

Article

Communication-Computation Co-Optimized Federated Learning for Efficient Large-Model Embedding Training

Yingying Luo ^{1,2,3}, Xi Jin ^{4,*}, Changqing Xia ^{1,2} , Chi Xu ^{1,2}  and Yiming Sun ^{1,2} ¹ Key Laboratory of Networked Control Systems, Chinese Academy of Sciences, Shenyang 110016, China; luoyingying@sia.cn (Y.L.)² Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China³ University of Chinese Academy of Sciences, Beijing 100049, China⁴ School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China

* Correspondence: jinxi@neu.edu.cn

Abstract

With the rapid development of the Industrial Internet of Things (IIoT) and intelligent manufacturing, massive amounts of heterogeneous and non-independent, identically distributed (non-IID) data are continuously generated in industrial environments. Large models have demonstrated strong generalization and transfer capabilities, offering new possibilities for predictive maintenance, anomaly detection, and intelligent decision-making in IIoT scenarios. However, the deployment of such models in industrial environments faces challenges due to resource constraints in communication and computation. To address this problem, this paper proposes a collaborative optimization framework that integrates client-side feature learning, a hierarchical client–edge–cloud federated aggregation, and network-computing resource scheduling for efficient large-model embedding training. A parameter search method based on the Kepler Optimization Algorithm (PSKOA) is introduced to jointly optimize the three interdependent dimensions: client-side model structure parameter, federated aggregation parameters, and scheduling strategy. Evaluations demonstrate that the proposed method significantly reduces model loss by 41.7% and shortens training time by 13.4% compared to the traditional Genetic Algorithm-based method. Additionally, the proposed method achieves 12.5% lower model loss and 3.1% faster training time compared to the Particle Swarm Optimization-based method. These results highlight that the proposed method effectively enhances both training efficiency and convergence performance by jointly optimizing communication, computation, and model structure, making it a practical and scalable solution for large-model embedding training in resource-constrained IIoT environments.



Academic Editor: Jinfeng Xu

Received: 5 November 2025

Revised: 19 November 2025

Accepted: 27 November 2025

Published: 3 December 2025

Citation: Luo, Y.; Jin, X.; Xia, C.; Xu, C.; Sun, Y. Communication-Computation Co-Optimized Federated Learning for Efficient Large-Model Embedding Training. *Mathematics* **2025**, *13*, 3871. <https://doi.org/10.3390/math13233871>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: large-model embedding training; federated learning; scheduling; client–edge–cloud; IIoT

MSC: 68T20

1. Introduction

With the increasing penetration of IIoT and smart manufacturing, massive heterogeneous data are being continuously generated at high frequency and over long durations. These data exhibit significant non-IID characteristics due to differences across factories, production lines, and regions [1]. Besides, the data carry rich and comprehensive information, enabling models to achieve accurate cognition and reasoning in complex industrial scenarios.

In recent years, breakthroughs in large-model pretraining techniques have given rise to models with unprecedented capabilities [2]. These models demonstrate strong generalization and transferability, particularly in understanding and decision support tasks [3,4]. In the context of IIoT, this implies that tasks traditionally requiring experienced engineers to manually compare and diagnose can now be automated or semi-automated by leveraging large models' perception and natural language understanding. For example, in predictive maintenance, a model analyzes anomalous keywords in equipment logs, thereby achieving precise fault diagnosis and early warning [5]. This not only improves maintenance efficiency but also reduces production downtime losses, facilitating the transition from reactive maintenance to proactive closed-loop optimization in industrial production.

Nevertheless, deploying large models directly in IIoT environments remains challenging. The emergent capabilities of large models primarily stem from learning and generalizing from vast training corpora. Although inference requires only a small amount of data to deliver accurate results, training demands substantial computational power and memory to handle large-scale datasets, which exceeds the capacity of resource-constrained clients [6]. Furthermore, uploading raw data such as high-resolution video and high-frequency time-series signals directly to the cloud for centralized processing would result in excessive uplink bandwidth overhead and uncertain queuing delays, potentially causing intolerable latency in production processes. Therefore, how to harness the cognitive power of large models under constrained network-computing resources has become a key scientific and engineering challenge in advancing smart manufacturing.

To address the challenge of large model training in IIoT, a promising approach is to perform feature learning at the clients. Even devices with constrained resources can efficiently join the training process at low cost [7,8]. Meanwhile, the cloud server aggregates the parameters from distributed devices to perform a comprehensive global analysis. This distributed training paradigm resembles Federated Learning (FL), wherein participating devices train models locally using private data and upload only model parameters to an aggregator, which then coordinates global updates [9]. Such a mechanism reduces raw data transmission, thereby reducing network load.

However, directly applying conventional FL to large models in IIoT remains inadequate. The transmission of parameters in large models still imposes a heavy communication load [10], while the significant heterogeneity in device computing capacity makes it difficult for efficient training. Existing studies often focus on improving either communication or computation efficiency of FL to achieve efficient training [11], but rarely address the scenario where both network-computing resources are simultaneously constrained.

Under the dual constraints of network-computing resources, this paper formulates the problem as a multi-objective optimization problem. This problem is then decoupled into three interdependent dimensions to achieve the co-optimization of computation and communication performance.

The contributions of this paper are as follows:

- To enable efficient large-model embedding training in IIoT, we adopt a client-edge-cloud hierarchical architecture that jointly considers limited network-computing resources. The system's communication and computation performance is formulated as a constrained multi-objective optimization problem, which is decoupled into three levels: client-side feature learning, federated aggregation, and network-computing resource scheduling.
- We further customize a swarm intelligence-inspired Kepler Optimization Algorithm (KOA) to jointly optimize client-side model parameters, aggregation strategies, and network-computing scheduling strategy. The improved KOA incorporates a process-

information storage table to significantly reduce repetitive computations and employs a priority-uniqueness mapping mechanism to preserve the uniqueness of priority assignments.

- Extensive evaluations demonstrate that the proposed method achieves lower model loss and shorter training time compared with existing approaches, validating its effectiveness in balancing training efficiency and model accuracy under constrained network–computing resources.

The rest of the paper is organized as follows: Section 2 reviews related work. Section 3 describes the FL-based system and models our multi-objective optimization problem. Section 4 introduces our algorithms. Section 5 shows evaluation results. Section 6 concludes this paper.

2. Related Work

2.1. IIoT and Large Models

The Industrial Internet of Things (IIoT) has become a key driver of intelligent manufacturing, integrating sensing, communication, and computation across distributed workshops [12]. Through large-scale deployment of sensors, controllers, and industrial gateways, IIoT enables real-time monitoring, predictive maintenance, and adaptive control. The collected data, including vibration, image, and temperature signals, are often high-frequency, heterogeneous, and non-IID, which poses significant challenges for intelligent analytics and decision-making [13].

Traditional fault diagnosis relied heavily on expert knowledge, which limited scalability and robustness. With the development of artificial intelligence, large models have shown powerful representation capabilities for complex industrial data. Pretrained large models have achieved remarkable performance in fault detection, quality inspection, and process optimization [14,15], providing a foundation for data-driven manufacturing. However, their immense parameter scale and computation demands make them difficult to deploy on resource-constrained IIoT [16].

To overcome these limitations, edge intelligence has emerged as a promising paradigm that integrates AI computation into edge infrastructure, reducing communication costs [17]. By processing data closer to its source, it minimizes latency and improves the overall efficiency of industrial intelligent applications. Despite these advances, deploying large models in IIoT remains challenging due to the coexistence of limited computation resources and unstable communication bandwidth [18]. Therefore, this paper focuses on collaborative optimization that jointly considers computation and communication to enable reliable industrial intelligence.

2.2. Federated Learning in IIoT

Federated Learning (FL) provides a natural fit for IIoT, as it allows data to remain local while enabling distributed training, thereby protecting privacy and improving efficiency [19]. Compared with centralized approaches, FL shows clear advantages in handling heterogeneous data, resource-constrained devices, and limited bandwidth [20]. Recent surveys highlight key challenges, including heterogeneous data modeling, device adaptability, and communication optimization, while also reporting successful applications in automotive, robotics, agriculture, energy, and healthcare [21].

