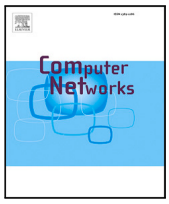
Computer Networks 271 (2025) 111592

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



DGPAS: DQN-GRU guided distributed DNN pipeline training and adjacent scheduling in edge networks

Jiayi Li [a], Xiaogang Wang [a,*], Haokun Chen [a], Zexin Wu [a], Ziqi Zhu [a], Jian Cao [b], Rajkumar Buyya [c]

[a] School of Electronic and Information, Shanghai Dianji University, Shanghai, 201306, Pudong New Area District, China
[b] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, 200240, Minghang District, China
[c] CLOUDS Laboratory, School of Computing and Information Systems, The University of Melbourne, Parkville, VIC 3010, Melbourne, Australia

ARTICLE INFO

Keywords:
Heterogeneous edge networks
DNN pipeline parallelism
DQN-GRU guided training
Adaptive adjacent scheduling

ABSTRACT

Deep neural networks (DNNs) are increasingly deployed in distributed edge computing environments to meet the computer vision detection tasks in various industrial production. However, DNN model training and inference at the edge sides often suffer from the situations such as limited computing resources, restricted bandwidth and heterogeneous device communication. The existing methods do not effectively solve the problems of imbalanced task scheduling and idle waiting time in heterogeneous edge devices. Toward this end, this paper proposes a DQN-GRU guided distributed DNN pipeline training and adjacent scheduling (DGPAS) model to significantly improve the training efficiency of DNNs for resource-constrained devices in heterogeneous edge networks. Combining deep reinforcement learning and temporal feature modeling, and based on the synchronous pipeline parallel mechanism, the model framework is divided into the preparation stage and execution phase for DNN model partitioning and device scheduling, respectively. In addition, an adaptive adjacent scheduling strategy during the execution phase is designed to effectively alleviate the bubble effect (i.e., the idle waiting time between edge devices), which is caused by data dependency from the DNN collaborative training between devices. Thereby, the DNN training time and the idle waiting time of devices are all improved in complex dynamic environments. Experimental results show that compared with the baseline and existing relevant methods, DGPAS reduces the average training time by 36.5% by integrating GRU with DQN under five types of mainstream DNN models. After adopting adaptive adjacent scheduling, the bubble rate is decreased by an average of 36.96% under the same DNN models and different number of mini-batches, significantly improving the training efficiency and robustness of the edge DNN model.

1. Introduction

In the field of industrial product component detection, deep neural networks (DNNs) have been widely adpoted [1–3], especially in computer vision [4,5] tasks such as quality control [6], defect recognition in optical thin films [7], mobile screens [8] and semiconductor manufacturing processes [9]. Most of traditional optics and visual detection methods employ the machine vision detection technology. Its principle involves using high-resolution cameras to capture images, and adopting algorithms to identify defects such as scratches and pits. However, the traditional algorithms require repeated parameter adjustments (such as light threshold values), resulting in high false detection rates under complex background interference. Moreover, customized detection solutions are needed for different industries. The emergence of DNNs has challenged traditional detection methods by dynamically analyzing complex component images, and achieving a more convenient and automated detection process. In terms of the execution environment for the detection, as the sources of industrial site data become more diverse and the scale of edge applications expands, traditional cloud computing has some limitations in the transmission of remote model data due to bandwidth bottlenecks and high latency, whereas edge computing with deep learning models deployed on edge devices near terminal data sources rises gradually [10]. However, general model training methods in edge sides have encountered some problems in terms of resource utilization and hardware performance which include limited computing power, insufficient storage space, restricted bandwidth and obvious heterogeneity. These always constrain edge devices, which

* Corresponding author.
E-mail addresses: lijy@st.sdju.edu.cn (J. Li), wangxg@sdju.edu.cn (X. Wang), chenhk@st.sdju.edu.cn (H. Chen), wuzx@st.sdju.edu.cn (Z. Wu), zhuzq@st.sdju.edu.cn (Z. Zhu), cao-jian@cs.sjtu.edu.cn (J. Cao), rbuyya@unimelb.edu.au (R. Buyya).

https://doi.org/10.1016/j.comnet.2025.111592
Received 5 April 2025; Received in revised form 13 July 2025; Accepted 27 July 2025
Available online 5 August 2025

make DNN model training on a single device inefficient or even difficult to execute [11]. In response to the resource efficiency differences caused by device heterogeneity, Humas et al. [12] propose the standardization mechanism of heterogeneous sensing resources. Dynamic performance quantification is utilized to alleviate the fluctuations in resource utilization, but there are still deficiencies in highly dynamic scenarios.

In terms of the theoretical basis of system optimization, the research on distributed and collaborative intelligent task allocation has been continuously advanced. For instance, Shi et al. [13] present multi-agent high-order interaction modeling based on hypergraphs, providing support for resource allocation and task scheduling in complex heterogeneous environments. The related methods also include the multi-objective evolutionary scheduling method in [14], the distributed traffic routing method in [15] and the edge multi-objective prediction in [16]. In addition, Duan et al. [17] propose an initialization-free distributed optimization algorithm for dynamic scheduling of microgrids, which demonstrates the effectiveness of distributed resource allocation in complex network environments, and provides a new idea for the optimization of multi-objective tasks in edge environments. These achievements have laid a theoretical foundation for the subsequent intelligent resource scheduling and efficient collaborative training in heterogeneous edge environments. Although the aforementioned theories and methods provide strong support, deploying deep neural network training on large-scale edge devices still faces many challenges.

To address this challenge, many researchers have begun exploring distributed deep learning training frameworks [18,19] in heterogeneous edge networks, by partitioning DNN models and datasets to train models [20,21] in parallel on numerous edge devices near the data source location. At the same time, Besta and Hoefler [22] conduct a systematic analysis of the concurrent execution modes of complex models such as graph neural networks in parallel and distributed computing environments, which summarizes the characteristics and performance bottlenecks of different parallel paradigms, and provides a reference for the efficient distributed scheduling of subsequent complex neural networks. Efficient distributed training strategies [23–27] play a crucial role in dealing with large model training. Among them, the pipeline parallelism has become an effective method for training models due to its low communication bandwidth requirements and excellent scalability. Seq1F1B [28] effectively reduces idle waiting time during the training process by optimizing pipeline scheduling, further improves the utilization rate of the pipeline. AutoPipe [29] proposes a pipeline parallel automatic configuration method based on reinforcement learning in a shared GPU cluster, which significantly enhances the training adaptability and efficiency under different task and resource states. Asteroid [21] further presents a hybrid pipeline parallelism and fault-tolerant mechanism for heterogeneous edge devices, achieving efficient collaboration and dynamic optimization of resources. The model partitioning and scheduling scheme on the basis of synchronous pipeline proposed by Huang et al. [30] effectively reduces the time for DNN synchronous training in heterogeneous IoT environments.

However, the aforementioned relatively fixed task scheduling strategy fails to fully solve the problem of imbalanced task allocation caused by performance differences between heterogeneous devices, and is further affected by the phenomenon "*bubbles*" [31] in pipeline parallel processing, which refers to the idle waiting time generated by certain devices waiting for data to arrive or waiting for other devices to complete calculations. The idle waiting time between these devices significantly reduces the performance of distributed DNN training and inference tasks deployed on these edge devices with strictly limited storage and execution time.

Aiming at the above issues, this paper puts forward a DQN-GRU guided distributed DNN pipeline training and adjacent scheduling (DGPAS) model, which is based on the synchronous pipeline parallelism algorithm, to solve the distributed training problem in heterogeneous edge networks. In this place, DQN denotes Deep Q-network, and GRU refers to Gated Recurrent Unit. By allocating and scheduling deep neural network computing tasks on the edge side, DGPAS can efficiently achieve real-time running results for the defect detection scenario on the manufacturing production line, and reduce the delay and bandwidth pressure caused by data transmission between the edge and the cloud sides. In the complex environment of heterogeneous equipments, this method can ensure the stability of the detection process. Specifically, this model is mainly divided into two stages. In the preparation stage, we design two network models, Stage Division Network (SDQN) and Device Scheduling Network (GRDQN). The SDQN network partitions the DNN model into different stages by means of layering, and then the GRDQN network matches the divided stages with the edge devices. The optimal partitioning and scheduling strategies are decided through the above two types of networks. During the execution phase, we devise an *adaptive adjacent scheduling* strategy, which can dynamically adjust task allocation based on the resource status and computing power of adjacent heterogeneous devices. This approach effectively alleviates the "bubble" problem caused by device performance differences, ensuring that the bubble rate can be significantly reduced, thus maintains a stable training efficiency.

The main contributions of this paper are as follows:

- DGPAS, a system model for efficient distributed DNN training in heterogeneous edge computing environments, is proposed, which includes the two network models: SDQN and GRDQN in the preparation stage, and the adaptive adjacent scheduling strategy in the execution phase.
- Combining DQN with GRU network to optimize the decision-making process of the two network models, SDQN and GRDQN, so as to obtain the optimal strategy in the model stage division and device scheduling, thereby the training efficiency of the models is effectively improved.
- A new adaptive adjacent scheduling strategy has been designed to dynamically adjust task allocation (i.e., adjust the computing loads between edge devices through time synchronization and load balancing optimization) based on the resource status and computing power of abutting devices, in order to solve the "data dependent bubble" problem caused by device performance differences, ensuring stable and efficient training efficiency even in complex dynamic environments.
- In the preparation stage and scheduling phase of the DGPAS scheme, regarding the training time, our designed network was compared with four baselines and current research methods on five typical models. In the execution phase, the decrease ratio of "bubble rate" and the training time before and after the adaptive adjacent scheduling strategy were analyzed. The effectiveness of the DGPAS system model has been demonstrated through a series of experiments on the DNN average training time and the decrease ratio of "bubble rate".

The rest of this paper is organized as follows. Section 2 reviews the research work in the relevant fields. In Section 3, we provide a detailed design on the proposed distributed DNN training framework (DGPAS) based on pipeline parallelism and DQN algorithms, including the proposed SDQN and GRDQN network models. Section 4 elaborates on the adaptive collaborative scheduling strategy, with a focus on how to reduce bubble effects and improve training efficiency through adjacent device collaboration. In Section 5, the effectiveness of the proposed framework and strategy is evaluated by using various performance indicators. Finally, we summarize the conclusions of this work in Section 6.

