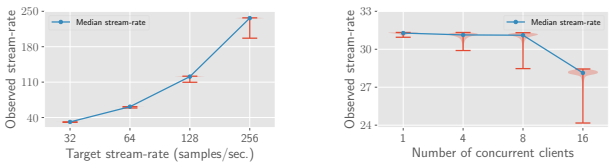
Figure 5: Compression overhead of different techniques.

Compression	ResNet18	VGG11	AlexNet	MobileNetV3
Top$k$-10×	99.09%	84.6%	87.23%	78.82%
Top$k$-1000×	99.05%	78.05%	75.75%	77.56%
DGC-10×	99.18%	84.22%	87.61%	79%
DGC-1000×	98.5%	78.75%	76.28%	72.77%
QSGD 8-bit	**99.26%**	85.48%	**89%**	**79.72%**
QSGD 16-bit	99.12%	**85.52%**	88.3%	77.24%
PowerSGD r-64	99.07%	75.45%	75%	77.69%
PowerSGD r-32	90.31%	6.74%	40%	77.79%

Table 2: Convergence quality in gradient compression.

the demonstration is intended to validate OmniFed's support for diverse algorithms under unified configuration. In default settings, ResNet18 achieved highest accuracy with Moon, VGG11 with FedAvg, AlexNet with FedDyn, and MobileNetV3 with FedProx. The current results were obtained with the default parameters specified for an algorithm (see github); performance may further be improved via algorithm-specific hyperparameter tuning, which is easily accomplished by modifying the configuration file.

3.4.2 Communication savings via Compression. Similar to §3.4.1, we compare the overall parallel and statistical performance of various compression techniques. Currently, we have implemented the



(a) Effective stream-rate (b) Multi-client stream-rate

Figure 6: Streaming simulation for real-time learning.

following gradient sparsification methods: Top$k$ [67], Deep Gradient Compression (DGC) [46], RedSync [24], SIDCo [4] and Random$k$ [75], Quantized Stochastic Gradient Descent (QSGD) quantization [6] and PowerSGD low-rank approximation [77]. A compressor and its parameters can be specified from within the config as part of the communicator, as shown for Top$k$ 1000× in Figure (4). Figure (5) shows the overhead of different compressors at various compression factors, while Table (2) logs the final test accuracy. Top$k$ and DGC are compressed to 10× and 1000× factors, while QSGD compresses to 2× and 4× with 16-bit and 8-bit, respectively (compressed w.r.t. 32-bit floats). The gradients are decomposed to rank 32 and 64 with PowerSGD compression. Although QSGD generally attains better accuracy than other methods (likely from using lower compression factors 2× and 4×), it comes at the cost of higher compression + communication cost (from Figure (5)). The total overhead of different compressors may vary due to the effective compression factor applied to reduce the volume of updates, and the communication collective used to exchange updates. Sparsification methods like DGC use all-gather, while quantization and low-rank approximation techniques are compatible with all-reduce.

3.4.3 Real-time learning over streaming data. To simulate settings with streaming data, we integrate OmniFed with Apache Kafka [26] to parse datasets that are subsequently published to Kafka topics. Kafka handles ordering issues within a partition, and can be paired with streaming frameworks like Apache Flink [15] to handle late or out-of-order events. The data streamed to a specific client (corresponding to its topic) can be set to a specific stream-rate that a user sets. FL clients run a custom PyTorch dataloader that subscribes to a topic to collect the corresponding data. With a single publisher process running, we plot the effective stream-rate achieved under slow to fast streams in Figure (6a) and see how accurately we can simulate streaming data on a client. A target stream-rate of 32 is closely met even while serving 16 concurrent clients with a single producer process on the central server, as shown in Figure (6b).

3.4.4 Privacy-Preserving features. Similar to adding different communicators and compressors, users can also specify which privacy-preserving technique to use and its pertinent parameters. For instance, we can apply `DifferentialPrivacy` within the module `src.omnifed.privacy` and specify privacy budget with ϵ and δ from the YAML configuration itself. With this, we measure accuracy of various DNNs as we set ϵ to [1.0, 10.0] and δ=1e-5 in Table (3a). Large ϵ implies higher privacy budget, adding lesser noise to model updates for the same iterations, achieving more accuracy than ϵ=1. For a given model and training configuration, users can also select and compare different techniques based on privacy requirements and compute budget. We tabulate the same in Table (3b) and observe that cryptography-based privacy techniques like HE and SA carry significant computational overhead compared to DP (especially for larger models like VGG11 and AlexNet). Although this outlines the need for further optimizations to accelerate HE and SA in DNN training, FL developers can experiment with various privacy mechanisms based on their needs and use cases.

