Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Communication-Efficient Federated Learning for IoT Edge Computing with LION-Based Optimization

Hamed Amirkhosro
Islamic Azad University
Seyyed Hamid Haji seyyed Javadi
h.s.javadi@shahed.ac.ir
Shahed University
Mohammad Faraji-Mehmandar
P.A.C, Islamic Azad University
Sam Jebbehdari
Islamic Azad University

Research Article

Keywords: Internet of Things, Edge Computing, Federated Learning, Hierarchical Federated Learning, Privacy-Aware Computing, Communication Efficiency, LION Optimizer

Posted Date: November 2nd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11330418/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Communication-Efficient Federated Learning for IoT Edge Computing with LION-Based Optimization

Hamed Amirkhosro[1], Seyyed Hamid Haji Seyyed Javadi [2*], Mohammad Faraji-Mehmandar[3] , Sam Jebbehdari[4]

h_amirkhosro@iau-tnb.ac.ir, h.s.javadi@shahed.ac.ir, mm.faraji@iau.ac.ir, sam.jabbejdari@iau.ac.ir

1 Department of computer engineering, N.T.C., Islamic Azad University, Tehran, Iran

2 Department of computer engineering, Shahed University, Tehran, Iran

3 Department of computer engineering, P.A.C., Islamic Azad University, Parand, Iran

4 Department of computer engineering, N.T.C., Islamic Azad University, Tehran, Iran

Abstract

Training a shared model on Internet of Things (IoT) data presents two practical difficulties: devices have limited computing resources, and repeated exchanges with a remote server consume time and bandwidth. Federated learning keeps the training data on the devices, but it does not remove these costs. We propose HELA-FL, a hierarchical framework that places edge servers between IoT clients and a central server. Clients learn from their local datasets, edge servers coordinate intermediate updates and aggregation, and the central server combines the resulting edge models. The framework uses the LION optimizer to support model optimization within this hierarchy. Its purpose is to make use of the greater computing capacity available at the edge while reducing the frequency of global communication.

We implemented the framework in Python and evaluated it on MNIST with a non-independent and identically distributed (non-IID) allocation of data. The simulation used five edge servers, each serving twenty clients. We compared HELA-FL with FedAvg, FedBuff, AdamW, FedCMFL, FedEdge, FedNova, FedScalar, and FedProx over twenty training rounds. The reported final accuracy was approximately 94%, with latency of about 12 to 12.5 seconds and a combined cost of approximately 220 units under the study's cost definition. Some baselines reached higher accuracy. The value of the proposed approach therefore lies in the balance between accuracy, latency, and cost in this experiment. These results support further investigation of hierarchical aggregation with LION for IoT edge systems; evaluation on a physical testbed remains future work.

Keywords: Internet of Things; Edge Computing; Federated Learning; Hierarchical Federated Learning; Privacy-Aware Computing; Communication Efficiency; LION Optimizer.

1. Introduction

IoT applications in healthcare, buildings, transport, and smart cities generate data across many devices and locations. These data can support classification, prediction, monitoring, and automated decisions. The difficulty is deciding where to process them. Sending every observation to a central data center gives the application access to substantial storage and computing resources, but it also places a continuing load on the network. The time spent transferring data can be significant when an application needs a quick response. Moving sensitive data outside the environment in which they were collected introduces further privacy and security concerns [2].

Edge computing brings processing closer to the data source. A nearby server can handle work that would otherwise travel to the cloud, reducing the distance over which data must be transmitted. Yet proximity alone does not solve the training problem. A network may contain sensors with little memory, battery-powered mobile devices, and edge servers with much greater processing capacity. Bandwidth and connectivity also vary. A learning method intended for this setting must account for these differences as well as the accuracy of its predictions.

Federated learning (FL) allows clients to train a common model without pooling their raw datasets. Each client trains on its own data and periodically sends model parameters or updates to an aggregation server. The server combines these contributions and returns an updated model [5]. This arrangement is useful when data ownership is distributed or local retention is a requirement. It also shifts the communication task: the system now exchanges model information rather than collecting all training records centrally.

Those exchanges can still be expensive. In a conventional FL system, many clients communicate directly with a remote aggregation server, often over numerous training rounds. A large model or a slow connection can make communication a substantial part of training time. Clients also finish their work at different times because their processors, datasets, and network connections differ [3]. A synchronous server must accommodate the slow participants, while asynchronous updates introduce questions about how to use information computed from older model versions.

The data themselves are another source of variation. Individual IoT devices rarely observe identical populations or conditions, so their local datasets are usually non-IID. Their updates may consequently favor different solutions. Differences in dataset size and the amount of local computation further complicate aggregation. These conditions make a flat client-server arrangement a poor fit for some IoT workloads, even when keeping data local is desirable.

We investigate a three-level arrangement in which clients communicate through nearby edge servers. These servers provide intermediate computation and aggregation before models reach the central server. Several local exchanges can then take place within an edge group, with less frequent communication across the wider network. The arrangement also makes greater use of computing resources that are already closer to the clients.

Our framework, HELA-FL (Hierarchical Edge-based Lion Adaptive Federated Learning), combines this hierarchy with LION optimization. The architecture includes IoT clients, edge

servers, and a central server. Raw training data stay with the clients. Edge servers coordinate their associated devices, process intermediate model information, and contribute edge-level models to global aggregation. LION is incorporated to improve the efficiency of the optimization steps, using a sign-based update derived from gradient and momentum information.

The study makes four contributions. First, it organizes local learning, edge aggregation, and global aggregation into separate levels. Second, it introduces LION into the hierarchical learning procedure to investigate its effect on convergence and training cost. Third, it evaluates the method with non-IID client data. Fourth, it compares the framework with eight federated learning and optimization baselines using accuracy, latency, and a combined communication and computation cost. The objective is a useful balance among these measures for devices with limited resources.

Section 2 introduces the background. Section 3 reviews related work and identifies the research question. Sections 4 and 5 describe the architecture and algorithm. Section 6 presents the simulation and results, and Section 7 discusses the conclusions and future work.

2. Background

2-1. Edge Computing

Edge computing distributes processing and storage near the locations where data are produced and used. Moving suitable workloads out of a distant data center can shorten response times and reduce traffic through the core network. The resources at the edge fall into two broad groups: edge devices and edge servers.