Integration of FL with edge computing has also attracted attention. For instance, Liu et al. proposed a software-defined AI-oriented edge framework for the IIoT, incorporating a time series-based device selection and offloading method in federated learning, which reduced training time by 30–50% and energy consumption by 35–55% compared to random selection [22]. With the growing adoption of edge computing, end-edge-cloud collaboration has become a dominant paradigm, enabling real-time monitoring and fault diagnosis closer to the data source [23,24]. Dynamic resource allocation, cross-layer scheduling, and

federated reinforcement learning have been identified as key strategies to further enhance system performance [25].

Nevertheless, most existing work in FL for IIoT focuses either on improving communication [26] or computation efficiency [27]. The co-optimization of communication and computation under simultaneous resource constraints has received little attention, a gap this paper aims to address.

3. System Model and Problem Statement

3.1. System Architecture

We consider a three-layer (client-edge-cloud) architecture for large-model embedding training in IIoT environments. In this architecture, clients are responsible for local feature learning; edge servers handle local aggregation and regional parameter updates, while the cloud server performs global aggregation and determines the scheduling strategy for limited network-computing resources. This architecture is designed to maximize the utility of the distributed computing resources and enhance communication efficiency under limited bandwidth, thereby achieving an effective co-optimization for computation and communication performance.

As illustrated in Figure 1, the system comprises three layers: cloud, edge, and client. These layers are interconnected via wired links between the cloud and edge, and wireless connections between the edge and clients. The architecture includes one cloud server, m edge servers, and multiple clients. Formally, the set of edge servers is denoted as $M = \{M_1, M_2, \dots, M_m\}$, each associated with a dedicated set of clients. These clients, deployed across workshops and production lines, continuously generate industrial data streams. Due to their limited computational capacity, clients cannot train the full large model. As a result, their task is focused on feature learning. Subsequently, they upload the learned model parameters rather than raw data to their associated edge server, thereby alleviating bandwidth load.

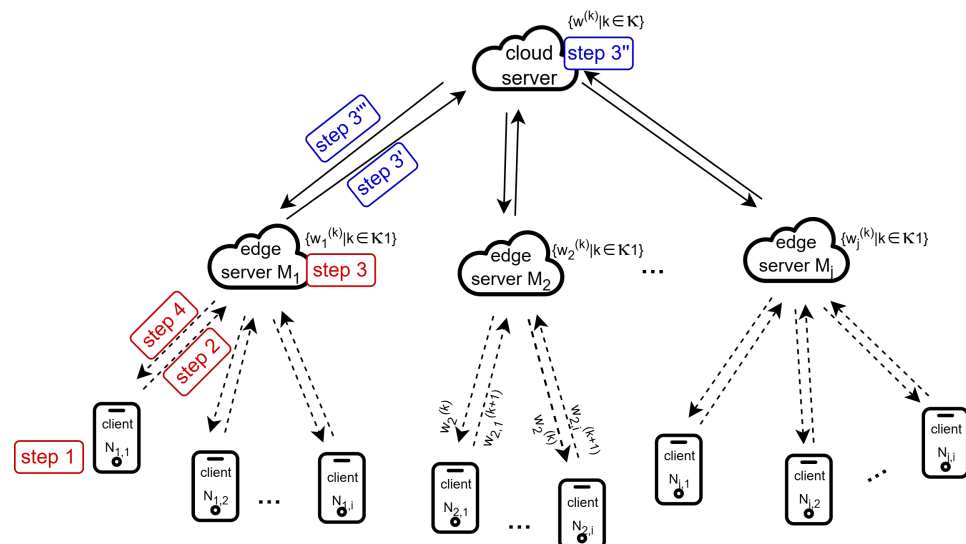


Figure 1. A client-edge-cloud architecture of FL-based IIoT.

Each edge server M_j manages its client set $N_j = \{N_{j,1}, N_{j,2}, \dots\}$ and coordinates the hierarchical training workflow. As illustrated in Figure 1, the global model is first initialized at the cloud and distributed to all clients through their corresponding edge servers. In Step 1, each client performs local feature learning for K_2 iterations using private data. Upon completion, the updated model parameters are uploaded to the edge server in Step 2. After collecting the updates from scheduled clients, the edge server conducts

local aggregation in Step 3 and redistributes the aggregated parameters to its clients in Step 4, completing one client–edge interaction round. This process repeats for $K1$ rounds. Once $K1$ client–edge rounds are completed, the edge server uploads its aggregated model parameters to the cloud for global aggregation (Step 3' and Step 3''). The cloud then broadcasts the updated global model back to all edge servers, which further forward it to their clients (Step 3'''). This completes one global aggregation round. The entire hierarchical training procedure continues for K global rounds. This hierarchical parameter exchange mechanism is illustrated in Figure 2.

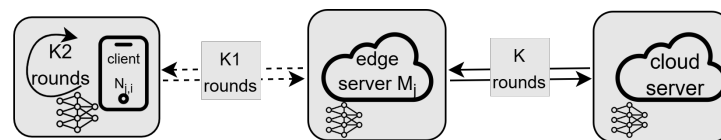


Figure 2. The relationship of updating round in the client-edge-cloud architecture.

The cloud server, functioning as the global decision-maker, possesses substantial computation and storage resources. Its primary responsibilities include aggregating model parameters from the edge servers and determining the optimal network-computing resource scheduling strategy, which is subsequently disseminated to the edge servers and clients.

In this system, communication and computation are tightly coupled. The wireless uplink is constrained by a maximum system capacity Q , limiting the number of simultaneous data streams. For the downlink, the broadcast mechanism inherently avoids link contention, introducing only a fixed delay that can be incorporated into the edge servers' computation time.

Furthermore, the heterogeneous computational capabilities of clients, edge servers, and the cloud server result in varying computing times, denoted as T_{client} , T_{edge} , and T_{cloud} , respectively. T_{client} also accounts for differences in the model structures across clients. Critically, the gap between computation and communication times is not significant enough to render the latter negligible. The communication time between a client $N_{j,i}$ and its edge server M_j is denoted as $T_{i,j}$. To simulate different wireless conditions, we assign varying values to the wireless capacity. The communication time is then set based on realistic magnitudes observed in practical network-computing environments.

3.2. Problem Statement

In the FL framework for network-computing co-optimization, the primary objective is to achieve efficient feature learning. However, heterogeneity in device computing capacities and bandwidth constraints poses significant challenges. Improper configuration of system parameters can consequently slow down the global model's convergence rate or even degrade its final performance. Therefore, it is crucial to jointly optimize both computation and communication performance.

The overall system performance is determined by two closely intertwined factors described in Section 4. First, model accuracy is influenced by the client-side model structure parameters and federated aggregation parameters. Second, the total training time is affected by the scheduling strategy and the same set of aggregation parameters.

It is important to note that increasing federated aggregation parameters (e.g., K , $K1$, $K2$) to improve model accuracy inevitably increases either the computational workload or the number of communication rounds, thus extending the overall training duration. Similarly, increasing the complexity of the model structure to boost accuracy results in longer computation times. Additionally, an improper *Priority* parameter can degrade the aggregation process at edge servers. As a result, the system must jointly optimize multiple interdependent parameters, including the model structure parameter α , the federated

aggregation parameters K , $K1$, and $K2$, and the network-computing scheduling parameter *Priority*, creating a multi-objective optimization problem.

Based on this formulation, the optimization objective of the system is defined as follows:

$$\min(w_T \times T + w_{loss} \times loss) \quad (1)$$

where

$$\begin{aligned} T_{total} &= f(Priority, K, K1, K2, Q), \\ loss &= f(\alpha, K, K1, K2). \end{aligned}$$

where T denotes the normalized overall training time derived from the original value T_{total} , and $loss$ represents the global model's training loss. The computation procedures for T_{total} and $loss$ are provided in Section 4. The weighting coefficients w_T and w_{loss} adjust the trade-off between communication efficiency and learning accuracy, where assigning a larger weight to one term emphasizes the importance of optimizing that aspect of the system's performance. Consequently, the overall training effectiveness is jointly determined by the client-side model structure parameter α , the federated aggregation parameters K , $K1$, and $K2$, as well as the network-computing scheduling priority parameter *Priority*.

To ensure fairness and comparability, the total training time T_{total} is transformed into a normalized value T through the following exponential scaling:

$$T = 1 - \exp(-\tau \times T_{total}) \quad (2)$$

where τ is a constant scaling factor. This transformation bounds T within the interval $[0, 1)$, aligning its magnitude with the typical range of model loss values and preventing the time term from dominating the optimization objective solely due to scale differences.