2. Related work

Deep neural network models have become increasingly complex over time, and the amount of training data is constantly climbing. It

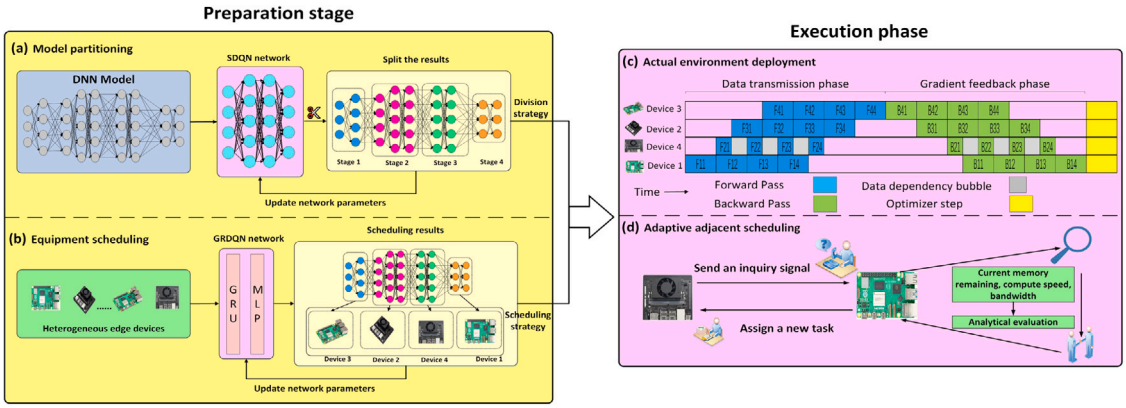


Fig. 1. The network framework of the proposed DGPAS system model.

has become very difficult to train them using only a single edge device. Therefore, relevant researchers hope to develop effective methods to improve the training efficiency and high scalability of DNN models in distributed environments. In recent years, there has been a lot of research on improving the training efficiency of models on heterogeneous edge devices. The distributed training framework based on pipeline parallelism especially improves the performance of large-scale model training by parallelizing computing tasks. We give the following brief reviews on relevant approaches in some existing work.

2.1. Distributed deep learning framework

In traditional distributed deep learning, earlier work mainly focuses on data parallelism and model parallelism. For example, Mobilenetv2 [5] optimizes the running efficiency of the model on mobile devices by designing reverse residuals and linear bottlenecks. PyTorch Distributed [32] and other frameworks provide data parallel based solutions that allow for training across multiple devices.

However, as the model size grows, a single parallelization approach is difficult to meet the training needs on resource-constrained edge devices. Therefore, pipeline parallelism has emerged as a new form of parallelism, which reduces the idle time of each node through pipeline processing. For example, Gpipe [33] reduces the memory usage of large models during training by using pipeline segmentation. And PipeDream [34] further improves training efficiency through asynchronous pipeline parallelism. In addition, in terms of optimizing the resource utilization of models, frameworks such as Megatron-LM [26] and DeepSpeed [24] decrease memory usage through model parallelism and activation recalculation techniques, enabling larger scale models to be trained in limited resource environments.

2.2. Heterogeneous distributed training

The heterogeneity of resources between different devices often creates bottlenecks in model training, so researchers are conducting research on heterogeneous distributed training methods. HeterPS [35] dynamically schedules device resources through reinforcement learning strategies for efficient DNN training in heterogeneous edge device environments. DAPPLE [36] improves training performance on edge devices by combining data parallelism with pipeline parallelism. In addition, Pipemare [37] introduces an asynchronous pipeline parallel mechanism to address latency and bandwidth issues in heterogeneous device environments.

2.3. Heterogeneous edge computing method based on reinforcement learning

In recent years, the emergence of reinforcement learning has provided new ideas for scheduling and resource allocation optimization in distributed systems. For example, scheduling methods based on policy gradient algorithms are gradually emerging in heterogeneous edge networks [38,39], which learn allocation strategies for different tasks between different devices to achieve optimal resource utilization. HeterPS [35] adopts reinforcement learning to optimize the scheduling process, which can effectively address the challenges in heterogeneous edge networks.

The above research still has many shortcomings in the resource utilization, device latency, and overall model training efficiency when facing complex dynamic distributed environments. Therefore, this paper further introduces the GRU network into the DQN algorithm, then combines them to the pipeline parallel DNN training framework, and finally proposes an adaptive adjacent scheduling strategy to achieve efficient distributed training by obtaining the optimal strategy in the early stage and making actual adjustments in the later stage.

3. System model

We design a DGPAS system model that combines reinforcement learning and pipeline parallel strategy, which enables efficient training of DNN models on heterogeneous edge devices.

3.1. The overall framework of the system model

As shown in Fig. 1, the proposed system model consists of the preparation stage and the execution phase, which contain four core modules (a) to (d) as follows.

(a) Model partitioning. We utilize the trained SDQN network, and rationally divide the model into multiple computing stages based on the input DNN model information.

(b) Device scheduling. The heterogeneous computing power, resource status of each edge device as well as the division of the model are all input into the GRDQN network, finally the optimal computing task and device matching strategy are output.

(c) Actual environment deployment. After obtaining the optimal strategy, each stage is deployed to the corresponding real environment based on this strategy.

(d) Adaptive adjacent scheduling. Aiming at the problem of resource waste that occurs during the execution phase, the task allocation is adjusted in real time according to the actual execution situation, thereby the influence of bubbles is reduced, and the training efficiency of the DNN model is improved.

To illustrate the above process more clearly, we draw a simplified version of the system flowchart as shown in Fig. 2.

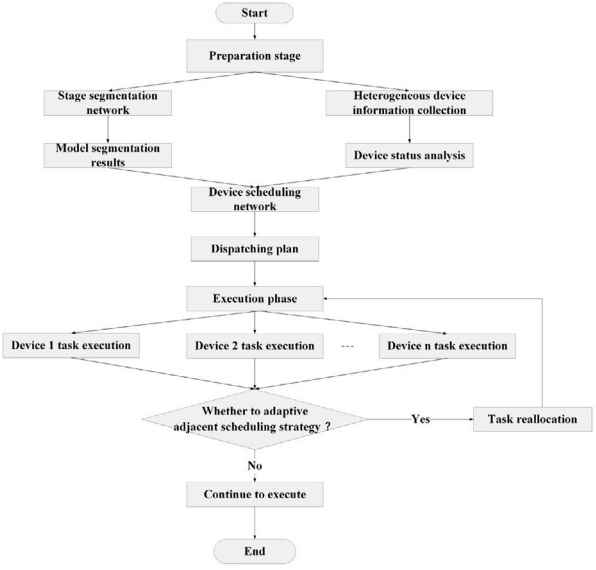


Fig. 2. A simple flowchart of the DGPAS system.

3.2. DQN based synchronous pipeline parallel DNN training framework

For a given DNN model, the pipeline parallel layered partitioning method can be used to split the model into multiple stages and allocate them to different edge devices, thereby achieving distributed training of the model among these devices. Specifically, this process can be modeled as a Markov decision process [40] and effectively solved by DQN [41]. Therefore, the stage division and device scheduling are simulated as neural networks in the DQN algorithm for learning optimization.

3.2.1. Model stage division

Define a DNN model as T, which contains J independent layers. We represent the set of all layers in model T as L, where $L = \{l_j \mid j = 1, 2, \ldots, J\}$. There is the state space S^{div} in the model stage division including all possible states s_t, where t represents the current time. Each state can be represented as $s_t = (R, R_{total})$, where $R = \left(W_j, Params_j, M_j, D_j^{out}\right)$ is the resource usage information, including the computation, parameter, memory usage, and data transmission volume of the jth layer. R_{total} denotes the total resource information of all layers, defined as $R_{total} = \left(W_{total}, Params_{total}, M_{total}, D_{total}^{out}\right)$. The action space A contains all possible actions $a_t = (stp, con)$, where stp represents dividing the model into a new stage after the current layer, and con represents the opposite.

We establish duel neural network - DQN, i.e., current and target value networks, all consisting of an input layer, an output layer and two hidden layers, with 8, 2, 160, 128, and 64 neurons in respective layer. DQN utilizes the Q-learning framework. It models the stage division as a sequential decision-making problem. Neural networks learn state–action mapping through environmental interaction. This makes adaptive learning unnecessary to predefined rules for generating the optimal partitioning strategy by maximizing long-term rewards. As shown in Fig. 3, the information of each layer l_i of the DNN model T is transmitted to the above current value network, and the Q value of the corresponding action is output for each layer. Based on the maximum Q value, the corresponding action is selected, and then the layers of DNN are divided into different stages. Assuming a total of s stages can be divided, and the model partitioning set is denoted as $\mathbf{S} = \left\{S_r\left(j, \ldots, j'\right) \mid j \in J, r = 1, 2, \ldots, s\right\}$.

3.2.2. Introducing GRU network collaborative device scheduling

In a heterogeneous distributed environment, we use the set $d = \left\{d_p \mid p = 1, 2, \ldots, P\right\}$ to represent different edge devices. The system state space S^{sch} in the device scheduling is defined to include all possible states, where each state can be denoted as $s_t = (C_{req}, D_{use})$. Thereinto, C_{req} indicates the computational requirements of the current stage, including the computational workload W_r and memory requirements M_r of that stage; D_{use} denotes the current usage of device resources, including memory M_p, bandwidth B_p, computing power F_p and whether the device has been allocated a phase. The action space A' is defined as all possible actions $a_t' = \left\{\left(S_r \to d_1\right), \left(S_r \to d_2\right), \ldots, \left(S_r \to d_p\right)\right\}$, indicating that the model partitioning block S_r of current stage $r \in [1, s]$ is assigned to a specific device d_p, and ensuring that each device has only one stage that needs to be calculated.

The GRU (Gated Recurrent Unit) neural network dynamically controls the flow of historical information through the dual gating mechanism of update gates and reset gates, and can capture the temporal dependence and sequence characteristics in device scheduling decisions. Through its built-in memory unit, it remembers and processes the information of the previous moment. Thus, this enable the model to fully consider the influence of historical resource allocation and stage dependency when making scheduling decisions currently. Compared with LSTM [42], GRU achieves comparable performance with fewer parameters through a simplified gate structure. Therefore, as depicted in Fig. 4, we have designed a GRDQN device scheduling network, which specifically adopts the GRU network as a part of the neural network, extracts the hidden state h_t using the Eq. (1), and combines it with other states at the current time as the input s' of DQN, so that the model can perform deeper optimization in time, thereby improve the overall scheduling effect.

$$h_t = GRU\left(x_t, h_{t-1}; \phi\right) \tag{1}$$

Where x_t is the input for the current time step, including device resources and stage dependencies, and h_{t-1} is the hidden state of the previous time step.

3.3. Optimize the dual network training framework

3.3.1. Stage division network (SDQN)

A good stage partitioning strategy can effectively improve the training effectiveness of the model, so optimizing the network training can obtain the optimal model parameters δ_{div}^* for the stage partitioning network.

(1) **Initialization.** Initialize the relevant parameters of the stage division network, including the experience replay pool $\mathcal{O}_{div}$, Q network and Qtarget network parameters δ_{div}, δ_{div}^-, etc.

(2) **Exploration and utilization.** By taking the state s_t of each time step t as the input of the neural network and using the ϵ-greedy strategy to select the action a_t, the model is divided into some layers. The action of selecting the probability of ϵ is defined as an exploration of partitioning; $a_t = \max_a Q(s_t, a; \delta_{div})$ is the selected action with the highest Q value in the current state.