Edge devices include smartphones, connected vehicles, sensors, and robots. Their computing capacity is constrained by their hardware, size, or available energy. An edge server has a different role. It supplies shared resources to nearby devices and may take the form of a personal computer, a compact server installation, or a micro data center associated with a router or access point. Devices can offload suitable work to these servers instead of executing everything locally [3].

This division supports applications that need low latency, local data processing, or operation across a wide geographical area. It can also reduce bandwidth use and improve energy efficiency, although the result depends on where tasks are placed and how nodes communicate. Mobility and heterogeneous hardware make coordination part of the problem. Privacy and security likewise depend on the design of the system; placing computation nearby is useful, but proximity is not itself a security mechanism.

Figure 1 shows the main components of an edge computing system. As machine learning has become part of these systems, the term edge intelligence has come to describe the use of AI capabilities within this distributed computing environment.

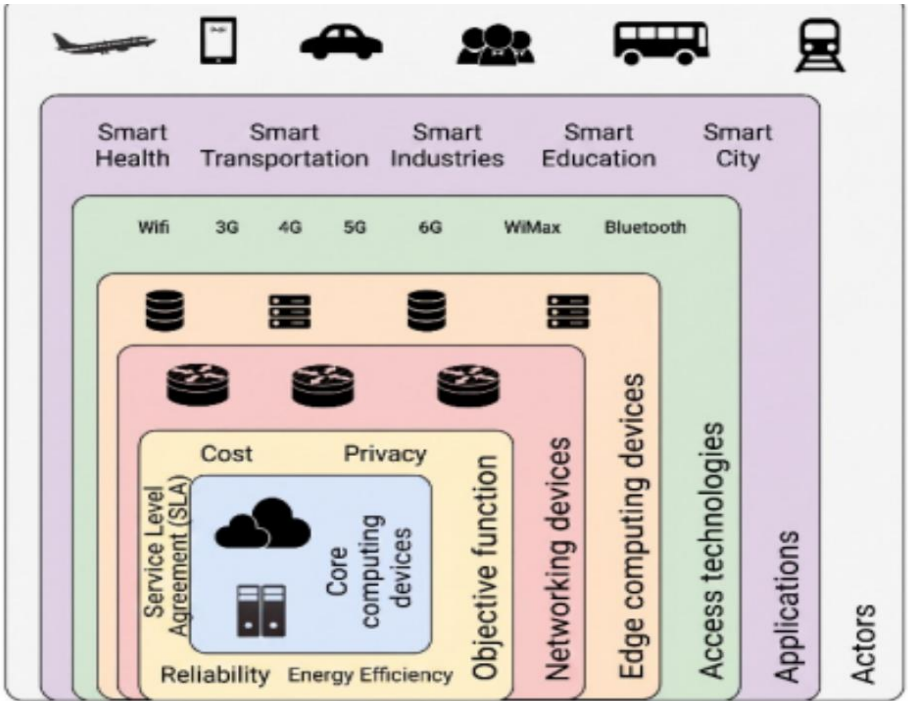


Figure 1. Core components of a typical edge computing architecture.

2-2. Artificial Intelligence at the Edge

AI methods support tasks such as learning, reasoning, perception, planning, and decision-making. Edge computing determines where those tasks can run. Combining the two allows an application to process observations close to users, while distributing work across devices and shared servers. Training and inference need not occur at the same location.

Moving more work toward the edge can reduce the volume of offloaded data and the distance over which it travels. This can lower transmission latency and demand on wide area network bandwidth, while supporting local data retention [1]. There is a trade-off: local resource management becomes more complex, and devices may consume more energy. The appropriate division depends on the application's latency, privacy, bandwidth, and energy requirements.

Other arrangements are possible within this broad division. Devices may train locally while a cloud server aggregates their parameters. Aggregation may instead be assigned to edge servers, or devices may coordinate both learning and aggregation without a cloud server in the training loop. FL is particularly relevant to these arrangements because participants can work with their own datasets rather than upload the underlying records.

Figure 2 distinguishes six levels of edge intelligence [5]. At Level 0, both training and inference take place in the cloud. Level 1 retains cloud training but shares inference between cloud and edge. At Level 2, the cloud still trains the model, while inference is performed collaboratively within the edge network, either by edge nodes or by nearby devices. Level 3 moves inference onto the

individual device after cloud training. Level 4 places both training and inference at the edge [4]. At Level 5, the end device performs both tasks itself.

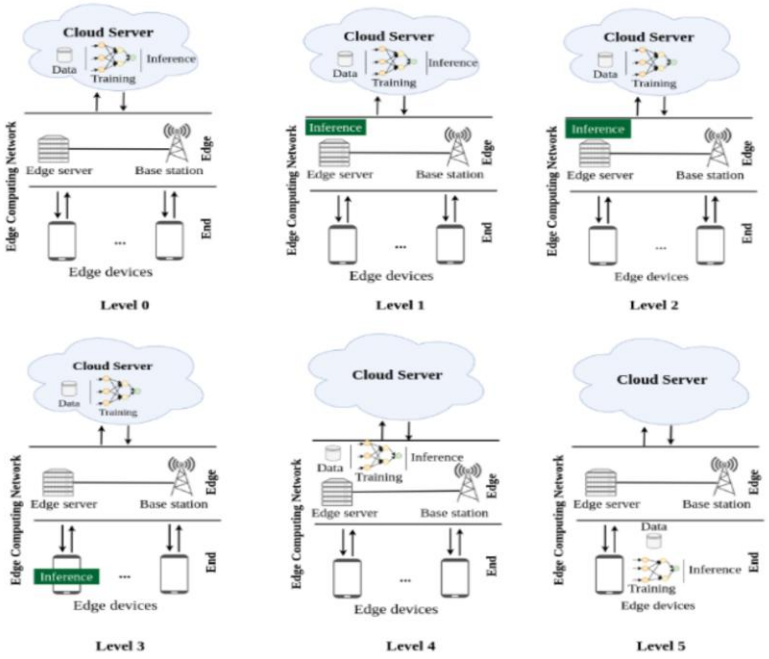


Figure 2. Edge intelligence levels.

2-3. Federated Learning

Federated Learning (FL) trains a shared model across decentralized clients. Instead of transferring their raw datasets to the model owner, clients compute local updates and communicate those updates for aggregation. The approach was introduced in Google's early work on learning from mobile-device data, including next-word prediction [5]. It provides a way to use distributed observations when collecting them centrally is impractical or undesirable.