Moreover, this weighted objective formulation enables flexible system optimization under different operational priorities. When w_T is assigned a larger value, the optimizer tends to favor configurations that reduce communication and scheduling delays, even at the cost of slightly higher loss. Conversely, increasing w_{loss} encourages the search to prioritize model accuracy, potentially tolerating longer training durations. This adjustable balance allows the framework to adapt to diverse industrial scenarios where either rapid convergence or high predictive quality may be preferred.

Furthermore, the scheduling must satisfy the communication constraint, ensuring that the number of simultaneous data flows, which corresponds to the number of clients transmitting data, does not exceed the maximum wireless network capacity Q in each round. Additionally, the heterogeneous computational capabilities of the devices result in varying computation times, which directly influence the overall training efficiency.

4. Algorithm Design

Building on the system architecture and problem statement described in Section 3, this section proposes a communication and computation co-optimization method that adopts a parameter search method based on the Kepler Optimization Algorithm (PSKOA). The optimization is conducted across three key dimensions:

- $K, K1, K2$: These parameters govern the federated learning process across the client-edge-cloud hierarchy. They control the number of local iterations and aggregation rounds, which are critical for balancing model performance with communication overhead.

- *Priority*: This parameter governs the sequence of client-side parameter transmission. An optimal priority scheme mitigates bottlenecks under bandwidth constraints, thereby reducing the total training time and improving resource utilization.
- α : This parameter defines the complexity of the feature learning model on each client. Optimizing α enables resource-constrained clients to achieve accurate training while adhering to their computational limits.

The three parameter sets are inherently coupled, $K, K1, K2$ and α jointly determine the model accuracy, while $K, K1, K2$ and *Priority* collectively govern the training time. This strong interdependence gives rise to a complex and non-convex solution space for the multi-objective optimization problem.

To address this problem, we employ the Kepler Optimization Algorithm (KOA), a swarm intelligence method inspired by orbital mechanics. KOA conceptualizes the search process as a dynamic solar system, where the Sun represents the current best solution, and planets symbolize candidate solutions. Guided by gravitational forces, the planets iteratively adjust their trajectories to approach the Sun. This mechanism effectively balances global exploration with local exploitation, making KOA particularly suitable for our complex multi-objective optimization problem.

4.1. Encoding and Decoding of Solutions

In the optimization process, each planet corresponds to a candidate solution represented by a set of system configuration parameters. The position vector of a planet encodes these parameters and is initialized as follows:

$$P_i^j = P_{\text{low}}^j + \text{rand}_{[0,1]} \times (P_{\text{up}}^j - P_{\text{low}}^j), \quad \begin{cases} i = 1, 2, \dots, \text{Agents_n}, \\ j = 1, 2, \dots, \text{dim}, \end{cases} \quad (3)$$

where $P_i = [K, K1, K2, \text{Priority}, \alpha]$ denotes the position vector of the i -th planet in the dim -dimensional search space; P_{low}^j and P_{up}^j represent the lower and upper bounds of the j -th decision variable, respectively, and $\text{rand}_{[0,1]}$ is a uniformly distributed random number within $[0, 1]$, ensuring that all planets are evenly initialized within the feasible solution space; Agents_n denotes the total number of planets maintained in the search process of the algorithm.

The gravitational mechanism that governs planetary motion is used to simulate the search dynamics of candidate solutions. In physics, the Sun's gravitational force keeps the planets in elliptical orbits; without this force, the planets would travel in a straight line toward infinity [28]. Similarly, in PSKOA, the best performing solution, referred to as the Sun, generates a gravitational pull that influences all other candidate solutions (planets), guiding them toward the most promising regions of the search space. Each planet has its own gravitational strength, proportional to its fitness value, and its trajectory is influenced by the attraction from the Sun.

After initialization, each candidate solution is decoded into a specific system configuration and evaluated based on two optimization objectives: total training time (T_{total}) and model loss (loss). The candidate solution with the best fitness is designated as the Sun. During subsequent iterations, the motion of all other planets is governed by the gravitational force exerted by the Sun. The attraction force of the Sun P_S and any planet P_i is defined as:

$$Fg_i(t) = e_i \times \mu(t) \times \frac{\overline{M}_{P_S} \times \overline{M}_{P_i}}{\overline{R}_i^2 + \varepsilon} + r_1 \quad (4)$$

where $\overline{M}_{P_S}$ and $\overline{M}_{P_i}$ denote the normalized masses of P_S and P_i , respectively. These quantities are obtained by normalizing the original masses M_{P_S} and M_{P_i} as defined in Equations (6) and (7). In this formulation, the function $Fhd(\cdot)$ corresponds to the objective function in this paper. e_i is the eccentricity of the planet's orbit, a value between 0 and 1, introduced to endow KOA with a stochastic characteristic; μ is a gravitational constant that decays exponentially over time; ϵ is a small positive constant to prevent division by zero; r_1 is a value in the range $[0, 1)$, introducing stochastic perturbations to enhance exploration.

$R_i(t)$ measures the Euclidean distance between the current system configuration P_i and the best-performing configuration P_S in the search space.

$$R_i(t) = \|P_S(t) - P_i(t)\|_2 = \sqrt{\sum_{j=1}^{dim} (P_S^j(t) - P_i^j(t))^2}. \quad (5)$$

In this work, the Euclidean distance corresponds to the sum of the $L2$ norms computed across all parameter dimensions between the current configuration and the current optimal configuration, capturing how far a candidate solution deviates from the optimum. The normalized value $\overline{R}_i$ is then obtained by scaling $R_i(t)$ into the range $[0, 1]$, enabling consistent comparison across different optimization iterations.

The masses of the Sun and each planet i at iteration t are computed directly from their fitness values using the objective function $Fhd(\cdot)$ in Equation (12), as follows:

$$M_{P_S} = r_2 \frac{Fhd_S(t) - worst(t)}{\sum_{k=1}^{Agents_n} (Fhd_k(t) - worst(t))}, \quad (6)$$

$$M_{P_i} = \frac{Fhd_i(t) - worst(t)}{\sum_{k=1}^{Agents_n} (Fhd_k(t) - worst(t))}. \quad (7)$$

where

$$Fhd_S(t) = \text{Sun_score}(t) = \min_{k \in \{1, 2, \dots, Agents_n\}} Fhd_k(t), \quad (8)$$

$$worst(t) = \max_{k \in \{1, 2, \dots, Agents_n\}} Fhd_k(t). \quad (9)$$

Here, r_2 is a random number between 0 and 1, used to diversify the mass distribution among planets. Fhd denotes the function used to evaluate the fitness of each planet as shown in Equation (12), and Sun_score represents the best fitness value, corresponding to the fitness of the Sun.

Under the gravitational force Fg defined in Equation (4), the position update for each planet P_i is governed by the following kinematic mechanism:

$$P_i(t+1) = P_i(t) + f \times V_i(t) + (Fg_i(t) + |r_3|) \times U \times (P_S(t) - P_i(t)) \quad (10)$$

where $P_i(t+1)$ is the new position of planet i at time $t+1$; f is a random sign factor with equal probability of being +1 or -1, employed to randomly reverse the search direction and thus promote exploratory behavior; $V_i(t)$ is the velocity of planet i required to reach the new position; $Fg_i(t)$ is the gravitational force of planet i at time t ; U is a binary mask vector that selectively activates specific dimensions for update; $|r_3|$ is the absolute value of a random variable drawn from a standard normal distribution, injecting Gaussian noise to mitigate premature convergence; $P_S(t)$ denotes the position of the Sun obtained up to iteration t .

Overall, the Kepler Optimization Algorithm (KOA) is used to optimize the system configuration parameters. Each planet represents a candidate solution and is first ran-

domly initialized within the search space (Equation (3)). The fitness of each planet is then evaluated using the weighted objective function (Equation (12)), from which its mass and the inter-planet distances are computed (Equations (7) and (5)). Based on these values, the gravitational force acting on each planet is obtained (Equation (4)), and the planet's position is updated accordingly, considering gravitational forces and orbital mechanics (Equation (10)). This update follows Newton's laws of motion and Kepler's laws of planetary motion. This procedure repeats until the maximum search rounds T_{max} is met, and the planet with the best fitness is selected as the optimal system configuration. Through this iterative process, KOA effectively explores the coupled parameter space and identifies configurations that balance training accuracy and time.