(3) **Training and adjustment.** Based on the sufficient interaction between the action and the environment, the feedback R_t and the next state s_{t+1} are obtained during the training process, and $\left(s_t, a_t, R_t, s_{t+1}\right)$ is stored in the experience pool. In order to better guide the selection of actions during the phase division process, we design the immediate reward R_t at each time step t to be either the load balancing reward r_{stp} or the communication volume reward r_{con}, i.e., $R_t \in \{r_{stp}, r_{con}\}$. The specific form is shown in Eqs. (2) and (3).

$$r_{stp} = \begin{cases} r_c & \text{if } W_s > \kappa \cdot F_p \\ -r_c' & \text{if } W_s \le \kappa \cdot F_p \end{cases} \tag{2}$$

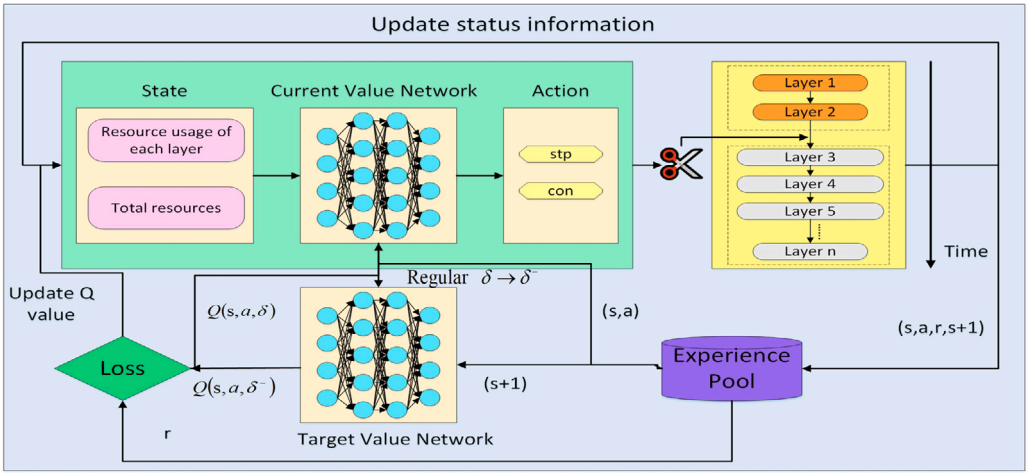


Fig. 3. Stage division flowchart based on DQN.

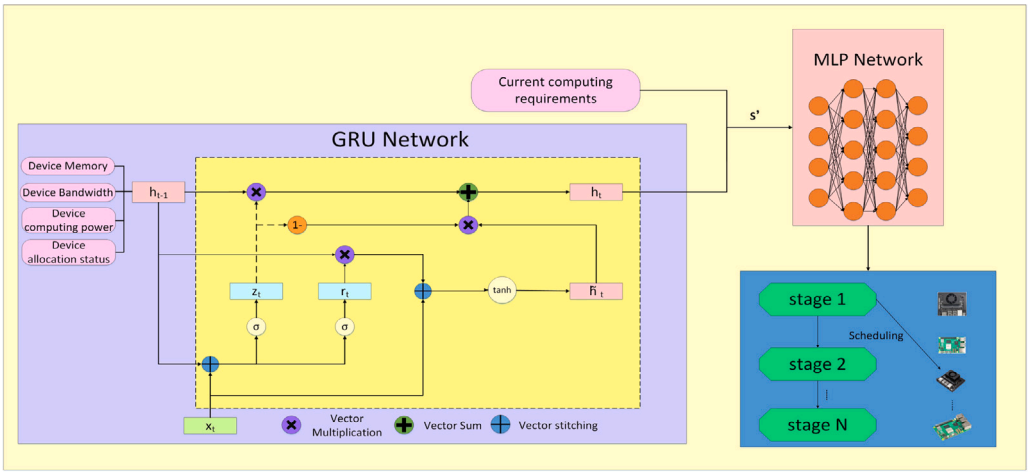


Fig. 4. GRDQN device scheduling network architecture diagram.

where r_c and $-r'_c$ are respectively the positive and negative constant values in relatively extreme cases, and κ denotes stage division threshold parameter. When the model chooses partitioning actions, if the computational cost W_s in the current stage is greater than κ times of computing power F_p and obtains reward r_c, conversely, imposes the penalty value $-r'_c$.

$$r_{con} = \begin{cases} r_f & \text{if } C_s > \varsigma \cdot B_p \\ -r'_f & \text{if } C_s \le \varsigma \cdot B_p \end{cases} \tag{3}$$

where r_f and $-r'_f$ are also respectively the positive and negative constant values in relatively extreme cases, and ς indicates the other stage division threshold parameter. When the model is not split, if the internal communication volume C_s in the current stage is larger than ς times of bandwidth B_p and obtains reward r_f, on the contrary, imposes the penalty value $-r'_f$.

Each layer of the DNN model has its own different computational complexity. If each layer cannot be reasonably divided into different stages, it is easy to cause uneven distribution of computational load, which affects the overall training performance. Therefore, the variance of stage time T_{phase} is taken as the main measurement parameter, as expressed in Eq. (4).

$$T_{phase} = \frac{1}{s}\sum_{r=1}^{s}\left(T_r - T_{Avg}\right)^2 \tag{4}$$

Where T_r is the computation time of the r-th stage, and T_{Avg} denotes the average computation time for all stages. After obtaining the complete phase division strategy, the actually calculated T_{phase} is compared with the average threshold $T_{balance}$, and the final reward R_{div} is allocated to the strategy. The specific form is shown in Eq. (5).

$$R_{div} = \begin{cases} \max\left(5, 6 - T_{phase} \times 10\right) & \text{if } T_{phase} \le T_{balance} \\ -\min\left(3, T_{phase}\right) & \text{if } T_{phase} > T_{balance} \end{cases} \tag{5}$$

If the number of stages divided and the duration of each stage are close to the average time, additional positive rewards will be given. Next, we randomly select a small batch of (s_t, a_t, R_t, s_{t+1}) from the experience pool, and calculate the target value as Eq. (6).

$$Qtarget_t = \begin{cases} R_t \\ \text{if } s_{t+1} \textit{ is termination state}; \\ R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \delta^-_{div}) \\ \text{Otherwise} \end{cases} \tag{6}$$

In order to reduce the difference between the target Q value and the network predicted Q value, DQN uses gradient descent to adjust the network parameters based on the calculated target Q value. By minimizing the loss function $L(\delta_{div})$ as shown in Eq. (7), the network's decisions gradually approach the true Q value:

$$L(\delta_{div}) = \mathbb{E}_{(s_t, a_t, R_t, s_{t+1}) \sim \mathcal{O}_s}\left[\left(Qtarget_t - Q(s_t, a_t; \delta_{div})\right)^2\right] \tag{7}$$

Algorithm 1 SDQN Stage Division Network Algorithm

Input: DNN model parameters R, R_{total}, maximum episodes $Round_1$, thresholds κ, ς, $T_{balance}$
Output: δ^*_{div}: Optimal strategy for stage division
1: **Initialize:** Q-network δ_{div}, target network δ^-_{div}, replay buffer $\mathcal{O}_{div}$
2: **for** episode = 1, $Round_1$ **do**
3: Reset the environment and observe initial state.
4: **for** t = 1,T **do**
5: $a_t \leftarrow$ select a random action with probability ϵ, otherwise $a_t = \max_a Q(s_t, a; \delta_{div})$
6: $R_t \in \{r_{stp}, r_{con}\}, s_{t+1} \leftarrow$ Execute action a_t
7: $R_{div} \leftarrow \begin{cases} \max\left(5, 6 - \frac{1}{s}\sum_{r=1}^{s}\left(T_r - T_{Avg}\right) \times 10\right) \\ -\min\left(3, \frac{1}{s}\sum_{r=1}^{s}\left(T_r - T_{Avg}\right)\right) \end{cases}$
8: $O_{div} \leftarrow$ Store (s_t, a_t, R_t, s_{t+1})
9: Sample a mini-batch from $\mathcal{O}_{div}$ for training
10: $Qtarget_t \leftarrow \begin{cases} R_t \\ R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \delta^-_{div}) \end{cases}$
11: $L(\delta_{div}) \leftarrow \mathbb{E}_{(s_t, a_t, R_t, s_{t+1}) \sim \mathcal{O}_t}\left[\left(Qtarget_t - Q(s_t, a_t; \delta_{div})\right)^2\right]$
12: $\delta^-_{div} \leftarrow$ Periodically update target network parameters with δ_{div}
13: **end for**
14: **end for**
15: **return** δ^*_{div}

After training the network repeatedly, the parameters of the current network are copied to the Qtarget network. When completing $Round_1$ rounds of iterative optimization, the optimal parameters δ^*_{div} for the stage partitioning network are obtained, thus the optimal model stage partitioning strategy is achieved.

The corresponding SDQN algorithm is devised and shown in Algorithm 1. Line 1 initializes the Q-network δ_{div}, target network δ^-_{div}, and replay buffer $\mathcal{O}_{div}$. Lines 2 to 12 iterate each episode until the maximum number of iterations $Round_1$ is reached. Line 3 resets the environment and observes the initial state. Lines 4 to 12 iterate for each time step t. Line 5 randomly selects an action a_t with a probability of ϵ, or $a_t = \max_a Q(s_t, a; \delta_{div})$. Line 6 executes action a_t, obtaining the next state s_{t+1} and the immediate reward R_t. The reward R_t is determined as either r_{stp} or r_{con} based on the executed action a_t. Line 7 calculates the reward R_{div} of the final stage partition network. Line 8 stores the current state, action, reward and next state (s_t, a_t, R_t, s_{t+1}) in the experience buffer O_{div}. Line 9 samples a mini-batch from $\mathcal{O}_{div}$ for training. Lines 10 and 11 calculate the target Q value and loss function respectively to update the optimization network. Line 12 regularly updates the target network parameters δ^-_{div} with δ_{div}. Finally, Line 15 returns the optimal stage division strategy δ^*_{div}.

The time complexity of Algorithm 1 is $\mathcal{O}\left(Round_1 \cdot T \cdot C\right)$, where $Round_1$ is the maximum iteration number, T is the number of time steps and C is the complexity of the neural network execution for each batch update considering forward and backward propagation.

3.3.2. Device scheduling network (GRDQN)

In order to obtain the optimal model parameters δ^*_{sch} for the GRDQN network, this device scheduling network need to be optimized and trained.

1. **Initialization.** The experience replay pool $\mathcal{O}_{sch}$, Q network (δ_{sch}) and Qtarget network parameters (δ^-_{sch}), and regularly synchronized parameters of Q network to Qtarget network are all initialized.
2. **Interaction with the environment.** Input the device information of each time step t into the GRU network to calculate the hidden state h_t, then combine it with other states in the system to form s'_t, and subsequently input it into the Q network. Use ϵ-greedy strategy to select action a_t, where there is a probability of ϵ randomly selecting an action; In other cases, $a_t = \max_a Q(s'_t, a; \delta_{sch})$ selects the action that maximizes the Q value in the s'_t state. After executing an action, a reward and the next state are returned through the environment to learn the value of the action.
3. **Training and Adjustment.** Store the current interaction information $\left(s'_t, a_t, R'_t, s'_{t+1}\right)$ into the experience pool.

Reducing task execution delay and balancing device loads are the two main goals of optimizing device scheduling. Therefore, based on these two factors, we design a reward function as Eq. (8).

$$\begin{aligned} R_{sch} &= \upsilon - \alpha D_t - \beta \sum_{p=1}^{P} L_p^2 \\ &= \upsilon - \alpha \frac{W_r}{F_p + \varepsilon} - \beta \sum_{p=1}^{P}\left(L_{memory}^2 + L_{compute}^2\right) \end{aligned} \tag{8}$$

Where D_t represents the task execution delay at time step t, $\sum_{p=1}^{P} L_p^2$ indicates the balance of device loads, α and β are coefficients used to adjust the weights of delay and load balancing. ε is set to prevent zero division errors. When an edge device meets the requirements of computing power, memory and bandwidth of the tasks simultaneously, an additional positive rewards υ will be given.