An FL system has two principal participants: the clients that hold the data and the server that coordinates the model. Figure 3 illustrates their interaction. The server distributes a model, clients train local copies, and the server combines the returned contributions into a new global model. Reliable coordination matters because interrupted or inconsistent exchanges can affect the model used in subsequent rounds.

The communication and computation framework connects these participants. The server normally has more resources than an individual client, whereas each client supplies both data and some processing capacity. Local computation can occur in parallel. A client can also retain a model for inference, reducing its dependence on remote requests. Whether these choices shorten total training time depends on communication, synchronization, and the capabilities of the participants.

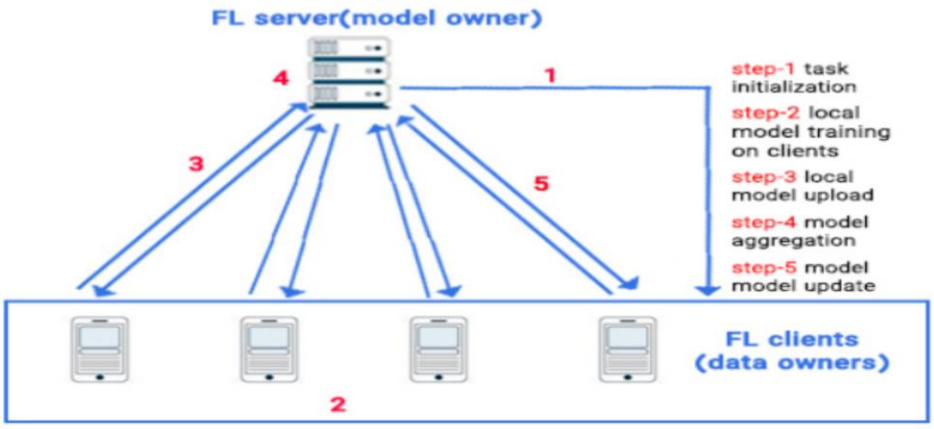


Figure 3. System architecture and workflow of federated learning.

Keeping raw data on the client reduces the need to disclose the underlying records. It should not be treated as a complete privacy guarantee, because model information is still exchanged. Privacy-preserving collaborative intelligence is therefore a wider design question than data locality alone [6]. FL has been studied or used in mobile applications, smart homes and cities, industrial monitoring, healthcare, financial services, and transportation. These applications differ considerably in their tolerance for delay and in the sensitivity and distribution of their data.

2-4. Lion Optimizer

A learning algorithm adjusts model parameters to reduce a loss function. For a neural network, the parameters include its weights and biases. Training usually begins from an initial parameter setting, computes predictions on a batch of data, measures their error, and obtains gradients through backpropagation [3]. A basic gradient-descent update is:

$$New\ weight = Old\ weight - (Learning\ rate \times Gradient)$$

The learning rate determines the size of each step. A positive gradient moves the corresponding weight downward under this rule, and a negative gradient moves it upward. Repeating the process changes the model's predictions. An epoch is one pass through the training set; an optimization step normally operates on a batch within that pass.

In FL, clients compute their updates using different local datasets. These updates must be combined despite data heterogeneity and limited opportunities for communication. Researchers have consequently examined several optimization methods, including adaptive optimizers such as Adam [8].

LION, introduced through symbolic discovery of optimization algorithms [28], uses the sign of an update formed from gradient and momentum information. Its update does not use the full magnitude of that combined direction. LION maintains momentum state but avoids the separate second-moment estimate used by Adam. This reduces optimizer-state memory and makes it a candidate for resource-constrained training. Its reported performance in deep learning motivates

examining it in an edge-based FL framework; its effectiveness in that setting still has to be established through the evaluation.

3. Related Works

Work on FL addresses several different sources of inefficiency. Statistical heterogeneity concerns the differences between local datasets; system heterogeneity concerns the resources and availability of clients. Communication volume, synchronization, and participation introduce additional costs. FedAvg [5] remains a common reference point, but its basic client-server arrangement does not resolve all of these issues.

We organize the literature into five groups: changes to the learning algorithm, optimizer-based methods, aggregation strategies, client selection, and communication-efficient methods. Table 1 summarizes the mechanisms considered in each group. The categories overlap, since a method can change both the optimization process and the amount of communication it requires.

Table 1. Summary of Approaches for Improving Federated Learning

Approach	Policy	Description
Learning Algorithm	Algorithm enhancement	Modify the objective, client weights, or learning rate to accommodate heterogeneity.
	Personalized learning	Assign client-specific learning tasks according to differences in local data.
Optimization	Adaptive federated optimization	Use adaptive server updates, client-drift correction, or normalized local updates.
	Lightweight and advanced optimizers	Use optimizers such as LION to investigate convergence and optimizer-state memory costs.
Aggregation	Asynchronous	Aggregate when an individual local update reaches the server.
	Semi-asynchronous	Aggregate after a subset of clients has supplied updates.
	Semi-synchronous	Coordinate aggregation while clients continue local training.
Client Selection	System capabilities	Select participants according to their resources and current system conditions.
	Dataset quality	Use the quality of local data when choosing participants.
Communication	Fewer exchanges	Reduce the frequency of model transmissions to the central server.
	Smaller transfers	Apply quantization, pruning, or compression to reduce the information sent per round.

3.1. Improving federated learning algorithms

Local datasets can differ in size, label distribution, and other statistical properties. Devices also differ in memory, processing speed, connectivity, and energy availability. As a result, clients may perform different amounts of work and produce updates that do not move toward the same solution.

Algorithm-level methods try to accommodate these differences without losing the benefit of collaborative training.

FedProx [9] extends FedAvg for heterogeneous networks. Its treatment of local computation allows clients to perform different amounts of work according to their available resources. This flexibility is useful when a fixed workload would exclude slow devices or delay every round. The method is intended to make federated optimization more tolerant of the differences among participants.

AdpFedPSO [10] uses particle swarm optimization to adjust clients' aggregation weights. The weighting takes client performance and consistency into account, rather than assigning contributions without regard to their behavior. The approach addresses data heterogeneity together with communication-related costs, although dynamic optimization adds computation of its own.