In this work, the parameter *Priority* in P_i defines the execution order of clients. The presence of duplicate values in *Priority* would introduce ambiguity into the scheduling sequence. Although uniqueness is guaranteed during initialization, duplicate values may arise in the search process. To address this issue, we adopt a mapping scheme in Algorithm 1 to ensure that any duplicate values are transformed into unique ones. The specific mapping rules are outlined as follows:

- (1) When traversing the *Priority* vector, the first occurrence of a value is retained. Any subsequent duplicate is flagged for replacement.
- (2) Each duplicate entry $Priority[i]$ is replaced by a value selected from the set of unused integers R . The replacement value is determined by:

$$Priority[i] \leftarrow R[Priority[i] \bmod r] \quad (11)$$

where r denotes the number of elements in the unused set R .

- (3) After replacement, the set R is updated by removing the assigned value, thereby maintaining the uniqueness of all entries in *Priority*.

Algorithm 1 Parameters Search based on KOA

Require: the number of planets $Agents_n$,
the maximum search rounds T_{max} ,
the initialized position of any planet i P_i
Ensure: the optimal solution Sun P_S ,
the best fitness value Sun_score

```

1:  $PL\_Fit = Fhd(P_i)$  for each planet  $i$ 
2:  $Sun\_score(0) \leftarrow \min(PL\_Fit)$ 
3:  $P_S(0) \leftarrow P[\arg \min(PL\_Fit)]$ 
4: while  $t < T_{max}$  do
5:   for each  $i \in [0, Agents\_n)$  do
6:      $P\_new_i \leftarrow P_i$ 
7:      $PL\_Fit1 = Fhd(P\_new_i)$ 
8:     if  $PL\_Fit1 < PL\_Fit[i]$  then
9:        $PL\_Fit[i] = PL\_Fit1$ 
10:       $P_i = P\_new_i$ 
11:      if  $PL\_Fit[i] < Sun\_score$  then
12:         $Sun\_score(t) = PL\_Fit[i]$ 
13:         $P_S(t) = P_i$ 
14:      end if
15:    end if
16:  end for
17: end while
18: return  $P_S, Sun\_score$ 

```

4.2. Parameters Search

The parameter search method based on the Kepler Optimization Algorithm (PSKOA) is presented in Algorithm 1. In each iteration, the fitness of the current best solution (Sun), denoted *Sun_score*, evaluates overall system performance. This fitness is computed using the weighted function in Equation (12), which combines two metrics: model loss (*loss*) and total training time (*T_{total}*), computed by Algorithm 2 and Algorithm 3, respectively.

$$Fhd(P_i) = w_T \times T + w_{loss} \times loss \quad (12)$$

where, w_T and w_{loss} are weights that balance the system's preference for communication efficiency (minimizing time) against computation accuracy (minimizing loss). This formulation achieves the joint optimization of both objectives, and the configuration yielding the minimum *Sun_score* is selected as the optimal solution.

Algorithm 2 Federated Learning Computing

Require: $M_j, M, N_{j,i}, N_j, K, K1, K2, \alpha$

Ensure: Global model average loss *loss*

```

1: for global_epoch = 1 to K do
2:   Distribute global model to clients as initialized/updated local models
3:   for  $M_j \in M$  do
4:     for local_epoch = 1 to K1 do
5:       for  $N_{j,i} \in N_j$  do
6:         Client  $N_{j,i}$  trains local model for K2 iterations using local data
7:       end for
8:       Edge server  $M_j$  aggregates parameters from connected clients
9:       Edge server  $M_j$  redistributes updated parameters to clients
10:    end for
11:  end for
12:  Cloud server updates global model
13:  Calculate global model average loss loss
14: end for
15: return loss

```

Given a specific set of configuration parameters (*Priority*, *K1*, *K2*, *K3*, α), the corresponding *loss* and *T_{total}* are first obtained, and then the overall fitness value is derived accordingly. This fitness value serves as the optimization criterion that guides planetary position updates in KOA, enabling the algorithm to iteratively converge toward the optimal configuration.

The process of planetary position updating is described in Algorithm 1. First, all candidate planets are initialized and evaluated, and the best one is selected as the Sun (Lines 1–3). The algorithm then enters the iterative phase, where each planet updates its position based on KOA's orbital mechanics model and subsequently undergoes re-evaluation (Lines 4–7). The fitness of the updated position P_{new_i} (Line 7) is calculated by decoding it into a system configuration, computing the *loss* and *T* metrics using Algorithms 2 and 3, and then applying Equation (12) to obtain the final fitness value. If any updated planet achieves a fitness superior to that of the current Sun, it replaces the latter as the new global best solution (Lines 8–13). This iterative search continues until the maximum search rounds T_{max} is met, after which the algorithm returns the optimal configuration (Sun) and its corresponding fitness value (*Sun_score*).

Algorithm 3 Federated Learning Scheduling

Require: Scheduling strategy *Priority*
 Federated aggregation parameters $K, K1, K2$
 Computation time: $T_{client}, T_{edge}, T_{cloud}$
 Communication time between client and edge server: $T_{i,j}$
 Wireless communication maximum capacity Q
 Maximum scheduling time $TIME$

Ensure: Total model training time T_{total}

```

1: Initialize states of clients, edge servers, and cloud server
2:  $t \leftarrow 0$ 
3: while  $t < TIME$  do
4:   for  $k = 1$  to  $K$  do
5:     Schedule clients based on Priority and  $Q$ 
6:     for  $k1 = 1$  to  $K1$  do
7:       Clients compute:  $K2$  local iterations taking time  $T_{client}$  each iteration
8:       Clients communicate with edge servers: time  $T_{i,j}$ 
9:       Edge servers aggregate: time  $T_{edge}$ 
10:    end for
11:    Cloud server aggregates: time  $T_{cloud}$ 
12:  end for
13:   $t \leftarrow t + 1$ 
14:  if scheduling completes then
15:    break
16:  end if
17: end while
18:  $T_{total} \leftarrow t$ 
19: return  $T_{total}$ 

```

It is important to highlight that each parameter update in the algorithm necessitates a full re-evaluation of the candidate solution, which introduces considerable computational overhead, even for small changes. However, the performance of FL remains largely unaffected when the parameter combination $(K, K1, K2, \alpha)$ stays the same. To address this inefficiency, we introduce a query table in line 7 of Algorithm 1, which stores previously evaluated parameter combinations along with their corresponding loss values. When evaluating a new candidate solution, the algorithm first checks the table for an existing match. If a match is found, the precomputed loss value is used directly; if no match exists, the loss is computed using Algorithm 2 and the result is then added to the table. This approach effectively eliminates redundant computations, optimizing the overall process.

The loss $loss$ is calculated as described in Algorithm 2. Firstly, clients receive the initialized global model (Line 2) and conduct local training for $K2$ iterations on their local datasets. Upon completing local training, they conduct client-edge interactions for $K1$ rounds (Lines 4–10). Finally, a global aggregation is performed at the cloud server (Line 12). The resulting global average loss, denoted as $loss$, serves as the metric for evaluating the client-side model's performance throughout the training process.

Unlike the traditional two-layer FL architecture, this paper introduces an additional layer of edge servers, which aggregate the connected clients and transmit the aggregated model parameters to the cloud server.

Accordingly, the computational complexity of the process is $O(K \times (K1 + agg(\cdot)) \times (K2 \times clients \times tra(\cdot) + agg(\cdot)))$, which can be denoted as $O(n^4 \times tra(\cdot))$. Here, $agg(\cdot)$ denotes the aggregation time at the edge or cloud server, and $tra(\cdot)$ represents the client-side training time for one iteration. When calculating the loss $loss$ using Algorithm 2, maintaining the storage table for loss values significantly reduces the overall computational cost by avoiding redundant $tra(\cdot)$ operations.