Next, a batch of experiences are randomly selected from the experience replay pool, their hidden states h_t and h_{t+1} are calculated, and then the target network is used to calculate the target Q value by Eq. (9).

$$Qtarget_t = R'_t + \gamma \max_{a_{t+1}} Q(s'_{t+1}, a_{t+1}; \delta^-_{sch}) \tag{9}$$

In addition, using mean square error as the loss function $L(\delta_{sch})$ to measure the distance between the target Q value and the estimated Q value as Eq. (10).

$$L(\delta_{sch}) = \frac{1}{2}\sum_{t}\left[\left(Qtarget_t - Q(s'_t, a_t; \delta_{sch})\right)^2\right] \tag{10}$$

Continuously repeating $Round_2$ iterations until the model converges, so that DQN can effectively learn better strategies in complex environments, while reducing the variance during training based on the target network and experience replay, and increasing the stability of learning.

The proposed GRDQN algorithm is designed based on the GRDQN device scheduling model, as shown in Algorithm 2. Lines 1 initializes the hidden state h_t of the GRU network, the parameter of the Q-network δ_{sch}, the target Q network parameter δ^-_{sch}, and the experience replay pool $\mathcal{O}_{sch}$, while employing the ϵ-greedy policy for exploration. Lines 2 to 14 create the outer loop, iterating over each episode until the maximum number of iterations $Round_2$ is reached. At the start of each episode, the environment resets, and the initial state s_0 is initialized. Lines 4 to 13 form the inner loop, iterating over each time step t. Lines 5 and 6 extract the current input x_t, which includes device status D_p, and feed it into the GRU network, calculating the current hidden state. Line 7 calculates the current state s'_t based on the hidden state h_t and the current resource level R_r. Line 8 selects an action a_t based on this state and the ϵ-greedy policy. Line 9 executes action a_t, yielding the reward R_{sch} and the next state s_{t+1}. Line 10 stores the experience $(s_t, a_t, R_{t'}, s_{t+1})$ of the current time step in the experience replay buffer O_{sch}. Line 11 samples a mini-batch from the experience replay buffer O_{sch}, forming the training dataset for the current time step. Lines 12 and 13 compute the target Q-value and the loss function using Eqs. (9) and (10), respectively. Line 14 periodically copies δ_{sch} to the target network δ^-_{sch} to maintain training stability. Finally, Line 17 returns the optimal policy δ^*_{sch}.

The time complexity of Algorithm 2 is $\mathcal{O}\left(\text{Round}_2 \cdot T \cdot \left(H \cdot P + H^2 + B \cdot N_q\right)\right)$, where $Round_2$ denotes the number of iterations in the outer loop. The symbol T represents the number of time steps, H denotes the hidden state dimension of the GRU network, P represents the number of devices, B indicates the batch size and N_q denotes the number of parameters in the Q-network.

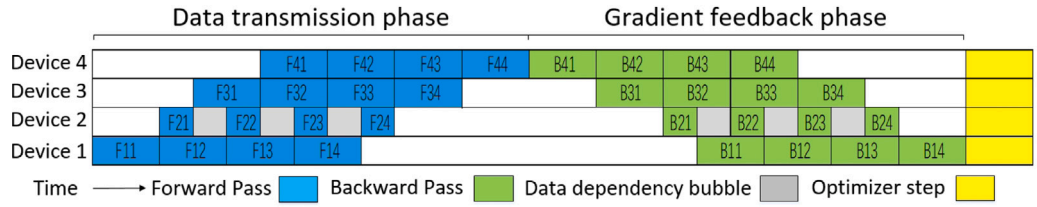


Fig. 5. Data dependency bubble structure diagram.

Algorithm 2 GRDQN Device Scheduling Network Algorithm

Input: Edge devices $d = \{d_p | p = 1, 2, \ldots, P\}$; Maximum number of iterations $Round_2$
Output: δ^*_{sch}: Optimal strategy for device scheduling
Initialize: GRU network for hidden state h_t; Q-network parameters δ_{sch}; target Q-network parameters δ^-_{sch}; replay buffer O_{sch}; ϵ-greedy exploration with $\epsilon \in [0.1, 1.0]$ and decay rate
for episode = 1, $Round_2$ **do**
  Reset environment and initialize initial state s_0
  for t=1,T **do**
    Extract current input $x_t = D$ including device states D_p
    $h_t \leftarrow GRU(x_t, h_{t-1}; \phi)$
    $s'_t \leftarrow$ Use h_t and R_r to get current state
    $a_t \leftarrow$ Use s'_t and ϵ-greedy policy
    $R_{sch}, s_{t+1} \leftarrow$ Execute action a_t
    $O_{sch} \leftarrow$ Store $(s_t, a_t, R_{t'}, s_{t+1})$ in replay buffer
    Sample a mini-batch of transitions from O_{sch}
    $Qtarget_t \leftarrow R'_t + \gamma \max_{a_{t+1}} Q(s'_{t+1}, a_{t+1}; \delta^-_{sch})$
    $L(\delta_{sch}) \leftarrow \frac{1}{2} \sum_t \left[\left(Qtarget_t - Q(s'_t, a_t; \delta_{sch}) \right)^2 \right]$
    Periodically update target Q-network parameters $\delta^-_{sch} \leftarrow \delta_{sch}$
  end for
end for
return δ^*_{sch}

4. Adaptive adjacent scheduling strategy in heterogeneous edge networks

4.1. The data dependency bubbles in pipeline parallel

Heterogeneous edge devices do not have exactly the same execution speed for micro-batches in pipeline tasks. Therefore, during the forward computing process of data transmission, if the device d_p in the r-th stage has a shorter execution time than the device d_{p-1} in the previous stage, due to the dependency relationship between parallel executing tasks in the pipeline, the device d_p cannot immediately perform the calculation of the next micro-batch, but stagnates and waits for the forward result calculated by the device d_{p-1} to be transmitted, as shown in Fig. 5. This stagnation causes the pipeline to be interrupted, which brings in additional bubbles, i.e., data dependency bubbles.

The above phenomenon also exists in the backpropagation process of the gradient feedback stage. If a device completes its calculations faster than the dependent device, it needs to wait for the dependent device to transmit the gradient results, and its backpropagation calculation will also be forced to wait, further exacerbating the generation of data dependency bubbles and reducing the overall efficiency of the model training.

Assuming that the micro-batches in each stage are equal T^r_f and T^r_b, which respectively represent the forward and backward propagation time of each micro-batch in stage r, where suppose $T^r_b = T^r_f$. In heterogeneous edge networks, the significant difference in device performance is the inevitable introduction of additional bubble time T_{pb}. In order to more accurately model the bubble consumption area S_{tpb}, we give the mathematical expression as Eq. (11).

$$S_{tpb} = T_{pb} + \sum_{r=1}^{s} \left[(s - r) \cdot \left(\int_0^{T^r_f} e^{-\lambda x} dx + \log(1 + T^r_b) \right) \right] \tag{11}$$

Among them, $\int_0^{T^r_f} e^{-\lambda x} dx$ is used to describe the cumulative delay introduced by heterogeneous device computing performance within the forward propagation time interval $[0, T^r_f]$. λ represents a parameter that measures the degree of device heterogeneity (i.e., the larger the λ, the more significant the performance difference and the faster the attenuation). The exponential decay model is used to reflect that as the calculation progresses, the delay introduced by the new unit time gradually decreases. $\log(1 + T^r_b)$ uses a convex function to model the nonlinear growth effect of backpropagation time T^r_b on bubble accumulation, that is, as the backpropagation time increases, the growth rate of pipeline bubbles shows an increasing trend.

4.2. Adaptive adjacent scheduling strategy

To avoid the accumulation of data dependency bubbles in complex dynamic environments, an adaptive adjacent scheduling strategy is proposed. When there is a significant difference in device performance, data dependency bubbles T_{pb}, dynamic delay term $\int_0^{T^r_f} e^{-\lambda x} dx$, and nonlinear cumulative effect $\log(1 + T^r_b)$ which will aggravate the decrease in training efficiency of DNN models. The adaptive adjacent scheduling strategy dynamically adjusts the computing loads between devices through time synchronization and load balancing optimization. Specifically, when a device completes a task ahead of scheduling, it actively sends an assistance request signal, including remaining memory (RM), computing speed and bandwidth information, to inquire whether neighboring dependent devices need assistance. For each micro-batch, a corresponding checkpoint is designed. When the checkpoint is reached during the calculation process, it stops to check whether assistance requests are received from neighboring devices. If a request is received, the data will be reallocated to balance execution time and reduce bubbles caused by device idling.

Taking an example about the specific process of the above scheduling strategy, and expounding the performance difference between device 1 and device 2 during the data transmission phase as depicted in Fig. 6. In the course of the initial forward propagation process, when device 2 completes the calculation of the first micro-batch, device 1 has not yet completed the calculation of the second micro-batch. At this point, device 2 actively sends an assistance request to device 1 as it has not received the corresponding data transmitted by device 1. When device 1 reaches the preset checkpoint, it checks whether it has received the assistance signal from device 2. If a signal is received, device 1 will reassign the remaining tasks and assign some tasks to device 2. After the first assistance, if a similar phenomenon occurs again when device 2 completes the third micro-batch, the assistance strategy will be activated again to reassign the corresponding tasks. Through the continuous collaboration of adaptive adjacent scheduling strategies, the proportion of bubbles in the forward propagation process gradually decreases, and the overall execution efficiency is significantly improved. In heterogeneous edge networks, the advantages of this strategy will be more prominent, which achieve maximum resource utilization and efficient training processes.

Define the data volume of device $d_{p'}$ on the kth micro-batch as $M_{p',k}$, where $p' \in P$, $k \in N$. while P represents the set of all devices involved in scheduling, and N is the micro-batch index set for the current stage. During parallel execution on different devices,

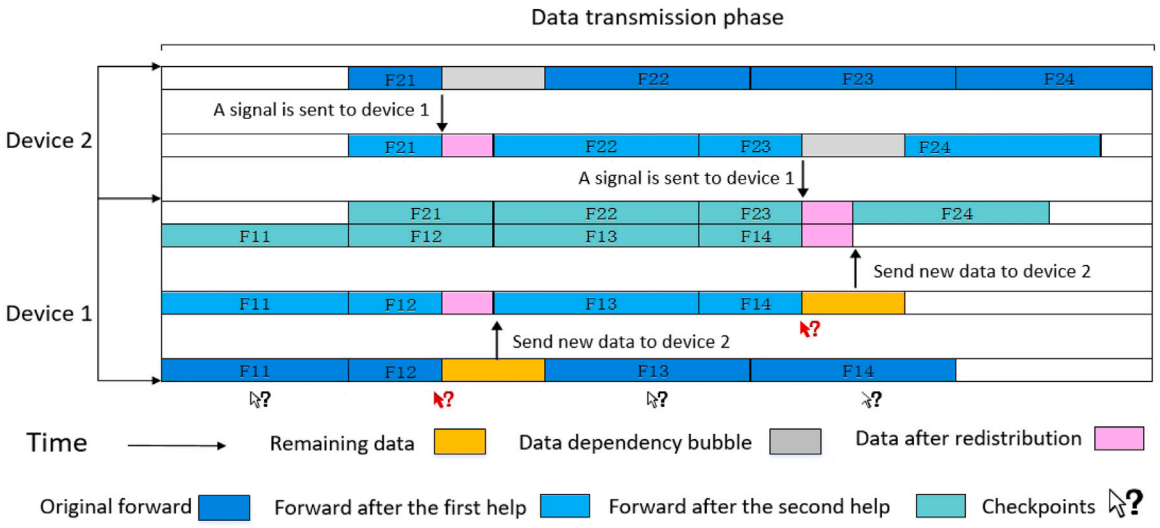


Fig. 6. Collaborative scheduling flowchart between adjacent devices.

if a data dependency bubble occurs during the same time period, the corresponding auxiliary strategy will be triggered to reallocate the remaining data $M_{p',k} - tF_{p'}$ from the tth batch of device $d_{p'}$, where t is the processing time consumed and $F_{p'}$ is the computing performance of device $d_{p'}$.