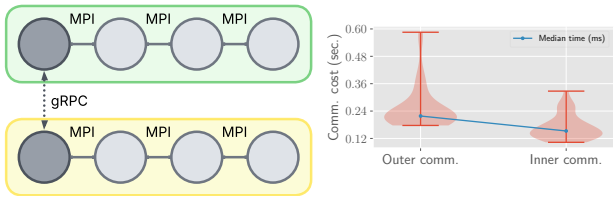
DNN	DP Acc. (%)	
	ϵ=1	ϵ=10
Res.	97.98	**98.06**
VGG	24.1	**28.6**
Alex.	16.31	**39.54**
Mob.	23.72	**58.8**

(a) Differential Privacy (DP)

DNN	Compute cost (sec.)		
	DP	HE	SA
Res.	1.45	68.72	229.6
VGG	14.4	786	2.3e3
Alex.	6.9	458.7	1.1e3
Mob.	1.2	29.8	83.3

(b) Privacy overhead

Table 3: Convergence quality and compute overhead associated with privacy-preserving methods DP, HE, and SA.



(a) Topology structure (b) Outer vs. Inner comm.

Figure 7: An example of cross-facility FL with multiple communication protocols and their respective overheads.

3.4.5 Custom topology with mixed-communication protocols. With OmniFed's modular and configurable design, users can perform cross-facility training where multiple sites (possibly geo-distributed) collaborate to learn a globally shared model. Figure (7a) simulates the aforementioned hub-and-spoke topology where nodes within a site are densely connected with high-bandwidth links, and sparsely connected across sites over high-latency, low-bandwidth network. Aggregation within a site can thus leverage bandwidth-optimal MPI collectives like ring-allreduce [66] over high bandwidths (corresponding to an inner communicator). At the same time, cross-site communication may use gRPC over relatively slower networks (i.e., cross-facility outer communicator). Here, inner comm. thus refers to data exchange within a sub-cluster, while outer comm. implies across sub-clusters. With OmniFed's modular approach, users can aggregate entire model updates over the fast, inner aggregator, and use compression (from §3.4.2) *only* over the slow, outer communicator (dashed line in Figure (7a)). Figure (7b) shows the overhead with MPI and gRPC as inner and outer communicators respectively. Inner comm. over MPI using bandwidth-optimal allreduce is faster than global gRPC based on client-server communication.

4 Conclusion and Future Work

In this work, we introduced OmniFed, a modular and flexible framework for FL/CL on edge and HPC systems. It enables rapid algorithm

prototyping through clear separation of concerns: configuration-driven topology definition, pluggable communication protocols, and supports override-what-you-need paradigm. It supports 10+ FL algorithms out-of-the-box, mixed-protocol communication (combining MPI and gRPC within a single deployment for enabling cross-facility training), pluggable privacy and compression methods. We emulate data-streaming scenarios, evaluate FL algorithms and gradient compression techniques to compare their performance. OmniFed addresses practical research needs for systematic and rapid FL prototyping, enabling researchers to focus on design and innovation rather than infrastructure and setup complexities. Although the current evaluation setup is limited to controlled single-facility settings, we are actively working on large-scale FL deployment via Docker and Slurm integration. We're plan to add peer-to-peer communication, MQTT protocol validation, heterogeneity-aware computing capabilities and integrate with optimized ML backends like DeepSpeed [60] and TorchTitan [45] for large models, and ExecuTorch [59] for edge devices. We believe such efforts will position OmniFed as a production-ready platform for collaborative research spanning over the full compute continuum.

Acknowledgments

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science, **Office of Advanced Scientific Computing Research** of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] 2025. CIFAR10 and CIFAR100. https://www.cs.toronto.edu/~kriz/cifar.html

[2] Martín Abadi, Paul Barham, Jianmin Chen, Z. Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. 2016. TensorFlow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation*.

[3] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. B. McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016).

[4] Ahmed Mohamed Abdelmoniem, Ahmed Elzanaty, Mohamed-Slim Alouini, and Marco Canini. 2021. An Efficient Statistical-based Gradient Compression Technique for Distributed Training Systems. *ArXiv* abs/2101.10761 (2021).

[5] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina, Paul N. Whatmough, and Venkatesh Saligrama. 2021. Federated Learning Based on Dynamic Regularization. *ArXiv* abs/2111.04263 (2021).

[6] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2016. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. In *Neural Information Processing Systems*.

[7] Inc. Anyscale. [n. d.]. Ray: A Fast and Simple Framework for Distributed Computing. https://www.ray.io/. Accessed: 2023-10-18.