FedAF [11] takes a different route. Clients provide compressed synthetic data and soft labels instead of complete local models for direct averaging. The server uses this information to build the global model. This is intended to limit the effects of client drift while retaining knowledge acquired earlier. The quality and cost of generating the synthetic data are therefore central to the method.

Fairness motivates FedGraphRL [12]. It represents relationships among clients as a collaboration graph and uses an adaptive graph convolutional network within reinforcement learning to determine aggregation weights. The objective includes both overall performance and the distribution of that performance across clients. Such graph-based weighting also depends on the quality of the relationships represented in the graph.

These methods alter local learning, client weighting, or the information supplied for global learning. Our question concerns an additional design choice: where the intermediate work should occur. HELA-FL introduces edge servers as aggregation and computation nodes between resource-constrained clients and the remote server.

3.2. Optimizer-based approaches

FedAvg commonly combines local stochastic gradient descent with server aggregation. This is straightforward, but heterogeneous data and local workloads can affect convergence. Optimizer-based methods change how local or aggregated updates are used, with the aim of reaching a useful model in fewer rounds or with more stable training.

Reddi et al. [24] introduced the FedOpt framework for adaptive server-side optimization. Its federated versions of Adagrad, Adam, and Yogi are known as FedAdagrad, FedAdam, and FedYogi. Rather than applying the aggregated client update directly, the server uses it within an adaptive optimization step. Their analysis and experiments address heterogeneous, non-convex settings and show why server optimization is a separate design choice from local training.

SCAFFOLD [25] addresses client drift with control variates. These variables correct the direction of local updates when clients train on non-IID data. The correction can improve convergence and reduce communication rounds, but it requires additional state and management of the control

variables. For small devices, this overhead must be considered alongside any reduction in training time.

FedNova [26] examines objective inconsistency caused by unequal local computation. If clients take different numbers of local steps, simply averaging their updates can move the global model toward an objective other than the intended one. FedNova normalizes the updates according to the local work performed before aggregation. This provides a way to account for unequal workloads rather than treating their outputs as directly comparable.

Primal-dual methods offer another formulation. FedADMM [27] applies the alternating direction method of multipliers to distributed learning, introducing dual variables as part of the optimization process. It addresses forms of statistical and system heterogeneity, including variable local computation. The additional variables and operations also make implementation more involved, which matters when the available resources are limited.

LION [28] was developed for neural-network optimization through symbolic search. It uses the sign of a momentum-related direction and requires less optimizer state than Adam. This memory requirement makes it relevant to the present study, although it does not remove the need to tune learning rates or evaluate performance for the chosen model and data.

Tang and Chang's FedLion [29] brings LION-based optimization into FL. The method considers convergence and uplink communication, using signed information to reduce what clients need to transmit. It is particularly relevant here because LION in a federated setting is already established prior work. The research question for HELA-FL is therefore the use of LION within a client-edge-cloud hierarchy, including the intermediate computation and aggregation performed at edge servers.

An optimizer and a communication topology address related but distinct costs. Faster optimization may reduce the number of rounds, while intermediate aggregation changes which nodes exchange information during those rounds. HELA-FL combines these choices and evaluates their resulting accuracy, latency, and combined cost. The comparison is concerned with that complete arrangement rather than a claim that LION itself is new.

3.3. Changing the aggregation strategy

A synchronous server waits for participating clients before constructing the next global model. When devices differ substantially, a few slow clients can determine the duration of a round. Aggregation strategies vary in how much waiting they permit and how they handle updates computed from different model versions.

FedAsync [13] updates the global model as client results arrive, using weighted asynchronous aggregation. It accounts for staleness because some arriving updates were computed from older parameters. Removing the requirement to wait for every client can improve responsiveness, but it also makes update timing part of the optimization problem.

FedBuff [14] buffers incoming updates and aggregates them once the buffer contains enough contributions. Clients can follow independent schedules, while the server still combines a group of updates at each aggregation step. This provides an intermediate approach between waiting for all clients and updating the model after every individual arrival.

Semi-synchronous FL, including SSFL [15], allows local training to continue while synchronization takes place at prescribed points. The aim is to reduce idle time while retaining coordinated global updates. Each of these approaches changes when aggregation occurs.

HELA-FL changes where some aggregation occurs. Nearby clients report to an edge server, and edge-level results are then sent to the central server. Adjusting aggregation timing and introducing this hierarchy are different ways to address delay; either must be assessed under the network and device conditions for which it is intended.

3.4. Client selection

Large FL systems need not involve every available client in every round. Selecting participants can reduce computation and communication, but selection also determines which data influence the model. A selection policy may consider data quality, available resources, reliability, past participation, or a client's likely contribution.

Wan et al. [16] use deep deterministic policy gradient (DDPG) for security-aware and fair client selection. Their assessment considers historical behavior for security and participation frequency for fairness. The method therefore treats the selection decision as more than a search for the fastest devices.

Marfo et al. [17] examine adaptive selection with differential privacy and fault tolerance. The number of participants changes with model performance and system constraints. Calibrated noise provides the privacy mechanism, while client scoring supports resilience to failures. These mechanisms introduce their own communication and computation requirements.

FedGCS [18] formulates selection through a learned representation and gradient-based optimization. Its process includes collecting decision data, training the selection model, optimizing within the learned representation, and generating a participation strategy. The effectiveness of the resulting policy depends on this training process and its parameters.

Selection reduces the number of contributions that must be processed, but aggressive exclusion may also reduce data diversity. HELA-FL instead concentrates on coordinating clients through nearby edge servers. It does not rely primarily on excluding more devices to reduce traffic to the central server. Selection and hierarchical aggregation could be combined, but adaptive selection is left for future work in the present framework.

3.5. Communication-efficient federated learning

Repeated transmission of a large parameter vector can dominate the cost of FL. Communication-efficient methods generally reduce either the frequency of exchanges or the amount of information

carried by each exchange. These choices affect reconstruction error, convergence, and, in some methods, privacy.

CEPAM [19] combines randomized vector quantization with an adjustable privacy mechanism. Its parameters control the balance among communication, quantization distortion, privacy, and accuracy. FedScalar [20] reduces an update to a scalar using a shared random vector. The server reconstructs an aggregate direction from these scalar contributions and the shared projection mechanism, sharply reducing uplink volume.

FedACG [21] uses look-ahead gradient information in the model sent to clients to address slow or unstable convergence under heterogeneous data and limited participation. It supplies information about the update direction within the existing exchange. FedSpar [22] reduces uplink traffic through sparsification: clients send selected high-magnitude elements and their positions, with quantization and reconstruction used to control compression error.