To achieve more synchronized execution time between devices $d_{p'}$ and d_p after data reallocation, an optimization strategy $\varphi\left(p' \to p, A_p\right)$ is designed based on the least squares method. This strategy represents the data size A_p of transmission from device $d_{p'}$ to device d_p. To avoid unreasonable task allocation, we give the constraint conditions to ensure that the transmission volume is within the allowable range of device margin RM_p, and the received data Rec_p is 0. Determining the optimal strategy for adjacent scheduling is just to solve the optimization problem as Eq. (12).

$$\begin{aligned} \min_{\varphi(p' \to p, A_p)} \quad & \left(\frac{M_{p',k} - tF_{p'} - A_p}{F_{p'}} - \frac{A_p}{F_p}\right)^2 \\ \text{s.t.} \quad & 0 \le A_p \le \min\left(M_{p',k} - tF_{p'}, RM_p\right), \\ & Rec_p = 0 \end{aligned} \tag{12}$$

Algorithm 3 shows dynamic task scheduling process in acceleration phase for the adaptive neighbor scheduling strategy. Line 1 initializes the strategy set φ to an empty set and sets A_p of all devices to 0. Lines 2 to 14 iterate each small batch k, and lines 3 to 13 iterate each device d_p in parallel. Line 4 indicates that each device d_p is calculated to the checkpoint for inspection. Line 5 checks whether there is an assistance signal. If the condition is met, lines 6 to 7 obtain the value of A_p by using the Eq. (12) and its constraints. Line 8 updates the strategy φ by adding the data transmission $(p' \to p, A_p)$ from device $d_{p'}$ to d_p. Lines 9 to 10 update the small batch size of devices $d_{p'}$ and d_p, and the value of Rec_p accordingly. Line 12 continues the calculation. Finally, line 15 returns the strategy φ.

The time complexity of Algorithm 3 is mainly affected by the number n of devices involved in the scheduling, so it can be considered to be $\mathcal{O}(n)$.

In order to illustrate the effectiveness of the proposed adaptive adjacent strategy, we present Theorem 1 and its proof as follows.

Theorem 1. *In complex dynamic heterogeneous edge networks, the use of adaptive adjacent scheduling strategies can effectively reduce data dependency bubbles generated during pipeline parallel processes within the given iteration cycle, and significantly improve the training efficiency of DNN models.*

Algorithm 3 Dynamic Task Scheduling in Acceleration Phase

Input: $M_{p',k}$: Size of the k-th micro-batch for device $d_{p'}$; $F_p, F_{p'}$: Computing performance of devices d_p and $d_{p'}$; RM_p: Remaining memory for device d_p
Output: φ: Optimized adaptive adjacent scheduling strategy

1: Initialize: $\varphi \leftarrow \emptyset, A_p = 0, \forall p$
2: **for** each k **do**
3: **for** each d_p in parallel **do**
4: Compute until checkpoint
5: **if** an assistance signal is detected at the checkpoint **then**
6: $\min_{A_p} \left(\frac{M_{p',k} - tF_{p'} - A_p}{F_{p'}} - \frac{A_p}{F_p}\right)^2$
7: s.t. $0 \le A_p \le \min(M_{p',k} - tF_{p'}, RM_p),\ Rec_p = 0$
8: $\varphi \leftarrow \varphi \cup \{(p' \to p, A_p)\}$
9: $M_{p',k} \leftarrow M_{p',k} - A_p,\ M_{p,k} \leftarrow M_{p,k} + A_p$
10: $Rec_p \leftarrow Rec_p + A_p/F_p$
11: **end if**
12: Resume the computing
13: **end for**
14: **end for**
15: **return** φ

Proof. Defining the total duration of a training process as T, where $t' \in [0, T]$, and assuming that the adjacent collaborative scheduling strategy between device $D_{p'}$ and D_p within time slice t' is $\xi^{t'}_{p'p}\left(p' \Rightarrow p, A_p\right)$. Based on the adaptive scheduling strategy and Eq. (12), let $\theta_{t'}$ represent the bubble reduction rate caused by initiating collaborative scheduling between adjacent devices within time slice t':

$$\theta_{t'} = 1 - \frac{\left|\max\left(\frac{M_{p',k} - t \cdot F_{p'} - A_p}{F_{p'}}, \frac{A_p}{B_{p'}}\right) - \frac{A_p}{F_p}\right|}{T_{pb}}$$

Accumulating the decrease in the bubble rate caused by the activation strategy in all time slices during the cycle, and then calculating the proportion of additional bubbles:

$$\begin{aligned} T_{pb} &\le T^0_{pb} \cdot \prod_{t'=1}^{T} \left(1 - \theta_{t'}\right) \\ &\le \frac{T^0_{pb}}{T_{pb}} \cdot \left|\max\left(\frac{M_{p',k} - t \cdot F_{p'} - A_p}{F_{p'}}, \frac{A_p}{B_{p'}}\right) - \frac{A_p}{F_p}\right| \end{aligned}$$

The above inequality verification shows that when the number of bubbles in each time slot decreases, the final proportion of additional bubbles T_{pb} will also be lower than the initial proportion of bubbles, where T^0_{pb} represents the initial additional bubbles when the adjacent

scheduling strategy ξ is not introduced. When $\theta_{t'}$ approaches 1 infinitely, the bubbles in each time slot also approach 0 infinitely, as described by the formula below:

$$\lim_{\theta_{t'}\to 1} T_{pb} \le T_{pb}^{0} \cdot \prod_{t'=1}^{T}(1-\theta_{t'}) = T_{pb}^{0} \cdot 0 = 0$$

When T_{pb} approaches 0 infinitely, the idle time caused by data dependency is reduced, thereby improving the training efficiency of the DNN model. Specifically, the training efficiency η of the model is represented by the following formula:

$$\lim_{T_{pb}\to 0} \eta = \frac{T_{pb}^{0} - T_{pb}}{T_{pb}^{0}} = 1 - \frac{0}{T_{pb}^{0}} = 1$$

The theoretical effectiveness of the adaptive adjacent scheduling strategy has been verified based on the above derivation.

5. Performance appraisal

In this section, we conducted extensive experiments to validate the effectiveness of the DGPAS system model. At first, the performance evaluation indicators related to the research objectives were defined, and the composition of the experimental platform and the relevant experimental parameters were described in detail. Then, for the experiments of the preparation stage, we compared our method with the baseline algorithm and advanced methods previously studied, and carried out multiple experiments under the different parameter conditions for comparative analysis. Next, for the experiments of the execution phase, we validated the effectiveness of our strategy by comparing the decrease in bubble rate and the improvement in training time before and after the proposed strategy. Finally, we summarized the experimental results and discussed their impact on practical applications. Our experimental code is available at https://github.com/Lilili214/DGPAS.

5.1. Definition of performance evaluation indicators

To evaluate the distributed training performance of the DGPAS model in heterogeneous edge environments, the experimental evaluation mainly adopts the following two core indicators, whose settings directly correspond to the goals of "improving training efficiency and alleviating the bubble effect" proposed in the introduction of this paper.

5.1.1. The total training time of the model

In synchronous pipeline parallelism, the training process is divided into s stages, and each stage runs on an edge node. The entire round of training tasks is divided into M micro-batches, and each micro-batch enters the pipeline in sequence. The parameter update time can almost be ignored because it is very short here. Therefore, the total training duration T_{total} is defined as follows.

$$T_{\text{total}} = t_{\text{finish}}^{(M,s)} - t_{\text{start}}^{(1,1)} \tag{13}$$

Where $t_{\text{start}}^{(1,1)}$ represents the time when the first micro-batch starts processing in the first stage, and $t_{\text{finish}}^{(M,s)}$ indicates the time when the last micro-batch is completed in the last stage. M denotes the total number of micro-batches, and s is the number of pipeline stages.

5.1.2. Bubble rate during the training process

The bubble rate is used to measure the proportion of idle resources of heterogeneous devices under data dependency constraints, which reflects the utilization rate of the pipeline parallel scheduling strategy. Its definition refers to the bubble modeling in Eq. (11), that is as follows.

$$R_{\text{bubble}} = \frac{S_{tpb}}{s \cdot T_{\text{total}}} \tag{14}$$

Where S_{tpb} represents the size of the bubble area, s denotes the number of pipeline stages, and T_{total} is the total training time. The lower the bubble rate is, the higher the resource utilization rate is, and the better the training efficiency is.

Fig. 7. The experimental edge network of the DGPAS system.

5.2. Experimental setup

5.2.1. Composition of experimental platform and related experimental parameters

Based on the basic principle of pipeline parallelism, we hope to adaptively divide the DNN model into different stages when facing dynamic and complex environments, and perform optimal scheduling on heterogeneous edge devices. Therefore, we built an experimental platform as shown in Fig. 7, which includes three different types of edge devices connecting to the same local area network through a switch. The detailed information about device names, CPUs, GPUs and memory is described in Table 1.

To verify the effectiveness of the proposed scheme, we select five classic DNN image classification models, AlexNet [43], VGG19 [44], InceptionV3 [45], ResNeXt101 [46], and RegNet-200 MF [47], for multiple experiments on our experimental platform. As shown in Table 2, each model contains the key information such as its name, parameter count, memory and floating-point numbers. For the selection of models, we focus on the representativeness of different complexities and scales to cover a variety of network structures ranging from lightweight to large-scale. These five models are complementary in terms of parameter scale and actual resource requirements, which is conducive to the systematic analysis of the applicability and robustness of the DGPAS scheme in diverse scenarios.

The actual execution environment is full of variable uncertainty, so we performed multiple micro-batch processes on each edge device for the five models mentioned above, and collected the actual execution time, memory usage and CPU utilization of each layer of each model. Specifically, each of edge devices was pre-trained 5 times, followed by recording the average of 10 runs of data. We conducted multiple simulation experiments using the collected data as actual parameters of the real environment. In the experiments, the size of the mini-batch was set as 256. Each mini-batch was divided into 8 micro-batches, and the size of each micro-batch was 32. This setting takes into account both the stability of parameter update and the efficiency of pipeline parallelism, and can effectively improve the system throughput rate and mitigate the impact of bubbles. Different models were trained multiple times, and the average values were taken as the final results to ensure the stability of the experimental data. At the same time, we verified that the adaptive adjacent assistance strategy can effectively reduce the bubble rate and improve the training efficiency through real parallel execution between devices with different micro-batch sizes.

5.2.2. The overview of compared methods

All the methods being compared are described below as follows.

Table 1
Hardware types and specifications for running experiments.