Algorithm 3 calculates the total training time T_{total} under the communication constraint Q . The procedure begins with device initialization (Line 1). Each client then performs local computation for $K2$ iterations, consuming a local computation time of T_{client} per iteration. According to the scheduling order determined by *Priority*, clients communicate with their corresponding edge servers, where each communication takes a duration of $T_{i,j}$ (Lines 5–8). Subsequently, the edge servers perform aggregation, requiring a time of T_{edge} (Line 9). After completing $K1$ rounds of client–edge interactions, global aggregation is carried out at the cloud server (Line 11), which consumes T_{cloud} . When the number of global updates reaches K , the scheduling process terminates, and the algorithm returns the total time consumed by the entire training process, denoted as T_{total} .

In this scheduling process, FL involves three-layer communication with a time complexity of $O(K \times K1 \times K2)$. Moreover, in Algorithm 3, each client is required to communicate. Therefore, the overall time complexity is $O(K \times K1 \times K2 \times clients)$, which can be denoted as $O(n^4)$.

Through this design, the PSKOA algorithm effectively searches multidimensional parameter space (*Priority*, $K1$, $K2$, $K3$, α), with its fitness function integrating both communication and computation objectives.

5. Performance Evaluation

This section evaluates the proposed PSKOA method in comparison with traditional parameter search approaches based on Genetic Algorithm (PSGA) and Particle Swarm Optimization (PSO). The evaluation is conducted from three perspectives: (1) Overall system performance, measured by the weighted indicator *Sun_score*, which integrates both communication (T_{total}) and computation (*loss*) performance; (2) Communication performance, measured by *Sun_T_total*, representing the training time of the current optimal solution and reflecting the efficiency of synchronization and data transmission among client–edge–cloud devices; and (3) Computation performance, measured by *Sun_loss*, representing the final model loss associated with the optimal configuration and indicating the quality of the trained model. We conduct a series of evaluations under different system resource configurations to compare PSKOA with benchmark algorithms and validate its effectiveness in IIoT environments.

5.1. Experimental Setup

In the evaluation, we simulate a client–edge–cloud architecture consisting of 15 clients, 3 edge servers, and 1 cloud server. Preliminary evolutions showed that setting the global aggregation rounds $K = 100$ enables stable convergence, with the global model loss reaching a consistently low level. Figure 3a,b illustrate the loss trajectories under different K values when $K1 = 12$ and $K1 = 16$. Although increasing K generally reduces the loss, the improvement becomes marginal once K exceeds 100. Specifically, the curves for $K = 200$ and $K = 300$ show little additional gain compared with $K = 100$ while incurring significantly higher communication cost. Therefore, $K = 100$ is selected as a balanced choice between convergence performance and communication overhead, and we subsequently fix K at this value while optimizing the client local training rounds $K2$ and edge aggregation rounds $K1$.

We utilize a Recurrent Neural Network (RNN) as the base model and train it on the AWE dataset [29], a vibration signal dataset containing 23,436 labeled samples. Each sample represents a vibration signal segment annotated with equipment health status (e.g., “Fault-4 mm” or “Normal”). To evaluate the effectiveness of PSKOA in joint communication and computation optimization, we compare it against six baseline methods, including variants based on KOA, GA, and PSO, as summarized below:

- Parameter search based on Kepler Optimization Algorithm (PSKOA_d): This algorithm employs KOA to jointly optimize all parameters, including $K1$, $K2$, $Priority$, and α , explicitly balancing model accuracy and training time.
- Parameter search based on Kepler Optimization Algorithm with given $Priority$ (PSKOA_g): This variant uses KOA to optimize $K1$, $K2$, and α under a fixed $Priority$, disregarding communication scheduling.
- Parameter search based on Genetic Algorithm (PSGA_d): GA is employed to optimize all parameters, including $K1$, $K2$, $Priority$, and α , balancing model accuracy and training time.
- Parameter search based on Genetic Algorithm with given $Priority$ (PSGA_g): GA optimizes $K1$, $K2$, and α under a fixed scheduling $Priority$.
- Parameter search based on Particle Swarm Optimization (PSO_d): PSO is used to jointly optimize all parameters, including $K1$, $K2$, $Priority$, and α , balancing accuracy and training time.
- Parameter search based on Particle Swarm Optimization with given $Priority$ (PSO_g): PSO optimizes $K1$, $K2$, and α under a fixed $Priority$.

All six methods aim to optimize key parameters to balance model performance and training efficiency. The simulations are implemented in Python 3.10 on a machine equipped with an NVIDIA GeForce RTX 3080 GPU.

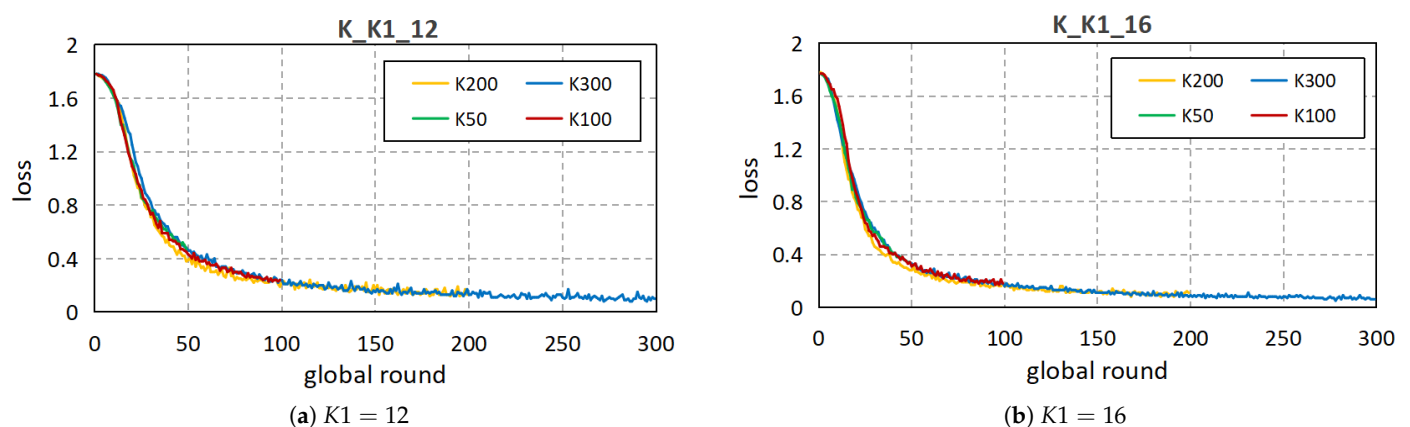


Figure 3. Training loss under different global aggregation rounds K for (a) $K1 = 12$ and (b) $K1 = 16$.

5.2. Simulation

The simulation evaluates the proposed PSKOA method under different system configurations, focusing on three key factors: the weighting between $loss$ and T_{total} in the Sun_score , the search range of client–edge interaction rounds $K1$, and the network communication capacity Q .

5.2.1. Weighting of $loss$ and T_{total}

In the first set of evolutions, the weights of training loss ($loss$) and total training time (T_{total}) are set to 0.6 and 0.4, respectively. The search range for $K1$ is defined as $(8, 22)$, and the network communication capacity is fixed at $Q = 5$.

Figure 4a shows how the Sun_score evolves with the number of search rounds. All six methods gradually improve system performance as the search progresses, with PSKOA_d consistently achieving the lowest Sun_score (around 0.5), indicating superior optimization capability. This advantage arises from KOA's mechanism of generating new candidate solutions around the current Sun , which helps avoid premature convergence. In comparison, PSO performs slightly worse, and GA is more prone to getting trapped in suboptimal solutions.

Furthermore, Figure 4b depicts the client-side loss trajectories under each method's optimal configuration. PSKOAD demonstrates superior convergence, reaching a loss of 0.07 in just 21,389 time units. This outperforms PSGAd (0.12 loss in 24,688 units) and PSOd (0.8 loss in 22,069 units), representing a 41.7% loss reduction and 13.4% faster training than PSGAd, and 12.5% lower loss with 3.1% shorter time than PSOd. These results confirm that optimizing the *Priority* parameter enhances performance by improving client-edge communication efficiency. As similar trends were observed in other settings, subsequent analyses omit these curves for brevity.