These methods modify what is transmitted or how often transmission is needed. Intermediate edge aggregation offers another design choice by changing the path between clients and the central server. Table 2 brings together the approaches reviewed above, their evaluation focus, and the advantages and limitations identified in the comparison.

Table 2. Comparison of Studies on Federated Learning

Reference (Year)	Proposed Approach Type	Proposed Algorithm	Evaluation Criteria	Advantages	Disadvantages
Tian Li et al. (2020)	FL Algorithm Enhancement	FedProx	Reduction in training time and communication cost, and reduction in the number of client rounds	Improved utilization of data heterogeneity; suitable for heterogeneous systems	High convergence speed; limited scalability
Ouyang et al. (2024)	FL Algorithm Enhancement	AdpFedPSO	Aggregation of weights based on system status and client performance	Reduction of data heterogeneity and increased communication speed	High computational complexity; requires optimization of model parameters
Wang et al. (2024)	Modification of Aggregation Method	FedFA	No aggregation of local models; use of compressed and synthetic data	Reduction of data heterogeneity and increased convergence speed	Complexity of data generation; dependency on the generated data
Yujia Xia et al. (2024)	Modification of Aggregation Method	FedGraphRL	Learning based on graph structure and client relationships	Reduction of data heterogeneity; suitable for medical applications	Dependence on graph quality; high computational complexity
Xie et al. (2020)	Modification of Aggregation Method – Asynchronous Aggregation	FedAsync	Reduction in training time through asynchronous aggregation and elimination of waiting for slow clients	Reduction of waiting time; improved system responsiveness	Increased system complexity; potential instability in training

Reference (Year)	Proposed Approach Type	Proposed Algorithm	Evaluation Criteria	Advantages	Disadvantages
John Nguyen et al. (2022)	Modification of Aggregation Method – Semi-Asynchronous Aggregation	FedBuff	Faster training via aggregation once a subset of clients has reported updates	Reduction of waiting time; suitable for large-scale systems	System heterogeneity; limited scalability
Dimitris Stripelis et al. (2022)	Modification of Aggregation Method – Semi-Synchronous Aggregation	SSFL	Reduction in training time through periodic aggregation of local models	Reduction of waiting time; suitable for heterogeneous systems	System heterogeneity; limited scalability
Reddi et al. (2021)	FL Algorithm / Optimizer Enhancement	FedOpt (FedAdagrad, FedAdam, FedYogi)	Adaptive server-side optimization; convergence under heterogeneous and non-IID settings	Improved convergence and flexibility compared with conventional FedAvg; supports multiple adaptive optimizers	Additional optimizer states and server-side tuning; increased optimization complexity
Karimireddy et al. (2020)	FL Algorithm / Optimizer Enhancement	SCAFFOLD	Reduction of client drift and communication rounds under non-IID data	Corrects client drift using control variates; improved convergence under data heterogeneity	Requires additional control variables and communication/state management
Wang et al. (2020)	FL Algorithm / Optimizer Enhancement	FedNova	Objective consistency and convergence under heterogeneous local computation	Normalizes local updates and addresses objective inconsistency	Additional normalization mechanism; performance depends on heterogeneous training conditions
Gong et al. (2022)	FL Algorithm / Optimizer Enhancement	FedADMM	Communication efficiency and robustness to statistical and system heterogeneity	Primal-dual optimization; supports variable local computation and strong convergence guarantees	Higher algorithmic complexity due to dual variables and augmented-Lagrangian optimization
Chen et al. (2023)	Optimizer-Based Approach	LION	Convergence, training efficiency, and optimizer-state memory	Memory-efficient sign-based updates; competitive training performance across deep-learning tasks	Requires careful learning-rate tuning; effectiveness can vary across tasks and architectures
Tang and Chang (2024)	FL Optimizer / Communication Enhancement	FedLion	Convergence speed and uplink communication cost	Combines Lion with federated optimization; signed updates can reduce uplink transmission requirements	Focuses primarily on federated optimization and does not by itself provide hierarchical edge aggregation
William Marfo et al. (2025)	Client Selection	CSDPFT	Increasing privacy and fault tolerance in client selection	Increased privacy; improved system reliability	Increased computational overhead; increased communication cost
Zhiyuan Ning et al. (2024)	Client Selection	FedGCS	Reduction of training time and energy consumption	Data and system heterogeneity; simple client selection mechanism; reduced energy consumption	Complexity of the training process; sensitivity to model parameters

Reference (Year)	Proposed Approach Type	Proposed Algorithm	Evaluation Criteria	Advantages	Disadvantages
Qi Weiling et al. (2024)	Communication Mechanism Improvement	CEPAM	Use of randomized vector quantization for privacy protection	Reduced communication cost; improved privacy protection	Increased noise; trade-off between privacy and accuracy
Mohammad Reza Rostami et al. (2024)	Communication Mechanism Improvement	FedScalar	Encoding updates into a scalar using a random vector	Reduced communication cost; increased compression efficiency	Increased reconstruction complexity; dependence on the random vector
Gihyeok Kim et al. (2024)	Communication Mechanism Improvement	FedACG	Use of a look-ahead gradient to improve communication efficiency	Reduced communication overhead; increased robustness to client heterogeneity	Increased computational complexity; dependence on gradient information
Ji-yeon Yoon et al. (2025)	Communication Mechanism Improvement	FedSpar	Top-S sparsification and optimized quantization	Reduced communication cost; increased communication efficiency	Lack of support for all types of data; increased resource requirements

3.6. Discussion and research gap

The studies reviewed here address different parts of the same training process. FedProx and adaptive weighting methods accommodate heterogeneous participants. FedAsync, FedBuff, and SSFL change synchronization. Selection methods control participation. CEPAM, FedScalar, FedACG, and FedSpar address the amount of information exchanged or the number of exchanges needed. FedOpt, SCAFFOLD, FedNova, FedADMM, and FedLion examine how model updates are formed and combined.

For IoT systems, these decisions must also account for the location of computing resources. Compressing a model update does not by itself reduce the work needed to produce it on a small device. Likewise, changing the aggregation schedule does not necessarily shorten the path to a remote server. We therefore investigate a hierarchy that uses edge servers for intermediate computation and aggregation while retaining raw data at the clients.