Hardware	CPU	GPU	Memory	Number
Jetson Orin Nano	6-core ARM Cortex-A78AE @ 1.5 GHz	NVIDIA Ampere architecture 1024 × NVIDIA CUDA Cores 32 × 3rd Gen Tensor Cores	8 GB 128-bit LPDDR5 68 GB/s	2
Jetson Xavier NX	6-core NVIDIA Carmel ARM v8.2 64-bit @ 1.4 GHz	NVIDIA Volta architecture 384 × NVIDIA CUDA Cores 48 × Tensor Cores	8 GB 128-bit LPDDR4x 51.2 GB/s	1
Raspberry Pi 5	Broadcom BCM2712 quad-core Arm Cortex A76 processor @ 2.4 GHz	/	8 GB RAM LPDDR4x	12

Table 2
Five typical DNN image classification model specifications.

Model name	Params	Memory	Flops
AlexNet	61,100,840	4.19 MB	715.54MFlops
VGG19	143,667,240	119.34 MB	19.67GFlops
InceptionV3	23,834,568	70.63 MB	5.73GFlops
ResNext101	88,791,336	276.02 MB	16.49GFlops
RegNet-200MF	21,152,810	7.90 MB	2.97GFlops

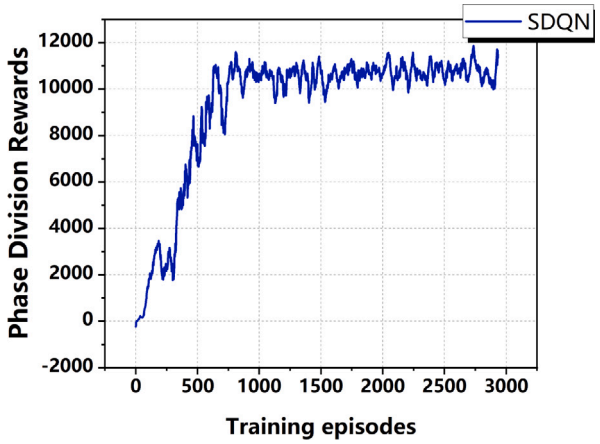


Fig. 8. The reward curve of SDQN network.

DGPAS strategy: The method proposed in this paper analyzes and evaluates the different parameters of the DNN models and the states of heterogeneous devices, and combines GRU to design two networks for stage partitioning and device scheduling. Based on the DQN algorithm, the network parameters are iteratively optimized to ultimately obtain the global optimal solution for model partitioning and device scheduling.

Dynamic Programming (DP) [48]: Using dynamic programming algorithms to segment and schedule tasks for the DNN models, and the multi-stage optimization is achieved, i.e., reaching relative load balancing between stages.

Particle Swarm Optimization (PSO) [49]: The partitioning and scheduling of the DNN models are encoded as the position and velocity vectors of the particles. Using the particle swarm optimization algorithm, each particle continuously adjusts the solution based on its own historical experience and the optimal solution of the group.

Genetic Algorithm (GA) [50]: Encoding the partitioning and scheduling of the DNN models into chromosomes, initializing the population to generate multiple schemes, and optimizing the chromosomes through selection, crossover and mutation operations, and using the fitness function to evaluate so as to obtain the optimal solution.

Proximal Policy Optimization (PPO) [51]: Proximal policy optimization treats the partitioning and scheduling of DNN models as a reinforcement learning problem, and searches for the optimal solution by continuously alternating between sampling and optimizing the strategy, and constraining the strategy update amplitude.

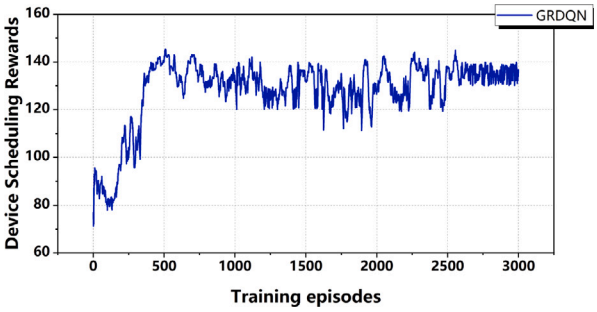


Fig. 9. The reward curve of GRDQN network.

PG-MPSS [30]: It is a policy-based reinforcement learning algorithm that encodes the different parameters of the model in high dimensions, replicates the slowest stage, and uses the policy gradient algorithm to optimize and update for finding the optimal solution.

AutoPipe [29]: It is a framework for pipeline parallel training in a shared GPU cluster. This method is based on reinforcement learning. It conducts high-precision modeling of the actual communication and computing processes, and utilizes multi-step decision-making for intelligent partitioning and allocation.

Seq1F1B [28]: It proposes pipeline parallel scheduling based on sequence partitioning. By further refining the schedulable units, it reduces pipeline bubbles during the training process, and optimizes the utilization of computing resources.

5.3. Performance analysis

5.3.1. Convergence analysis of SDQN and GRDQN network

Taking VGG19 as an example, we conducted a convergence analysis of the SDQN network and the GRDQN network. As depicted in Fig. 8, we can see that the reward value of the SDQN network shows a clear upward trend in the early stage of training, which indicates that the network can quickly learn a better stage division strategy in this stage. As the training progresses, the reward value begins to fluctuate to a certain extent. Although the fluctuation range is large, it does not show a downward trend as a whole, but enters a relatively stable stage. Similarly, we can see from Fig. 9 that the reward value of the GRDQN network also shows a clear upward trend in the early stage of training, i.e., quickly learning the optimal scheduling plan of the devices, and then enters a fluctuation period. Through continuously iterative learning, the rewards finally stabilize at around 130.

Based on the two graphs above, we can conclude that both the SDQN and GRDQN networks demonstrate good convergence during the training processes. Through multiple iterations, the system ultimately learns the optimal stage partitioning and device scheduling strategies, and achieves relatively stable performance levels in the later stages of training.

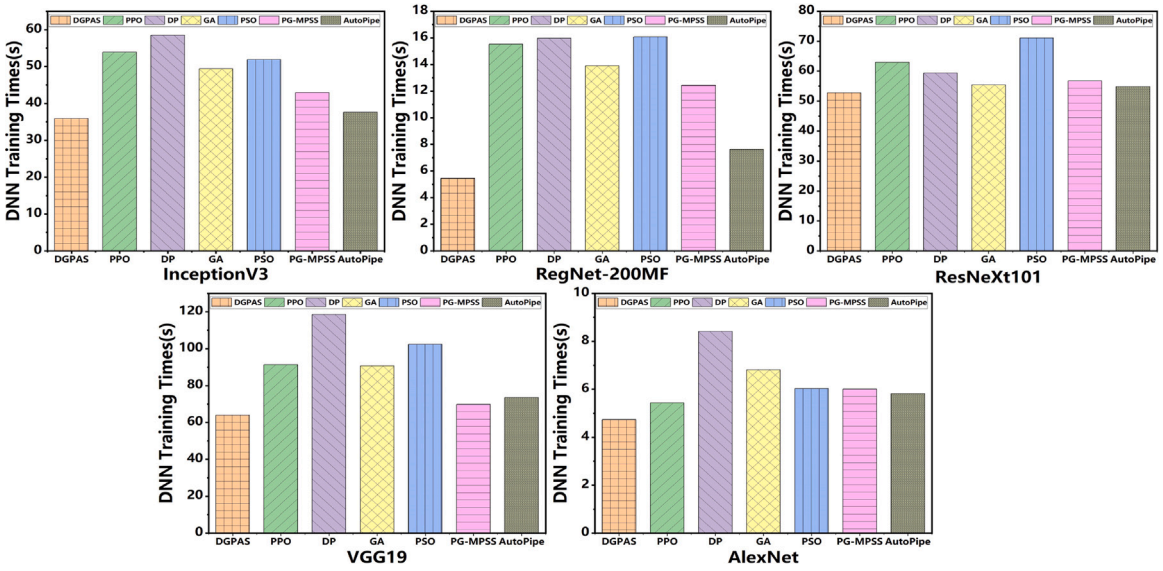


Fig. 10. The training time of different models.

5.3.2. Comparative analysis of DGPAS strategy and other algorithms

In order to demonstrate the effectiveness of this approach in improving the training performance of DNN models, comparative experiments were conducted on five typical DNN image classification models (i.e., AlexNet, VGG19, InceptionV3, ResNeXt101 and RegNet-200MF) by using DGPAS, four typical baseline strategies (i.e., DP, PSO, GA and PPO), as well as the improvement schemes PG-MPSS and AutoPipe in related fields. The average values of 10 experimental results were calculated to obtain five sets of comparative results as shown in Fig. 10.

As depicted in Fig. 10, we can see that DGPAS has achieved significant training time optimization effects than other methods. Taking the AlexNet model as an example, DGPAS reduces the training time by 21.3% and 18.7% respectively compared with the two improved methods, PG-MPSS and AutoPipe, further the time is decreased by 43.7%, 30.6%, 21.6% and 12.9% respectively than the four classic methods of DP, GA, PSO and PPO. Under the ResNeXt101 model, DGPAS's training time is reduced by 7.0%, 3.7%, 11.1%, 4.8%, 25.7% and 16.1% respectively compared with PG-MPSS, AutoPipe, DP, GA, PSO and PPO. For VGG19, the time is decreased by 8.4%, 12.97%, 46.0%, 29.4%, 37.5% and 29.9% respectively compared with other approaches. Under InceptionV3, the proportions of the reduction are 16.3%, 4.5%, 38.0%, 27.2%, 30.8% and 33.4% respectively. On RegNet-200MF, DGPAS can also effectively decrease the training time compared with the above-mentioned methods. Therefore, whether facing traditional classical algorithms or multiple improved methods in recent years, DGPAS can effectively improve the training efficiency under different network structures.

5.3.3. Comparative analysis of bubble rate under adaptive adjacent scheduling strategy

The model partitioning and device scheduling schemes from static training often have difficulty in maintaining optimal states under heterogeneous computing environments. Specifically, the load and memory usage of edge devices will constantly change over time, which may lead to fluctuations in the computing power of heterogeneous devices. The bandwidth and delay indicators of the networks may also change due to factors such as signal interference, leading to insufficient resource utilization and decreased training efficiency. Therefore, our strategy can dynamically adjust the bubble problem during execution by continuously reallocating data between adjacent devices.

To verify the effectiveness of the proposed strategy, we conducted experiments on five typical DNN image classification models: ResNeXt101, VGG19, InceptionV3, RegNet-200MF and AlexNet. We tested the changes in the decrease ratio of "bubble rate" before and after the adaptive adjacent scheduling strategy under different number of mini-batches, as shown in Fig. 11. It can be seen that as the number of mini-batches continues to increase, the decrease ratio of "bubble rate" before and after the adaptive adjacent scheduling strategy shows a significant upward trend, indicating that our strategy can effectively reduce the number of bubbles generated during actual execution.

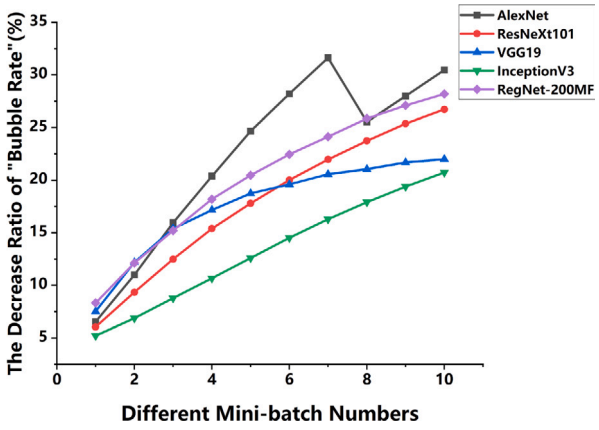


Fig. 11. Under the five typical models, the decrease ratio of "bubble rate" before and after the strategy of different mini-batch numbers changed.