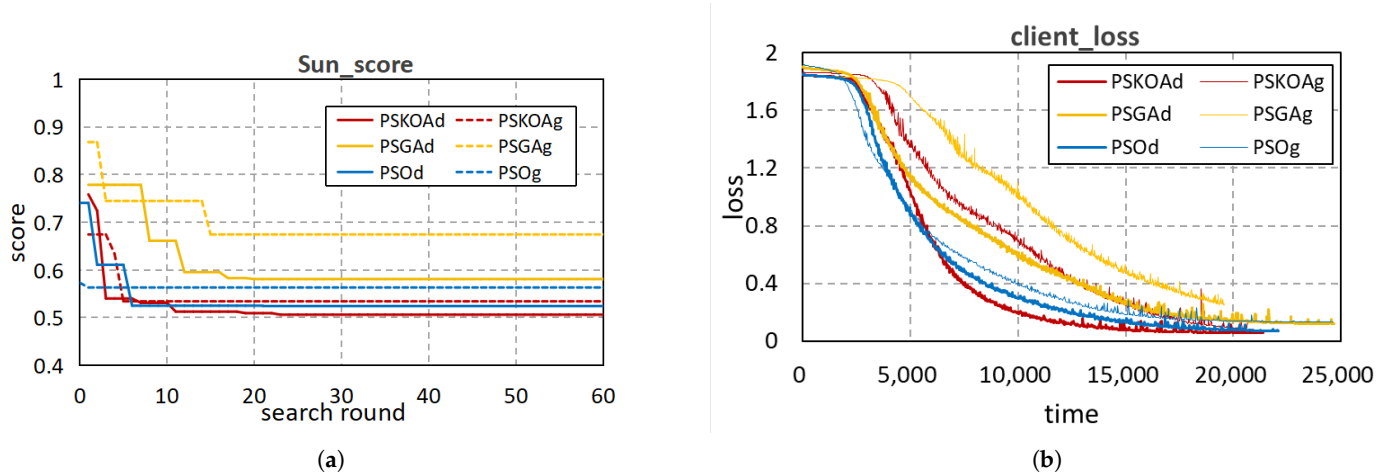


Figure 4. (a) Relationship between Sun_score and the number of search rounds when the weight ratio of *loss* to T_{total} is set to 0.6:0.4. (b) Variation in client-side loss over time after applying the optimal solutions obtained by different methods under the same weight setting (0.6:0.4).

Figure 5a,b present the evolution of *Sun_loss* and *Sun_T_total*, respectively, under the weight configuration ($w_{loss} = 0.6, w_T = 0.4$). As shown in Figure 5a, PSKOAD achieves the lowest training loss throughout the search process, demonstrating its strong optimization capability. In contrast, PSGA_g yields the highest *Sun_loss*, while the two PSO-based methods (PSOd and PSOG) maintain moderate performance, converging faster than the GA-based variants but remaining inferior to PSKOAD.

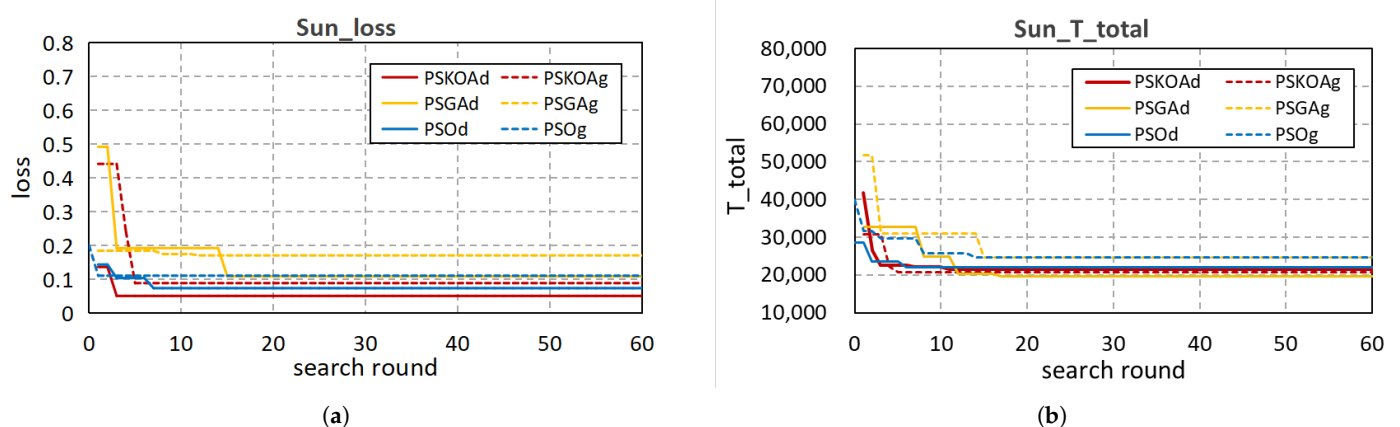


Figure 5. When the weight ratio of *loss* to T_{total} is set to 0.6:0.4. (a) Relationship between *Sun_loss* and the number of search rounds. (b) Relationship between *Sun_T_total* and the number of search rounds.

In terms of training time, Figure 5b shows that PSKOAD converges to approximately 2000 time units, achieving the best performance among all algorithms. Both PSO variants converge more quickly than their GA counterparts, but still require a longer duration than

PSKOA. PSGAg again performs the worst, taking substantially more time to converge. This inferior performance is primarily attributed to its fixed *Priority*, which prevents effective adjustment of network–computing resource allocation during optimization.

Overall, the results indicate that system performance in industrial FL hinges on a proper balance between training loss and training time. By jointly optimizing communication and computation parameters, the proposed PSKOA method achieves faster convergence and higher model accuracy, demonstrating superior efficiency under IIoT resource constraints.

When the weights are set to 0.8 for *loss* and 0.2 for T_{total} , with $K1 \in (8, 22)$ and $Q = 5$, the relationship between *Sun_score* and the number of search rounds is shown in Figure 6a. Three algorithms ultimately converge to the same optimal *Sun_score* of about 0.53, with PSKOA reaching this point the fastest, in only 11 rounds. The PSO-based methods (PSOd and PSOG) also approach this optimum but at a slower pace, reflecting KOA's stronger global search ability and its effective use of *Priority* optimization. Figure 6b shows the results when the weight ratio is changed to 0.9 for *loss* and 0.1 for T_{total} . As expected, the *Sun_score* of all six methods decreases as the search progresses. Within the first 20 rounds, PSKOA and PSKOA_g converge to a *Sun_score* of approximately 0.4, whereas the GA-based methods (PSGAd and PSGAg) remain near 0.5. The PSO algorithms again show moderate performance, better than GA but consistently inferior to KOA. These comparisons collectively demonstrate that KOA achieves a more effective balance between exploration and exploitation than both GA and PSO, resulting in faster and more stable convergence.

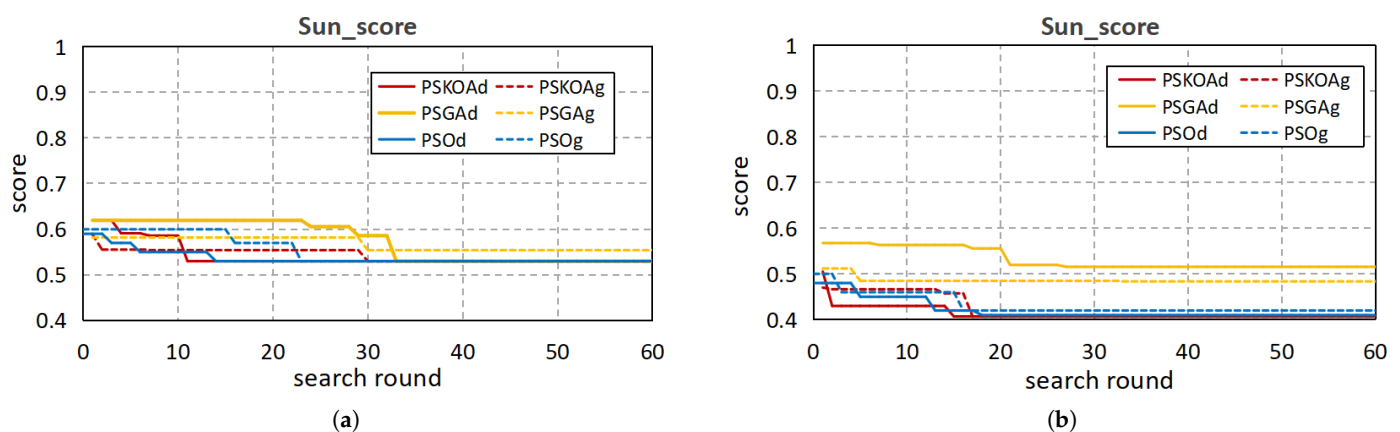


Figure 6. (a) Relationship between *Sun_score* and the number of search rounds when the weight ratio of *loss* to T_{total} is 0.8:0.2. (b) Relationship between *Sun_score* and the number of search rounds when the weight ratio of *loss* to T_{total} is 0.9:0.1.