HELA-FL couples this client-edge-cloud arrangement with LION-based optimization. The intended benefit comes from organizing work around the capacities and locations of the nodes. Our experiments examine whether the resulting system offers a useful accuracy-cost-latency trade-off under heterogeneous client data. This is the scope of the contribution; it does not establish that a single arrangement is best for every federated workload.

4. Proposed Framework

HELA-FL separates learning within an edge group from global aggregation across edge groups. IoT devices retain their local data and communicate with an assigned edge server. Each edge server coordinates its clients and passes an edge-level model to the central server. The three levels are shown in Figure 5; Figure 4 provides the conventional FedAvg arrangement for comparison.

In FedAvg, the central server distributes the current global model to selected clients. Each client trains locally for several epochs, commonly using SGD, and sends the updated parameters back for aggregation. The server constructs the next global model and repeats the cycle [23]. In an IoT deployment, smartphones, watches, laptops, and cameras may all act as clients despite having much less computing capacity than the server.



Figure 4. The structure of the standard FedAvg algorithm.

The proposed architecture assigns more of the work to nearby edge servers. Its local-update design divides work between devices and servers: devices handle the lower model layers, while more demanding processing is offloaded to the edge. Intermediate aggregation allows local work to proceed without contacting the central server after every update. LION is used to support convergence within this arrangement.



Figure 5. The architecture of the proposed HELA-FL framework.

The client level consists of end-user devices such as smartphones, cameras, smartwatches, augmented- and virtual-reality headsets, vehicle platforms, and smart glasses. They collect

observations and can perform initial processing. Their available energy and processing power vary, and the data held by different devices may be both unequal in size and non-IID.

The edge-server level provides shared resources within the local network. An edge server may be a router with computing capacity, a personal computer, a micro-base station, or a regional micro-cloud. Compared with client devices, these nodes generally offer more storage, processing power, and a stable power supply. Their proximity can support shorter, faster communication paths, although actual performance depends on the network.

The central server sits above the edge groups and combines their models. It can use the resources of a cloud-provider cluster, but communication to this level crosses a wider network. Congestion, bandwidth variation, and distance may affect those exchanges. The framework therefore reserves global aggregation for the end of a sequence of local training steps.

A training iteration has two phases. The first involves local work between clients and their associated edge servers. The second aggregates edge-level results at the central server. Initially, the central server distributes model parameters to the edge servers, and clients obtain the parameters from their assigned server. Devices compute with their local data, edge servers process the resulting model information, and refreshed parameters are returned as the local process proceeds.

The intended advantages are a lighter training burden on IoT devices, fewer global exchanges, and efficient optimization with LION. Its sign-based parameter step uses both the current gradient and momentum information. It retains a momentum buffer, so the memory advantage over Adam comes from avoiding an additional second-moment buffer. Whether this advantage produces lower end-to-end cost depends on the rest of the learning and communication procedure; Section 6 reports the simulation used to assess that question.

5. Proposed Algorithm

The training procedure begins when the central server initializes and distributes the global model. Clients obtain the model through their edge server and train on their local data in parallel. Edge servers collect the resulting local models and apply the edge aggregation and optimization procedure. After the configured local epochs, they send their edge models to the central server for global aggregation.

The central server evaluates the aggregated model against the cloud-held evaluation dataset. Training stops when the specified performance threshold is reached or the maximum number of rounds has been completed. Otherwise, the updated global parameters are sent back to the edge servers for another round. This stopping rule limits unnecessary training after the target is met; it does not, by itself, establish protection against overfitting.

Algorithm 1 expresses the procedure. Let M be the number of edge servers, K the number of clients assigned to each server, E the number of local epochs, b the batch size, and η the learning rate.

The client loop applies LION to the local model. The edge step is represented by Aggregate_LION, followed by aggregation at the cloud server.

Algorithm 1. HELA-FL with LION

```
Input: M edge servers; K clients per server; E local epochs;
       batch size b; learning rate eta; maximum rounds T_max;
       convergence threshold Threshold
Output: Global_Model

Initialize Global_Model at the cloud server
For t = 1 to T_max:
    Send Global_Model to all M edge servers
    In parallel, for each edge server j:
        Edge_Model_j = receive(Global_Model)
        Local_Updates_List = []
        In parallel, for each client k assigned to server j:
            Client_Model_k = download(Edge_Model_j)
            For e = 1 to E:
                Split Client_Data_k into batches of size b
                For each batch:
                    Gradient = compute_gradient(Client_Model_k, batch)
                    Client_Model_k = LION_Optimizer_Update(
                        Client_Model_k, Gradient, learning_rate=eta)
            Send Client_Model_k to edge server j
            Append Client_Model_k to Local_Updates_List
        Aggregated_Edge_Model_j = Aggregate_LION(Local_Updates_List)
        Send Aggregated_Edge_Model_j to the cloud server
    Receive Aggregated_Edge_Models from all M edge servers
    Global_Model = Average(Aggregated_Edge_Models)
    Performance = Evaluate(Global_Model, Cloud_Test_Dataset)
    If Performance >= Threshold:
        Stop training
Return Global_Model
```

6. Implementation and Performance Evaluation

We implemented the simulation in Python using PyTorch for model construction and training and NumPy for numerical operations. Model updates used a custom LION optimizer. The dataset was MNIST, comprising 60,000 training images and 10,000 test images. The data were downloaded when a local copy was unavailable.

Preprocessing converted the images to tensors with pixel values in the interval [0, 1]. We then normalized the values using the MNIST mean, $\mu = 0.1307$, and standard deviation, $\sigma = 0.3081$:

$$normalized_x = (x - \mu) / \sigma$$

Here x denotes a pixel value after rescaling. We also introduced label noise into the training set to represent errors that can occur in supervised datasets.