5.3.4. Comparative analysis of training efficiency under adaptive adjacent scheduling strategy

Because the "bubble" problem is actually the device idle time caused by data dependency during the execution process, the continuous adjustment of the adaptive adjacent scheduling strategy can effectively reduce the idle waiting time of the devices, thereby further improving the training efficiency of the DNN models. As shown in Fig. 12, for five typical DNN image classification models, namely ResNeXt101, VGG19, InceptionV3, RegNetX200MF and AlexNet, we compared the training time performances of adopting our optimal partition scheduling (DGPAS), the optimization strategy introducing adaptive adjacent scheduling (DGPAS_Co), and the advanced pipeline parallel method (Seq1F1B) in the current distributed training field.

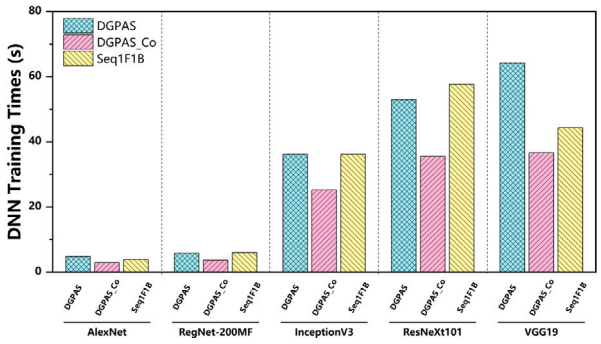


Fig. 12. Comparison of training time using our DGPAS_Co and not using it under different models.

Table 3
Comprehensive comparison of average training time (s) for all methods on five models.

Method	AlexNet	ResNeXt101	VGG19	InceptionV3	RegNet-200MF
DGPAS	**4.73**	**52.77**	**63.96**	**35.94**	**5.45**
PG-MPSS	6.01	56.75	69.81	42.94	12.43
DP	8.41	59.39	118.50	58.50	15.97
GA	6.81	55.41	90.63	49.39	13.89
PSO	6.03	71.04	102.34	51.92	16.07
PPO	5.43	62.95	91.30	53.92	15.52
AutoPipe	5.82	54.83	73.52	37.63	7.62
DGPAS_Co	**2.96**	**35.56**	**36.66**	**25.28**	**3.74**
Seq1F1B	3.8976	57.6603	44.3120	36.1771	6.0470

By comparing the five groups of experiments in Fig. 12, we can find that there are significant differences in the training efficiency of the three methods on all models. Compared with DGPAS, the training time of DGPAS_Co with the adaptive adjacent scheduling reduces by 38.91%, 36.12%, 30.11%, 32.82% and 45.95% respectively on AlexNet, RegNetX200MF, InceptionV3, ResNeXt101 and VGG19. Compared with Seq1F1B, DGPAS_Co also saved 24.0%, 38.2%, 30.1%, 38.3% and 17.3% of the training time respectively. These results demonstrate the superiority of our method in distributed training.

To demonstrate the performance of the proposed method in enhancing the efficiency and adaptability of DNN distributed training in edge environments, we have summarized average training time of all compared methods, as shown in Table 3. In terms of model partitioning and scheduling strategies, compared with other typical or improved optimization algorithms, the average training time of DGPAS on each model is the lowest. This not only verifies the outstanding performance of the scheme we proposed in terms of balanced resource utilization and execution efficiency, but also further supports the original intention of the research, that is, the existing methods have obvious bottlenecks in heterogeneous and resource-constrained scenarios. Furthermore, in response to the practical challenges such as dynamic resource fluctuations during the distributed training process, the adaptive adjacent scheduling strategy introduced by DGPAS_Co effectively avoids the problem that the "optimal" strategy is no longer optimal due to the complex dynamic environment during the training process of the DNN model. It is significantly superior to the original DGPAS method and other improved distributed training methods. This further enhances the training efficiency and system robustness.

6. Conclusions and future work

In response to the time-consuming training and scheduling challenges arisen from using DNNs in industrial product detection scenario, the DGPAS system model is proposed to address the issues of low resource utilization, device delay and low training efficiency in distributed training of DNN models in heterogeneous edge networks. This model constructs the key DNN model partitioning and device scheduling as two collaborative networks in pipeline parallelism, and uses the DQN algorithm for learning optimization. Experiments have proved that the proposed method has the less training time and lower "bubble rate" (i.e, idle waiting time between devices) than existing methods in various typical DNN models and real heterogeneous edge environments. Moreover, it can ensure the stability of the detection process under the heterogeneous performance of devices and the dynamic changes of the network.

In future applications of industrial product image detection, such as in automotive component manufacturing plants, it is necessary to conduct real-time defect detection on the products (such as gears and bearings) on the production line. The edge devices should have a high processing speed, and the algorithms should have a high detection accuracy rate. This can be achieved through the method of the DNN pipeline parallel training. By applying our improved method, we divide the backbone network such as RestNet into 4 stages, and the detection head uses YOLO. The gradient calculations can be simultaneously performed on multiple micro-batches using the GPUs of different edge devices. In terms of some key performance indicators such as the training speed, the accuracy rate of normal product identification, recall rate, and F1 value for crack detection, our approach will show a significant improvement.

However, this study still has certain limitations. The current methods are mainly aimed at medium and small-scale heterogeneous scenarios, and the scalability in extremely large-scale device clusters still needs to be further verified and improved. Further work will be oriented towards more deep learning models and diverse heterogeneous computing power platforms, so as to further enhance the generalization and robustness of DGPAS in different models and application scenarios, and achieve more flexible and efficient distributed training.

CRediT authorship contribution statement

Jiayi Li: Writing – original draft, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Xiaogang Wang:** Writing – review & editing, Resources, Project administration, Methodology, Funding acquisition. **Haokun Chen:** Validation, Software. **Zexin Wu:** Validation, Software. **Ziqi Zhu:** Visualization, Validation. **Jian Cao:** Resources, Project administration, Funding acquisition. **Rajkumar Buyya:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported in part by National Natural Science Foundation of China (Granted Number 62072301), in part by Shanghai Science and Technology Innovation Action Plan (Granted Number 22ZR1425300), and in part by Program of Technology Innovation of the Science and Technology Commission of Shanghai Municipality (Granted Number 21511104700).

Data availability

The research code for this paper is illustrated in the experimental section of the paper, linked to the share base of the author's github account.

References

[1] M.W. Hridoy, M.M. Rahman, S. Sakib, A framework for industrial inspection system using deep learning, Ann. Data Sci. 11 (2) (2024) 445–478, http://dx.doi.org/10.1007/s40745-022-00437-1.

[2] J. Lu, S.-H. Lee, Real-time defect detection model in industrial environment based on lightweight deep learning network, Electronics 12 (21) (2023) http://dx.doi.org/10.3390/electronics12214388.

[3] D.G. Pena, D.G. Perez, I.D. Blanco, J.M. Juarez, Exploring deep fully convolutional neural networks for surface defect detection in complex geometries, Int. J. Adv. Manuf. Technol. 134 (1–2) (2024) 97–111, http://dx.doi.org/10.1007/s00170-024-14069-7.

[4] D.-Y. Jung, Y.-J. Oh, N.-H. Kim, A study on GAN-based car body part defect detection process and comparative analysis of YOLO v7 and YOLO v8 object detection performance, Electronics 13 (13) (2024) http://dx.doi.org/10.3390/electronics13132598.

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, MobileNetV2: Inverted residuals and linear bottlenecks, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520, http://dx.doi.org/10.1109/CVPR.2018.00474.

[6] J. Villalba-Diez, D. Schmidt, R. Gevers, J. Ordieres-Mere, M. Buchwitz, W. Wellbrock, Deep learning for industrial computer vision quality control in the printing industry 4.0, Sensors 19 (18) (2019) http://dx.doi.org/10.3390/s19183987.

[7] N. Tuyen Le, J.-W. Wang, M.-H. Shih, C.-C. Wang, Novel framework for optical film defect detection and classification, IEEE Access 8 (2020) 60964–60978, http://dx.doi.org/10.1109/ACCESS.2020.2982250.

[8] C. Li, X. Zhang, Y. Huang, C. Tang, S. Fatikow, A novel algorithm for defect extraction and classification of mobile phone screen based on machine vision, Comput. Ind. Eng. 146 (2020) http://dx.doi.org/10.1016/j.cie.2020.106530.

[9] G. Tello, O.Y. Al-Jarrah, P.D. Yoo, Y. Al-Hammadi, S. Muhaidat, U. Lee, Deep-structured machine learning model for the recognition of mixed-defect patterns in semiconductor fabrication processes, IEEE Trans. Semicond. Manuf. 31 (2) (2018) 315–322, http://dx.doi.org/10.1109/TSM.2018.2825482.

[10] D.H. Nam, A comparative study of mobile cloud computing, mobile edge computing, and mobile edge cloud computing, in: 2023 Congress in Computer Science, Computer Engineering, and Applied Computing, CSCE, 2023, pp. 1219–1224, http://dx.doi.org/10.1109/CSCE60160.2023.00204.

[11] M.M.H. Shuvo, S.K. Islam, J. Cheng, B.I. Morshed, Efficient acceleration of deep learning inference on resource-constrained edge devices: A review, Proc. IEEE 111 (1) (2023) 42–91, http://dx.doi.org/10.1109/JPROC.2022.3226481.

[12] Q. Hua, D. Yang, S. Qian, J. Cao, G. Xue, M. Li, Humas: A heterogeneity- and upgrade-aware microservice auto-scaling framework in large-scale data centers, IEEE Trans. Comput. 74 (3) (2025) 968–982, http://dx.doi.org/10.1109/TC.2024.3506862.

[13] J. Shi, C. Liu, J. Liu, Hypergraph-based model for modeling multi-agent Q-learning dynamics in public goods games, IEEE Trans. Netw. Sci. Eng. 11 (6) (2024) 6169–6179, http://dx.doi.org/10.1109/TNSE.2024.3473941.

[14] W. Zhang, S. Zhang, Research on multi-objective optimisation for shared bicycle dispatching, Int. J. Veh. Inf. Commun. Syst. 9 (4) (2024) 372–392.

[15] G. Sun, Y. Zhang, H. Yu, X. Du, M. Guizani, Intersection fog-based distributed routing for V2V communication in urban vehicular ad hoc networks, IEEE Trans. Intell. Transp. Syst. 21 (6) (2020) 2409–2426, http://dx.doi.org/10.1109/TITS.2019.2918255.

[16] L. Song, G. Sun, H. Yu, D. Niyato, ESPD-LP: Edge service pre-deployment based on location prediction in MEC, IEEE Trans. Mob. Comput. 24 (6) (2025) 5551–5568, http://dx.doi.org/10.1109/TMC.2025.3533005.

[17] Y. Duan, Y. Zhao, J. Hu, An initialization-free distributed algorithm for dynamic economic dispatch problems in microgrid: Modeling, optimization and analysis, Sustain. Energy Grids Netw. 34 (2023) http://dx.doi.org/10.1016/j.segan.2023.101004.