[8] Manoj Ghuhan Arivazhagan, V. Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. 2019. Federated Learning with Personalization Layers. *ArXiv* abs/1912.00818 (2019).

[9] Roopkatha Banerjee, Prince Modi, Jinal Vyas, Chunduru Sri Abhijit, Tejus Chandrashekar, Harsha Varun Marisetty, Manik Gupta, and Yogesh L. Simmhan. 2025. Flotilla: A scalable, modular and resilient federated learning framework for heterogeneous resources. *J. Parallel Distributed Comput.* 203 (2025), 105103.

[10] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. 2021. TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption. *ArXiv* abs/2104.03152 (2021).

[11] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D. Lane. 2020. Flower: A Friendly Federated Learning Research Framework.

[12] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, H. B. McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. *ArXiv* abs/1902.01046 (2019).

[13] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. B. McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017).

[14] Wes Brewer, Matthias Maiterth, Vineet Kumar, Rafal P. Wojda, Sedrick Bouknight, Jesse Hines, Woong Shin, Scott Greenwood, David Grant, Wesley Williams, and Feiyi Wang. 2024. A Digital Twin Framework for Liquid-cooled Supercomputers as Demonstrated at Exascale. *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis* (2024), 1–18.

[15] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink : Stream and batch processing in a single engine. *IEEE Data(base) Engineering Bulletin* 36 (2015), 28–33. https://api.semanticscholar.org/CorpusID:263802939

[16] Manuel Jorge Cardoso, Wenqi Li, Richard Brown, Nic Ma, Eric Kerfoot, Yiheng Wang, Benjamin Murrey, Andriy Myronenko, and Can Zhao et al. 2022. MONAI: An open-source framework for deep learning in healthcare. *ArXiv* abs/2211.02701 (2022).

[17] Ryan Chard, Y. Babuji, Zhuozhao Li, Tyler J. Skluzacek, Anna Woodard, Ben Blaiszik, Ian T Foster, and Kyle Chard. 2020. funcX: A Federated Function Serving Fabric for Science. *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing* (2020).

[18] Hao Chen, Kim Laine, and Rachel Player. 2016. Simple Encrypted Arithmetic Library - SEAL v2.1. In *Financial Cryptography Workshops*.

[19] Muthu Dayalan. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51 (2008), 107–113.

[20] Whitfield Diffie and Martin E. Hellman. 1976. New Directions in Cryptography. *Democratizing Cryptography* (1976).

[21] Arthur Douillard, Qixuang Feng, Andrei A. Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. 2023. DiLoCo: Distributed Low-Communication Training of Language Models. *ArXiv* abs/2311.08105 (2023).

[22] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* 9 (2014), 211–407.

[23] Inc. Facebook. 2023. Gloo: Collective Communications Library. https://github.com/facebookincubator/gloo. Accessed: 2023-10-30.

[24] Jiarui Fang, Haohuan Fu, Guangwen Yang, and Cho-Jui Hsieh. 2018. RedSync : Reducing Synchronization Traffic for Distributed Deep Learning. *ArXiv* abs/1808.04357 (2018).

[25] Li Fei-Fei, Rob Fergus, and Pietro Perona. 2006. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28 (2006), 594–611.

[26] Apache Software Foundation. 2022. Apache Kafka. https://kafka.apache.org/. Version 3.2.0.

[27] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Symposium on the Theory of Computing*.

[28] Google. 2008. Protocol Buffers: Google's Data Interchange Format. Github. https://github.com/protocolbuffers/protobuf

[29] Google. 2015. gRPC: A high-performance, open-source universal RPC framework. https://grpc.io/

[30] Filip Granqvist, Congzheng Song, Áine Cahill, Rogier van Dalen, Martin Pelikan, Yi Sheng Chan, Xiaojun Feng, Natarajan Krishnaswami, Vojta Jina, and Mona Chitnis. 2024. pfl-research: simulation framework for accelerating research in Private Federated Learning. *arXiv preprint arXiv:2404.06430* (2024).

[31] Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, and Bowen Zhang et al. 2025. Apple Intelligence Foundation Language Models. *ArXiv* abs/2407.21075 (2025).

[32] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. 2020. FedML: A Research Library and Benchmark for Federated Machine Learning. *ArXiv* abs/2007.13518 (2020).

[33] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), 770–778.

[34] Andrew G. Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for MobileNetV3. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), 1314–1324.

[35] Urs Hunkeler, Hong Linh Truong, and Andy J. Stanford-Clark. 2008. MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks. *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)* (2008), 791–798.