The classifier was a compact PyTorch model named SimpleCNN. It had two convolutional layers with pooling operations and two fully connected layers for digit classification. The division shown in the implementation description is:

$$Middle\ Output \Rightarrow\ Conv1 + ReLU \rightarrow MaxPool \rightarrow Conv2 + ReLU \rightarrow x$$

$$Final\ Prediction \Rightarrow\ Flatten \rightarrow MaxPool \rightarrow FC1 + ReLU \rightarrow FC2 \rightarrow x$$

The LION configuration used a learning rate of 0.001, also written as 10−3, and a weight-decay coefficient of 10−4. The optimizer operates on the learnable model parameters, including weights, and uses two exponential averaging coefficients, β1 and β2. The parameter update applies the sign of a momentum-related direction. The update expressions are:

$$m_t = \beta_1 \cdot m_{t-1} + g_t \cdot (\beta_1 - 1)$$

$$\Theta = \Theta \cdot (weight_decay \cdot \eta - 1)$$

$$\Theta_t = \Theta_{t-1} - \eta \cdot sign(m_t)$$

$$m_t = \beta_2 \cdot m_t + (\beta_2 - 1) \cdot g_t$$

In this notation, gt is the current gradient, mt is the momentum quantity, Θ represents model parameters, and η is the learning rate. The loss used for classification is cross-entropy:

$$L = Cross\ Entropy\ (output, target)$$

Training time was recorded for the local models. Their parameters were then aggregated to update the shared model. The averaging relationship is expressed as:

$$\bar{w}_t = w_i^{(j)} \cdot \Sigma_{j=1}^{n}\ (1 / N)$$

Here wi(j) denotes parameter i in model j, and N is the number of contributing models. The training loop repeats local learning, aggregation, and global model updates while recording accuracy and cost. The number of global rounds is configurable; the reported comparison used twenty rounds.

6.3 Cost measures

We counted communication in bytes. A client receives a model and returns an updated copy, giving two transfers of size S. An edge server likewise sends and receives a model when it communicates with the cloud. For N participating clients and M edge servers, we used:

$$Communication\ cost = 2SM + 2SN$$

S is the model size in bytes. The term 2SN counts the client-edge transfers, and 2SM counts the edge-cloud transfers.

For computation, we added the time spent training at the individual clients:

$$Computation\ Cost = \Sigma_{i=1}^{N} T_i$$

Ti denotes the training time for client i. We then combined these two measurements:

$$Total\ Cost = \alpha \cdot Communication\ Cost + \beta \cdot Computation\ Cost$$

The coefficients were $\alpha = 10{-}6$ and $\beta = 1$. Thus, communication was scaled before being added to the training-time sum. The cost values below refer to this calculation and have no monetary interpretation.

6.4 Comparative results

Figure 7 presents the twenty-round MNIST comparison. The eight baselines were FedAvg, FedBuff, AdamW, FedCMFL, FedEdge, FedNova, FedScalar, and FedProx. Accuracy, total cost, delay, and test loss are shown separately.

HELA-FL gained accuracy in the early rounds and finished at approximately 94%. This was below the roughly 98% reached by FedCMFL and FedScalar. Several other methods ended near 84%. The accuracy curve therefore places our method between these groups; the experiment does not show an accuracy advantage over the best-performing baselines.

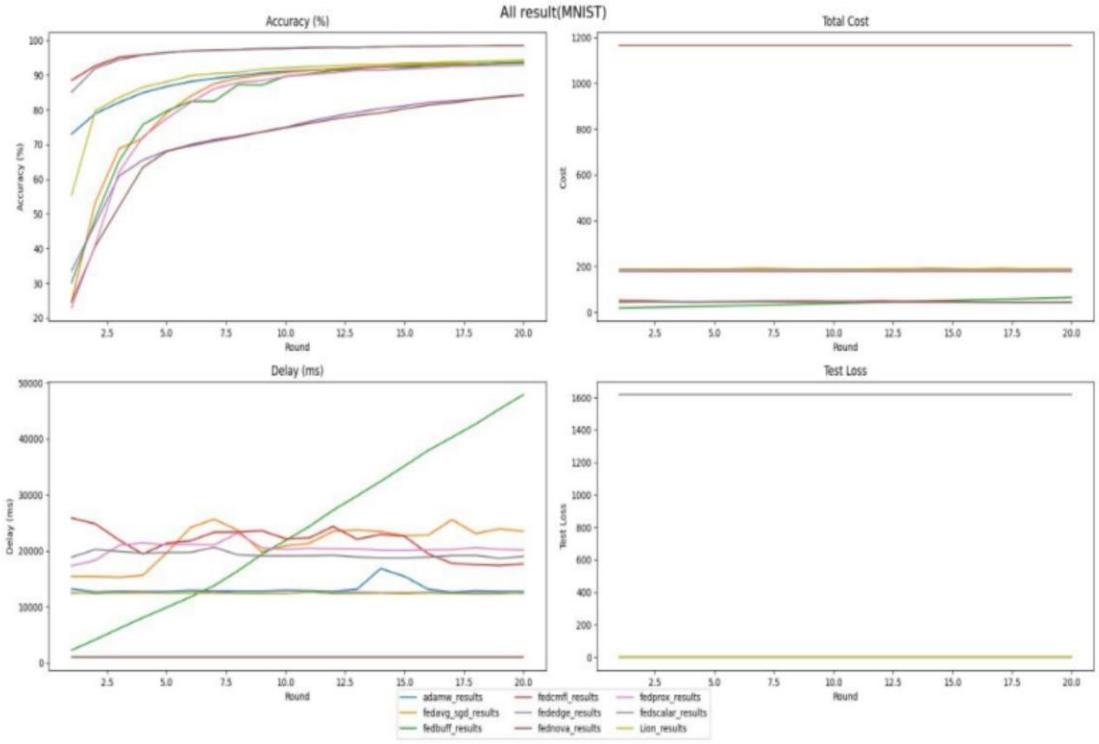


Figure 7. Accuracy, cost, delay, and test-loss comparison between the proposed algorithm and eight baseline methods.

The delay comparison gives a different view. At the end of training, HELA-FL was at about 12 to 12.5 seconds. Figure 7 uses milliseconds, so this corresponds to 12,000 to 12,500 on that axis. Some methods were around 20 to 25 seconds, and FedBuff reached approximately 47 to 48 seconds. The hierarchy was designed to limit expensive global exchanges, and the measured delay is consistent with that purpose. A separate ablation would be needed to distinguish the effect of the hierarchy from the effect of the optimizer.

The final combined cost reported for HELA-FL was approximately 220 units, using the weights in Section 6.3. Both communication volume and client training time enter this value. Changing their relative weights would change the cost comparison, even if the underlying training measurements stayed the same.

On the test-loss plot, HELA-FL and most of the other curves are close to zero at the displayed scale. Small differences are difficult to read from this panel. We therefore consider it alongside the accuracy plot rather than infer equal predictive performance from the apparent overlap.