5.2.2. Parameter Search Scope

Figure 7 illustrates the relationship between *Sun_score* and the number of search rounds, with weights settings of 0.6 for *loss* and 0.4 for T_{total} , $K1 \in (12, 26)$, and $Q = 5$.

It is noteworthy that the value range of $K1$ directly affects the optimization process. As illustrated in Figure 7, when $K1$ is searched within the interval (12, 26), the *Sun_score* curves of all approaches, including PSKOA, PSGA, and the PSO-based baselines (PSOd and PSOG), show a clear performance degradation compared with the case where $K1$ ranges in (8, 22). Although PSKOA still achieves the best solution among all algorithms, its *Sun_score* increases noticeably in the larger search range, indicating that an excessively large $K1$ leads to suboptimal system behavior. Therefore, selecting an appropriate $K1$ interval is crucial for achieving better overall performance.

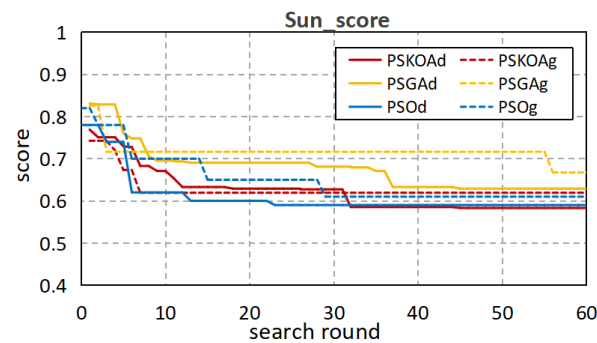


Figure 7. Relationship between Sun_score and the number of search rounds for $K1$ ranging in (12, 26).

5.2.3. Communication Capacity

Figure 8a presents the relationship between Sun_score and the number of search rounds when the network communication capacity is constrained to $Q = 3$, with the loss weight set to 0.6, the time weight set to 0.4, and the search range of $K1$ defined as (8, 22). Under this setting, PSKOAg converges the fastest within approximately 5 search rounds, yet the resulting solution is inferior. In contrast, PSKOAd requires around 18–19 search rounds to converge but achieves the lowest Sun_score among all approaches. The PSO-based variants (PSOd and PSOG) also exhibit slower improvement and stabilize at higher Sun_score values. Compared with the case of $Q = 5$, all algorithms produce larger Sun_score values when $Q = 3$, indicating performance degradation. This trend demonstrates that limited communication capacity prolongs the training process and reduces the overall system performance.

Figure 8b illustrates the case when the communication capacity is increased to $Q = 7$, with the same weight settings ($w_{loss} = 0.6, w_T = 0.4$) and $K1 \in (8, 22)$. PSKOA achieves a Sun_score close to 0.5, substantially lower than the approximately 0.6 achieved by PSGA. PSGAg becomes trapped in a local optimum, while PSKOA continues improving throughout the search. These results confirm that KOA more effectively leverages increased communication resources to achieve superior optimization performance.

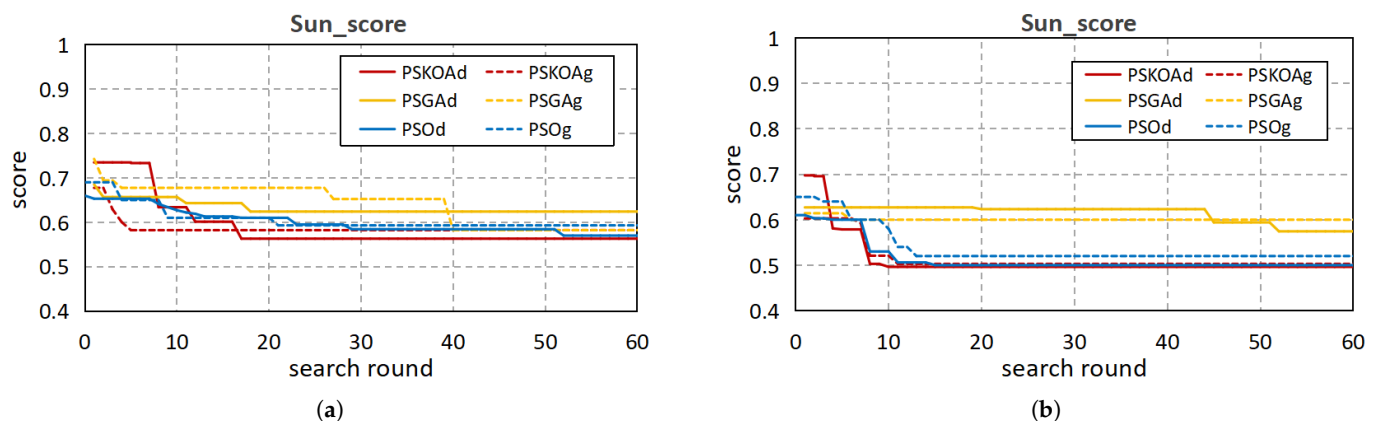


Figure 8. (a) Relationship between Sun_score and the number of search rounds when network communication capacity $Q = 3$. (b) Relationship between Sun_score and the number of search rounds when network communication capacity $Q = 7$.

By explicitly incorporating both network-computing latency, the proposed method more faithfully reflects the constraints of real industrial environments, thereby enhancing its practical relevance and deployment feasibility. Moreover, extensive evolutions under diverse system configurations and resource conditions demonstrate that the method re-

mains consistently effective and robust across a wide range of application scenarios, further underscoring its suitability for real-world IIoT systems.

6. Conclusions

This paper presented a collaborative optimization framework for efficient large model embedding training under constrained network–computing resources in IIoT environments. The proposed framework integrates client-side feature learning, hierarchical client–edge–cloud federated aggregation, and network–computing resource scheduling into a unified multi-objective optimization problem. To effectively solve this problem, we propose a parameter search method based on the Kepler Optimization Algorithm (PSKOA), which extends the original KOA with a process information storage mechanism to reduce redundant computation.

Extensive evaluations demonstrate that PSKOA consistently outperforms existing baseline methods. In particular, PSKOAd achieves approximately 41.7% lower model loss and 13.4% shorter training time than PSGAd, while also improving model accuracy by about 12.5% and reducing training time by 3.1% compared with PSOd. These results confirm that jointly optimizing model structure parameters, federated aggregation parameters, and scheduling priority substantially improves training efficiency and convergence performance. The consistent trends observed across different resource settings further validate the robustness and practicality of the proposed framework. Future work will explore extending PSKOA to heterogeneous multi-modal federated learning, as well as developing adaptive and intelligent scheduling mechanisms capable of handling dynamic network conditions and evolving resource constraints.

Author Contributions: Conceptualization, X.J.; methodology, Y.L.; software Y.L.; validation, Y.L. and Y.S.; writing—original draft preparation, Y.L., Y.S. and X.J.; writing—review and editing, C.X. (Changqing Xia) and C.X. (Chi Xu); funding acquisition, X.J., Y.S., C.X. (Changqing Xia) and C.X. (Chi Xu). All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the following grants: the National Natural Science Foundation of China (Grants 62133014 and 62303449); the Independent Subject of the State Key Laboratory of Robotics (Grant 2024-Z12); the Technology Program of Liaoning Province (Grants 2024020984-JH2/1024, 2024-MSBA-84, 2024-MSBA-85, and 2024JHI/11700050); the Liaoning Revitalization Talents Program (Grant XLYC2202048); and the Fundamental Research Project of SIA (Grant 2023JC1K09).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to institutional and commercial confidentiality agreements.

Conflicts of Interest: The authors declare no conflicts of interest.