[36] Zhouyuan Huo, Qian Yang, Bin Gu, Lawrence Carin, and Heng Huang. 2020. Faster On-Device Training Using New Federated Momentum Algorithm. *ArXiv* abs/2002.02090 (2020).

[37] An Ji, Bortik Bandyopadhyay, Congzheng Song, Natarajan Krishnaswami, Prabal Vashisht, Rigel Smiroldo, Isabel Litton, Sayantan Mahinder, Mona Chitnis, and Andrew W Hill. 2025. Private Federated Learning In Real World Application - A Case Study. *ArXiv* abs/2502.04565 (2025).

[38] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. 2019. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In *International Conference on Machine Learning*.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60 (2012), 84 – 90.

[40] Kubernetes 2025. Kubernetes Official Documentation. https://kubernetes.io/docs/. [Online; accessed 08-Aug-2025].

[41] Qinbin Li, Bingsheng He, and Dawn Xiaodong Song. 2021. Model-Contrastive Federated Learning. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), 10708–10717.

[42] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. 2020. Ditto: Fair and Robust Federated Learning Through Personalization. In *International Conference on Machine Learning*.

[43] Xiaoxiao Li, Meirui Jiang, Xiaofei Zhang, Michael Kamp, and Qi Dou. 2021. FedBN: Federated Learning on Non-IID Features via Local Batch Normalization. *ArXiv* abs/2102.07623 (2021).

[44] Zilinghan Li, Shilan He, Ze Yang, Minseok Ryu, Kibaek Kim, and Ravi Madduri. 2024. Advances in Appfl: a Comprehensive and Extensible Federated Learning Framework. *2025 IEEE 25th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)* (2024), 01–11.

[45] Wanchao Liang, Tianyu Liu, Less Wright, Will Constable, Andrew Gu, Chien-Chin Huang, Iris Zhang, Wei Feng, Howard Huang, Junjie Wang, Sanket Purandare, Gokul Nadathur, and Stratos Idreos. 2024. TorchTitan: One-stop PyTorch native solution for production ready LLM pre-training. *ArXiv* abs/2410.06511 (2024). https://api.semanticscholar.org/CorpusID:273228883

[46] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. 2017. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. *ArXiv* abs/1712.01887 (2017).

[47] Yang Liu, Tao Fan, Tianjian Chen, Qian Xu, and Qiang Yang. 2021. FATE: An Industrial Grade Platform for Collaborative Learning With Data Protection. *J. Mach. Learn. Res.* 22 (2021), 226:1–226:6.

[48] Ilya Loshchilov and Frank Hutter. 2017. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

[49] Heiko Ludwig, Nathalie Baracaldo, Gegi Thomas, Yi Zhou, Ali Anwar, Shashank Rajamoni, Yuya Ong, Jayaram Radhakrishnan, Ashish Verma, Mathieu Sinn, et al. 2020. IBM Federated Learning: an Enterprise Framework White Paper V0. 1. *arXiv preprint arXiv:2007.10987* (2020).

[50] H. B. McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2016. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *International Conference on Artificial Intelligence and Statistics*.

[51] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux journal* 2014, 239 (2014), 2.

[52] Alfonso Monaco, Ester Pantaleo, Nicola Amoroso, Antonio Lacalamita, Claudio Lo Giudice, Adriano Fonzino, Bruno Fosso, Ernesto Picardi, Sabina Sonia Tangaro, Graziano Pesole, and Roberto Bellotti. 2021. A primer on machine learning techniques for genomic applications. *Computational and Structural Biotechnology Journal* 19 (2021), 4345 – 4359.

[53] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I. Jordan, and Ion Stoica. 2017. Ray: A Distributed Framework for Emerging AI Applications. *ArXiv* abs/1712.05889 (2017).

[54] NVIDIA. 2023. NCCL: NVIDIA Collective Communications Library. https://github.com/NVIDIA/nccl. Accessed: 2023-10-30.

[55] ORNL. 2025. PETINA: Privacy prEservaTIoN Algorithms. Github. https://github.com/ORNL/PETINA

[56] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, and Trevor Killeen et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *ArXiv* abs/1912.01703 (2019).

[57] Python Software Foundation. 2023. *hashlib — Secure hashes and message digests*. https://docs.python.org/3/library/hashlib.html Accessed: 2023-10-12.

[58] Python Software Foundation. 2023. *hmac — Keyed-Hashing for Message Authentication*. https://docs.python.org/3/library/hmac.html Accessed: 2023-10-12.

[59] PyTorch. 2025. ExecuTorch: On-Device AI for Mobile and Embedded via PyTorch. https://github.com/pytorch/executorch. v0.7.0-release.