For this run, the useful result is the combination of approximately 94% accuracy, low delay relative to several baselines, and the reported composite cost. These measurements concern the simulated non-IID MNIST setting. They do not establish performance on a deployed IoT network.

19

7- Conclusion and Future Work

The motivation for HELA-FL was to use the computing capacity between an IoT device and the cloud. In the proposed three-level architecture, clients retain their data, edge servers coordinate intermediate learning and aggregation, and the central server combines the edge models. We incorporated LION into this arrangement and evaluated it against eight baselines.

The twenty-round MNIST experiment gives a reason to investigate this design further. It reached approximately 94% accuracy with a final delay of about 12 to 12.5 seconds and a reported cost of approximately 220 units. Higher accuracy was obtained by some competing methods, so the benefit depends on how much weight the application places on delay and cost.

A next step would be evaluation under less controlled network conditions, with clients joining and leaving, variable bandwidth, and unequal processing capacity. A physical edge or fog testbed would show whether the simulated gains persist. Adaptive selection and resource allocation also warrant examination as the system grows, together with comparisons of LION and other optimizers within the same hierarchy. Finally, keeping raw data local should be supplemented with explicit privacy mechanisms. Differential privacy and secure aggregation are candidates for that work.

References

[1] Adam, M., et al., Survey of Multimodal Federated Learning: Exploring Data Integration, Challenges, and Future Directions. IEEE Open Journal of the Communications Society, 2025. 6: p. 2510–2538.

[2] Baduwal, M., P. Paudel, and V. Chaudhary, Federated Learning: A Survey of Core Challenges, Current Methods, and Opportunities, in Preprints. 2026, Preprints.

[3] Saeed, N., M. Ashour, and M. Mashaly, Comprehensive review of federated learning challenges: a data preparation viewpoint. Journal of Big Data, 2025. 12(1): p. 153.

[4] Cooray, L., et al., Deep federated learning: a systematic review of methods, applications, and challenges. Frontiers in Computer Science, 2025. Volume 7 - 2025.

[5] McMahan, B., et al. Communication-efficient learning of deep networks from decentralized data. in Artificial intelligence and statistics. 2017. Pmlr.

[6] Rahman, R., Federated learning: A survey on privacy-preserving collaborative intelligence. arXiv preprint arXiv:2504.17703, 2025.

[7] Chen, X., et al., Symbolic discovery of optimization algorithms. Advances in Neural Information Processing Systems, 2023. 36: p. 49205–49233.

[8] Kingma, D.P. and J. Ba, Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[9] Li, T., et al., Federated optimization in heterogeneous networks. Proceedings of Machine learning and systems, 2020. 2: p. 429–450.

[10] Ouyang, C. et al., Enhancing federated learning with dynamic weight adjustment based on particle swarm optimization. Discover Computing, 2024. 27(1): p. 35.

[11] Wang, Y., et al. An Aggregation-Free Federated Learning for Tackling Data Heterogeneity. in 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2024.

[12] Xia, Y., et al. Enhancing Federated Learning Performance Fairness via Collaboration Graph-Based Reinforcement Learning. in Medical Image Computing and Computer Assisted Intervention – MICCAI 2024. 2024. Cham: Springer Nature Switzerland.

[13] Xie, C., S. Koyejo, and I. Gupta, Asynchronous federated optimization. arXiv preprint arXiv:1903.03934, 2019.

[14] Nguyen, J., et al. Federated learning with buffered asynchronous aggregation. in International conference on artificial intelligence and statistics. 2022. PMLR.

[15] Stripelis, D., P.M. Thompson, and J.L. Ambite, Semi-Synchronous Federated Learning for Energy-Efficient Training and Accelerated Convergence in Cross-Silo Settings. ACM Trans. Intell. Syst. Technol., 2022. 13(5): p. Article 78.

[16] Wan, T., et al., A Secure and Fair Client Selection Based on DDPG for Federated Learning. International Journal of Intelligent Systems, 2024. 2024(1): p. 2314019.

[17] Marfo, W., D.K. Tosh, and S.V. Moore. Adaptive Client Selection in Federated Learning: A Network Anomaly Detection Use Case. in 2025 International Conference on Computing, Networking and Communications (ICNC). 2025.

[18] Ning, Z., et al., FedGCS: a generative framework for efficient client selection in federated learning via gradient-based optimization, in Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence. 2024: Jeju, Korea. p. Article 526.

[19] Ling, C.W., et al., Communication-Efficient and Privacy-Adaptable Mechanism for Federated Learning. arXiv preprint arXiv:2501.12046, 2025.

[20] Rostami, M. and S. Kia, FedScalar: Federated Learning with Scalar Communication for Bandwidth-Constrained Networks. arXiv preprint arXiv:2410.02260, 2024.

[21] Kim, G., J. Kim, and B. Han. Communication-Efficient Federated Learning with Accelerated Client Gradient. in 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2024.

[22] Yun, J., et al., Communication-Efficient Federated Learning Over Capacity-Limited Wireless Networks. IEEE Transactions on Cognitive Communications and Networking, 2025. 11(1): p. 621–637.

[23] Talaei, M. and I. Izadi, Enhancing federated learning with adaptive differential privacy and priority-based aggregation. arXiv preprint arXiv:2406.18491, 2024.

[24] Reddi, S.J., et al., Adaptive Federated Optimization. International Conference on Learning Representations (ICLR), 2021.

[25] Karimireddy, S.P., et al., SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. International Conference on Machine Learning (ICML), 2020, 119: p. 5132–5143.

[26] Wang, J., et al., Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. Advances in Neural Information Processing Systems (NeurIPS), 2020, 33.

[27] Gong, Y., Li, Y., and Freris, N.M., FedADMM: A Robust Federated Deep Learning Framework with Adaptivity to System Heterogeneity. arXiv preprint arXiv:2204.03529, 2022.

[28] Chen, X., et al., Symbolic Discovery of Optimization Algorithms. Advances in Neural Information Processing Systems (NeurIPS), 2023, 36: p. 49205–49233.

[29] Tang, Z. and Chang, T.-H., FedLion: Faster Adaptive Federated Optimization with Fewer Communication. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2024, pp. 13316–13320. DOI: 10.1109/ICASSP48485.2024.10447045.