Nomenclature

M	the set of edge servers
M_j	one edge server
N_j	the client set connected with edge server M_j
$N_{j,i}$	one client connected with edge server M_j
Q	the system maximum capacity of wireless communication
T_{client}	the computing time in clients
T_{edge}	the computing time in edge servers
T_{cloud}	the computing time in the cloud server

$T_{i,j}$	the communication time between the client $N_{j,i}$ and the edge server M_j
$K2$	the round of iterations performed by clients using local data
$K1$	the round of client-edge interaction within a global model update cycle
K	the round of global model updating
α	the client model structure parameter
<i>Priority</i>	the network-computing resource scheduling strategy
P_S	the position of the Sun
P_i	the position of any planet i

References

- Radford, A.; Kim, J.W.; Hallacy, C.; Ramesh, A.; Goh, G.; Agarwal, S.; Sastry, G.; Askell, A.; Mishkin, P.; Clark, J.; et al. Learning transferable visual models from natural language supervision. In Proceedings of the International Conference on Machine Learning, Virtual, 18–24 July 2021; pp. 8748–8763.
- Alayrac, J.B.; Donahue, J.; Luc, P.; Miech, A.; Barr, I.; Hasson, Y.; Lenc, K.; Mensch, A.; Millican, K.; Reynolds, M.; et al. Flamingo: A visual language model for few-shot learning. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 23716–23736.
- Li, Y.; Liu, H.; Wu, Q.; Mu, F.; Yang, J.; Gao, J.; Li, C.; Lee, Y.J. Gligen: Open-set grounded text-to-image generation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Vancouver, BC, Canada, 17–23 June 2023; pp. 22511–22521.
- Zhang, M.; Chang, K.; Wu, Y. Multi-modal semantic understanding with contrastive cross-modal feature alignment. *arXiv* **2024**, arXiv:2403.06355.
- Resende, C.; Folgado, D.; Oliveira, J.; Franco, B.; Moreira, W.; Oliveira, A., Jr.; Cavaleiro, A.; Carvalho, R. TIP4. 0: Industrial internet of things platform for predictive maintenance. *Sensors* **2021**, *21*, 4676. [[CrossRef](#)] [[PubMed](#)]
- Thompson, N.C.; Greenewald, K.; Lee, K.; Manso, G.F. The computational limits of deep learning. *arXiv* **2020**, arXiv:2007.05558.
- McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the Artificial Intelligence and Statistics, Ft. Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.
- Li, T.; Sahu, A.K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; Smith, V. Federated optimization in heterogeneous networks. *Proc. Mach. Learn. Syst.* **2020**, *2*, 429–450.
- Nishio, T.; Yonetani, R. Client selection for federated learning with heterogeneous resources in mobile edge. In Proceedings of the ICC 2019—2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–7.
- Nguyen, D.C.; Ding, M.; Pathirana, P.N.; Seneviratne, A.; Li, J.; Poor, H.V. Federated learning for internet of things: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1622–1658. [[CrossRef](#)]
- Almanifi, O.R.A.; Chow, C.O.; Tham, M.L.; Chuah, J.H.; Kanesan, J. Communication and computation efficiency in federated learning: A survey. *Internet Things* **2023**, *22*, 100742. [[CrossRef](#)]
- Gupta, P.; Krishna, C.; Rajesh, R.; Ananthakrishnan, A.; Vishnuvardhan, A.; Patel, S.S.; Kapruan, C.; Brahmabhatt, S.; Kataray, T.; Narayanan, D.; et al. Industrial internet of things in intelligent manufacturing: A review, approaches, opportunities, open challenges, and future directions. *Int. J. Interact. Des. Manuf.* **2022**, 1–23. [[CrossRef](#)]
- Singh, A.K.; Kundur, D.; Conti, M. Introduction to the special issue on integrity of multimedia and multimodal data in Internet of Things. *ACM Trans. Multimed. Comput. Commun. Appl.* **2024**, *20*, 1–4. [[CrossRef](#)]
- Zhang, X.; Rane, K.P.; Kakaravada, I.; Shabaz, M. Research on vibration monitoring and fault diagnosis of rotating machinery based on internet of things technology. *Nonlinear Eng.* **2021**, *10*, 245–254. [[CrossRef](#)]
- Rahmatov, N.; Paul, A.; Saeed, F.; Hong, W.H.; Seo, H.; Kim, J. Machine learning-based automated image processing for quality management in industrial Internet of Things. *Int. J. Distrib. Sens. Netw.* **2019**, *15*, 1550147719883551. [[CrossRef](#)]
- Hurst, A.; Lerer, A.; Goucher, A.P.; Perelman, A.; Ramesh, A.; Clark, A.; Ostrow, A.J.; Welihinda, A.; Hayes, A.; Radford, A.; et al. Gpt-4o system card. *arXiv* **2024**, arXiv:2410.21276. [[CrossRef](#)]
- Xu, D.; Li, T.; Li, Y.; Su, X.; Tarkoma, S.; Jiang, T.; Crowcroft, J.; Hui, P. Edge intelligence: Empowering intelligence to the edge of network. *Proc. IEEE* **2021**, *109*, 1778–1837. [[CrossRef](#)]
- Team, G.; Anil, R.; Borgeaud, S.; Alayrac, J.B.; Yu, J.; Soricut, R.; Schalkwyk, J.; Dai, A.M.; Hauth, A.; Millican, K.; et al. Gemini: A family of highly capable multimodal models. *arXiv* **2023**, arXiv:2312.11805. [[CrossRef](#)]
- Boobalan, P.; Ramu, S.P.; Pham, Q.V.; Dev, K.; Pandya, S.; Maddikunta, P.K.R.; Gadekallu, T.R.; Huynh-The, T. Fusion of federated learning and industrial Internet of Things: A survey. *Comput. Netw.* **2022**, *212*, 109048. [[CrossRef](#)]
- Ali, A.; Husain, M.; Hans, P. Federated Learning-Enhanced Blockchain Framework for Privacy-Preserving Intrusion Detection in Industrial IoT. *arXiv* **2025**, arXiv:2505.15376.

21. Kong, L.; Tan, J.; Huang, J.; Chen, G.; Wang, S.; Jin, X.; Zeng, P.; Khan, M.; Das, S.K. Edge-computing-driven internet of things: A survey. *ACM Comput. Surv.* **2022**, *55*, 1–41. [[CrossRef](#)]
22. Liu, X.; Dong, X.; Jia, N.; Zhao, W. Federated learning-oriented edge computing framework for the IIoT. *Sensors* **2024**, *24*, 4182. [[CrossRef](#)] [[PubMed](#)]
23. Liu, J.; Du, Y.; Yang, K.; Wu, J.; Wang, Y.; Hu, X.; Wang, Z.; Liu, Y.; Sun, P.; Boukerche, A.; et al. Edge-Cloud Collaborative Computing on Distributed Intelligence and Model Optimization: A Survey. *arXiv* **2025**, arXiv:2505.01821. [[CrossRef](#)]
24. Li, A.; Sun, J.; Zeng, X.; Zhang, M.; Li, H.; Chen, Y. Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking. In Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems, Coimbra, Portugal, 15–17 November 2021; pp. 42–55.
25. Wu, D.; Ullah, R.; Harvey, P.; Kilpatrick, P.; Spence, I.; Varghese, B. Fedadapt: Adaptive offloading for iot devices in federated learning. *IEEE Internet Things J.* **2022**, *9*, 20889–20901. [[CrossRef](#)]
26. Chen, M.; Poor, H.V.; Saad, W.; Cui, S. Convergence time optimization for federated learning over wireless networks. *IEEE Trans. Wirel. Commun.* **2020**, *20*, 2457–2471. [[CrossRef](#)]
27. Xia, Q.; Ye, W.; Tao, Z.; Wu, J.; Li, Q. A survey of federated learning for edge computing: Research problems and solutions. *High-Confid. Comput.* **2021**, *1*, 100008. [[CrossRef](#)]
28. Moser, J. Is the solar system stable? *Hamilt. Dyn. Syst. Repr. Sel.* **1987**, *1*, 20.
29. Zhang, W.; Yang, D.; Xu, Y.; Huang, X.; Zhang, J.; Gidlund, M. DeepHealth: A self-attention based method for instant intelligent predictive maintenance in industrial Internet of things. *IEEE Trans. Ind. Inform.* **2020**, *17*, 5461–5473. Available online: <https://github.com/Intelligent-AWE/DeepHealth> (accessed on 7 October 2020). [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.