[18] M. Dehghani, Z. Yazdanparast, From distributed machine to distributed deep learning: a comprehensive survey, J. Big Data 10 (1) (2023) http://dx.doi.org/10.1186/s40537-023-00829-x.

[19] Y. Chen, Q. Yang, S. He, Z. Shi, J. Chen, M. Guizani, FTPipeHD: A fault-tolerant pipeline-parallel distributed training approach for heterogeneous edge devices, IEEE Trans. Mob. Comput. 23 (4) (2024) 3200–3212, http://dx.doi.org/10.1109/TMC.2023.3272567.

[20] S. Ye, J. Du, L. Zeng, W. Ou, X. Chu, Y. Lu, X. Chen, Galaxy: A resource-efficient collaborative edge AI system for in-situ transformer inference, 2024, arXiv:2405.17245.

[21] S. Ye, L. Zeng, X. Chu, G. Xing, X. Chen, Asteroid: Resource-efficient hybrid pipeline parallelism for collaborative DNN training on heterogeneous edge devices, 2024, arXiv:2408.08015.

[22] M. Besta, T. Hoefler, Parallel and distributed graph neural networks: An in-depth concurrency analysis, IEEE Trans. Pattern Anal. Mach. Intell. 46 (5) (2024) 2584–2606, http://dx.doi.org/10.1109/TPAMI.2023.3303431.

[23] C. Zhang, M. Yu, L. Yu, P. Cong, Y. Yan, J. Bao, J. Jiang, X. Wang, X. Ye, T. Tang, L. Xiao, MSCH: Microbatch-based selective activation checkpointing with recomputation hidden for efficient training of LLM models, IEEE Access 12 (2024) 178460–178475, http://dx.doi.org/10.1109/ACCESS.2024.3456788.

[24] J. Rasley, S. Rajbhandari, O. Ruwase, Y. He, DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters, in: KDD '20: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2020, pp. 3505–3506, http://dx.doi.org/10.1145/3394486.3406703.

[25] S. Rajbhandari, J. Rasley, O. Ruwase, Y. He, ZeRO: Memory optimizations toward training trillion parameter models, in: Proceedings of SC20: The International Conference for High Performance Computing, Networking, Storage and Analysis, SC20, 2020, http://dx.doi.org/10.1109/SC41405.2020.00024.

[26] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, B. Catanzaro, Megatron-LM: Training multi-billion parameter language models using model parallelism, 2020, arXiv:1909.08053.

[27] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, M. Zaharia, Efficient large -scale language model training on GPU clusters using megatron-LM, in: SC21: International Conference for High Performance Computing, Networking, Storage and Analysis, 2021, http://dx.doi.org/10.1145/3458817.3476209.

[28] A. Sun, W. Zhao, X. Han, C. Yang, X. Zhang, Z. Liu, C. Shi, M. Sun, Seq1F1B: Efficient sequence-level pipeline parallelism for large language model training, 2024, arXiv:2406.03488.

[29] J. Hu, Y. Liu, H. Wang, J. Wang, AutoPipe: Automatic configuration of pipeline parallelism in shared gpu cluster, in: Proceedings of the 53rd International Conference on Parallel Processing, 2024, pp. 443–452, http://dx.doi.org/10.1145/3673038.3673047.

[30] B. Huang, X. Huang, X. Liu, C. Ding, Y. Yin, S. Deng, Adaptive partitioning and efficient scheduling for distributed DNN training in heterogeneous IoT environment, Comput. Commun. 215 (2024) 169–179, http://dx.doi.org/10.1016/j.comcom.2023.12.034.

[31] P. Qi, X. Wan, G. Huang, M. Lin, Zero bubble pipeline parallelism, 2023, arXiv:2401.10241.

[32] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, S. Chintala, PyTorch distributed: Experiences on accelerating data parallel training, Proc. Vldb Endow. 13 (12) (2020) 3005–3018, http://dx.doi.org/10.14778/3415478.3415530.

[33] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q.V. Le, Z. Chen, GPipe: Efficient training of giant neural networks using pipeline parallelism, 2018, CoRR, abs/1811.06965, arXiv:1811.06965.

[34] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N.R. Devanur, G.R. Ganger, P.B. Gibbons, M. Zaharia, PipeDream: Generalized pipeline parallelism for DNN training, in: Proceedings of the Twenty-Seventh Acm Symposium on Operating Systems Principles, SOSP '19, 2019, pp. 1–15, http://dx.doi.org/10.1145/3341301.3359646.

[35] J. Liu, Z. Wu, D. Feng, M. Zhang, X. Wu, X. Yao, D. Yu, Y. Ma, F. Zhao, D. Dou, HeterPS: Distributed deep learning with reinforcement learning based scheduling in heterogeneous environments, Futur. Gener. Comput. Syst. Int. J. Escience 148 (2023) 106–117, http://dx.doi.org/10.1016/j.future.2023.05.032.

[36] S. Fan, Y. Rong, C. Meng, Z. Cao, S. Wang, Z. Zheng, C. Wu, G. Long, J. Yang, L. Xia, L. Diao, X. Liu, W. Lin, DAPPLE: A pipelined data parallel approach for training large models, 2020, arXiv:2007.01045.

[37] B. Yang, J. Zhang, J. Li, C. Ré, C.R. Aberger, C.D. Sa, PipeMare: Asynchronous pipeline parallel DNN training, 2020, arXiv:1910.05124.

[38] Y. Gong, H. Yao, J. Wang, L. Jiang, F.R. Yu, Multi-agent driven resource allocation and interference management for deep edge networks, IEEE Trans. Veh. Technol. 71 (2) (2022) 2018–2030, http://dx.doi.org/10.1109/TVT.2021.3134467.

[39] T. Liu, S. Ni, X. Li, Y. Zhu, L. Kong, Y. Yang, Deep reinforcement learning based approach for online service placement and computation resource allocation in edge computing, IEEE Trans. Mob. Comput. 22 (7) (2023) 3870–3881, http://dx.doi.org/10.1109/TMC.2022.3148254.

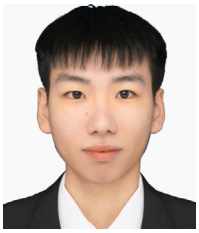
[40] W. Wang, Y. Zhang, Y. Jin, H. Tian, L. Chen, MDP: Model decomposition and parallelization of vision transformer for distributed edge inference, in: 2023 19th International Conference on Mobility, Sensing and Networking, MSN, 2023, pp. 570–578, http://dx.doi.org/10.1109/MSN60784.2023.00086.

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M.A. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533, http://dx.doi.org/10.1038/nature14236.

[42] S.B. Tandale, M. Stoffel, Recurrent and convolutional neural networks in structural dynamics: a modified attention steered encoder-decoder architecture versus LSTM versus GRU versus TCN topologies to predict the response of shock wave-loaded plates, Comput. Mech. 72 (4) (2023) 765–786, http://dx.doi.org/10.1007/s00466-023-02317-8.
[43] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, Commun. ACM 60 (6) (2017) 84–90, http://dx.doi.org/10.1145/3065386.
[44] S. Pan, Z. Chang, WD-1D-VGG19-FEA: An efficient wood defect elastic modulus predictive model, Sensors 24 (17) (2024) http://dx.doi.org/10.3390/s24175572.
[45] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2016, pp. 2818–2826, http://dx.doi.org/10.1109/CVPR.2016.308.
[46] S. Xie, R. Girshick, P. Dollár, Z. Tu, K. He, Aggregated residual transformations for deep neural networks, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2017, pp. 5987–5995, http://dx.doi.org/10.1109/CVPR.2017.634.
[47] I. Radosavovic, R.P. Kosaraju, R. Girshick, K. He, P. Dollar, Designing network design spaces, in: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, 2020, pp. 10425–10433, http://dx.doi.org/10.1109/CVPR42600.2020.01044.
[48] S.S. Hassan, S. Sarkka, Fourier-Hermite dynamic programming for optimal control, IEEE Trans. Autom. Control 68 (10) (2023) 6377–6384, http://dx.doi.org/10.1109/TAC.2023.3234236.
[49] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, J. Li, Cost-driven off-loading for DNN-based applications over cloud, edge, and end devices, IEEE Trans. Ind. Inform. 16 (8) (2020) 5456–5466, http://dx.doi.org/10.1109/TII.2019.2961237.
[50] L. Cui, J. Zhang, L. Yue, Y. Shi, H. Li, D. Yuan, A genetic algorithm based data replica placement strategy for scientific applications in clouds, IEEE Trans. Serv. Comput. 11 (4) (2018) 727–739, http://dx.doi.org/10.1109/TSC.2015.2481421.
[51] S. Liu, Research on manipulator control strategy based on PPO algorithm, in: 2023 Global Conference on Information Technologies and Communications, GCITC, 2023, pp. 1–4, http://dx.doi.org/10.1109/GCITC60406.2023.10426371.

Jiayi Li is currently pursuing the master's degree with the School of Electronics and Information, Shanghai Dianji University, China. Her research interests include distributed machine learning, computer network and scheduling algorithms, edge computing and deep learning.

Xiaogang Wang (Member, IEEE) received the Ph.D. degree in computer science and technology from Shanghai Jiao Tong University, China, in 2018. He is currently a Professor with the School of Electronics and Information, Shanghai Dianji University, China. He was also the Visiting Research Scholar with the CLOUDS Laboratory, University of Melbourne, Australia, from 2019 to 2020. He has published more than 50 papers in some journals and conferences such as TC, TSC, JSS, FGCS, CC, APIN, WI-IAT and APSCC. His research interests include distributed machine learning, computer network and scheduling algorithms, cloud and service computing, edge computing and deep learning. He is a member of the China Computer Federation.



Haokun Chen is currently pursuing the master's degree with the School of Electronics and Information, Shanghai Dianji University, China. His research interests include distributed machine learning, computer network and scheduling algorithms, edge computing and deep learning.

Zexin Wu is currently pursuing the master's degree with the School of Electronics and Information, Shanghai Dianji University, China. His research interests include distributed machine learning, computer network and scheduling algorithms, edge computing, object detection and deep learning.

Ziqi Zhu is currently pursuing the master's degree with the School of Electronics and Information, Shanghai Dianji University, China. Her research interests include distributed machine learning, computer network and scheduling algorithms, edge computing, object tracking and deep learning.

Jian Cao (Senior Member, IEEE) received the Ph.D. degree from the Nanjing University of Science and Technology, in 2000. He is currently a Professor in the Department of Computer Science and Engineering at Shanghai Jiao Tong University. His main research interests include service computing, cloud computing, network computing and scheduling algorithms, cooperative information systems and software engineering. He has published more than 200 papers in prestigious journals, such as TPDS, TMC, TSC, JSS and FGCS. He is a distinguished member of the China Computer Federation.

Rajkumar Buyya (Fellow, IEEE) received the Ph.D. degree in computer science and software engineering from Monash University, Melbourne, Australia, in 2002. He is a Redmond Barry distinguished professor and director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, the University of Melbourne, Australia. He served as a Future fellow of the Australian Research Council during 2012-2016. He has authored more than 625 publications and seven textbooks. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=155, g-index=334, 126,300+ citations). He served as the founding editor-in-chief of the IEEE Transactions on Cloud Computing. He is currently serving as co-editor-in-chief of Journal of Software: Practice and Experience.