[60] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 3505–3506. doi:10.1145/3394486.3406703

[61] G. Anthony Reina, Alexey Gruzdev, Patrick Foley, Olga S. Perepelkina, Mansi Sharma, Igor Davidyuk, Ilya Trushkin, Maksim Radionov, Aleksandr Mokrov, Dmitry Agapov, Jason Martin, Brandon Edwards, Micah J. Sheller, Sarthak Pati, Prakash Narayana Moorthy, Shih-Han Wang, Prashant Shah, and Spyridon Bakas. 2021. OpenFL: the open federated learning library. *Physics in Medicine and Biology* 67 (2021).

[62] Holger R. Roth, Yan Cheng, Yuhong Wen, Isaac Yang, Ziyue Xu, Yuan-Ting Hsieh, Kristopher Kersten, Ahmed El Harouni, Can Zhao, Kevin Lu, Zhihong Zhang, Wenqi Li, Andriy Myronenko, Dong Yang, Sean Bin Yang, Nicola Rieke, Abood Quraini, Chester Chen, Daguang Xu, Nic Ma, Prerna Dogra, Mona G. Flores, and Andrew Feng. 2022. NVIDIA FLARE: Federated Learning from Simulation to Real-World. *ArXiv* abs/2210.13291 (2022).

[63] Minseok Ryu, Youngdae Kim, Kibaek Kim, and Ravi Madduri. 2022. APPFL: Open-Source Software Framework for Privacy-Preserving Federated Learning. *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2022), 1074–1083.

[64] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. 2018. Federated Optimization in Heterogeneous Networks. *arXiv: Learning* (2018).

[65] Anjana M. Samarakoon, David Alan Tennant, Feng Ye, Qiang Zhang, and Santiago A. Grigera. 2022. Integration of machine learning with neutron scattering for the Hamiltonian tuning of spin ice under pressure. *Communications Materials* 3 (2022).

[66] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *ArXiv* abs/1802.05799 (2018).

[67] Shaohuai Shi, Xiaowen Chu, Ka Chun Cheung, and S. See. 2019. Understanding Top-k Sparsification in Distributed Deep Learning. *ArXiv* abs/1911.08772 (2019).

[68] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014).

[69] The MPICH Development Team. 2023. MPICH: High-Performance and Widely Portable MPI. https://www.mpich.org. Accessed: 2023-10-30.

[70] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *The International Journal of High Performance Computing Applications* 19 (2005), 49 – 66.

[71] Sahil Tyagi. 2025. An Overview of Computational and Communication Mechanisms for Scalable AI Systems. https://www.researchgate.net/publication/390663625_An_Overview_of_Computational_and_Communication_Mechanisms_for_Scalable_AI_Systems. Unpublished manuscript.

[72] Sahil Tyagi and Prateek Sharma. 2020. Taming Resource Heterogeneity In Distributed ML Training With Dynamic Batching. *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)* (2020), 188–194.

[73] Sahil Tyagi and Prateek Sharma. 2025. OmniLearn: A Framework for Distributed Deep Learning Over Heterogeneous Clusters. *IEEE Transactions on Parallel and Distributed Systems* 36 (2025), 1253–1267.

[74] Sahil Tyagi and Martin Swany. 2022. ScaDLES: Scalable Deep Learning over Streaming data at the Edge. *2022 IEEE International Conference on Big Data (Big Data)* (2022), 2113–2122.

[75] Sahil Tyagi and Martin Swany. 2023. GraVAC: Adaptive Compression for Communication-Efficient Distributed DL Training. *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)* (2023), 319–329.

[76] Steve Vinoski. 2006. Advanced Message Queuing Protocol. *IEEE Internet Computing* 10 (2006).

[77] Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. 2019. PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization. *ArXiv* abs/1905.13727 (2019).

[78] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. 2020. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. *ArXiv* abs/2007.07481 (2020).

[79] Omry Yadan. 2019. Hydra - A framework for elegantly configuring complex applications. Github. https://github.com/facebookresearch/hydra

[80] Omry Yadan. 2019. OmegaConf. Github. https://github.com/omry/omegaconf

[81] Matei A. Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *Symposium on Networked Systems Design and Implementation*.

[82] Alexander Ziller, Andrew Trask, Antonio Lopardo, Benjamin Szymkow, Bobby Wagner, Emma Bluemke, Jean-Mickael Nounahon, Jonathan Passerat-Palmbach, Kritika Prakash, Nick Rose, Théo Ryffel, Zarreen Naowal Reza, and Georgios Kaissis. 2021. PySyft: A Library for Easy Federated Learning.