Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2023.043822

ARTICLE

A Secure and Cost-Effective Training Framework Atop Serverless Computing for Object Detection in Blasting Sites

Tianming Zhang[1], Zebin Chen[1], Haonan Guo[2], Bojun Ren[1], Quanmin Xie[3,*], Mengke Tian[4,*] and Yong Wang[4]

[1]Department of Computer and Science, Shanghai Jiaotong University, Shanghai, 200240, China

[2]Aerospace System Engineering Shanghai, Shanghai, 200240, China

[3]State Key Laboratory of Precision Blasting, Jianghan University, Wuhan, 430056, China

[4]Beijing Microelectronics Technology Institute, Beijing, 100076, China

*Corresponding Authors: Quanmin Xie. Email: xqmblast@jhun.edu.cn; Mengke Tian. Email: mtianaa@connect.ust.hk

Received: 13 July 2023 Accepted: 15 November 2023 Published: 29 January 2024

ABSTRACT

The data analysis of blasting sites has always been the research goal of relevant researchers. The rise of mobile blasting robots has aroused many researchers' interest in machine learning methods for target detection in the field of blasting. Serverless Computing can provide a variety of computing services for people without hardware foundations and rich software development experience, which has aroused people's interest in how to use it in the field of machine learning. In this paper, we design a distributed machine learning training application based on the AWS Lambda platform. Based on data parallelism, the data aggregation and training synchronization in Function as a Service (FaaS) are effectively realized. It also encrypts the data set, effectively reducing the risk of data leakage. We rent a cloud server and a Lambda, and then we conduct experiments to evaluate our applications. Our results indicate the effectiveness, rapidity, and economy of distributed training on FaaS.

KEYWORDS

Serverless computing; object detection; blasting

1 Introduction

Nowadays, in the era of big data, various industries are collecting and analyzing massive amounts of data to further improve production or service efficiency. For example, using deep learning in the Industrial Internet of Things (IIoT) [1] or healthcare system [2]. In the field of blasting, planning blasting operations by analyzing data from blasting sites can significantly improve blasting quality and ensure human security. Today, mobile blasting robots [3–5] are usually used to place explosives instead of manual operation at blasting sites. Accurately identifying the work site within a complex blasting environment stands as the fundamental and indispensable initial phase for a mobile blasting robot. In the demolition blasting scene, there are thousands of blast holes in the buildings to be demolished. Without accurate identification, it will be impossible to accurately judge the explosive quantity and

placement position, leading to excessive consumption of blasting materials and a significant increase in potential security hazards.

Among many dangerous tasks, mobile blasting robots need to quickly analyze the data of the surrounding environment, such as detecting surrounding objects, so that they can sense the danger source in advance and alarm the explosion event. Deep learning methods are widely applied in these fields. Object detection [6] has always been an essential topic in image processing. Recently, object detection research has been developed rapidly with the development and broad application of deep learning. As machine learning is introduced into increasing fields to solve problems, it has become a consensus to solve the problem of object detection through machine learning. For example, using machine learning to solve the problem of image segmentation [7], image super-resolution [8], and 3D object detection [9]. However, training deep learning models is a typical computing-intensive task, which often requires the support of hardware devices with parallel capabilities. Therefore, many machine learning participants refer to cloud computing for computing power to train deep models.

Recently, serverless computing [10–12] has emerged as a new paradigm of computation infrastructure, which can support large-scale and elastically expanded data analysis and has been offered by major cloud service providers (e.g., AWS Lambda, Azure Functions, and Google Cloud Functions). By implementing the training tasks of deep learning models on the serverless platform, more deep learning practitioners without hardware facilities can use cloud power to complete the training for complex models. Many developers favor serverless computing as it lifts the burden of provisioning and managing cloud computation resources (e.g., with auto-scaling). Therefore, training machine learning (ML) models using serverless infrastructure has also attracted increasingly intensive attention from academia [13–15].

However, there is still a long way to go in training deep learning models using serverless computing, mainly due to the following three challenges. The first challenge is that cloud computing providers are not necessarily trusted. Still, the data for model training is private or valuable, so data owners are unwilling to expose the data to the cloud platform. When data utilized for model training is uploaded to a third-party platform, the security of such data becomes a paramount concern for its owners. The common solution is that users can adopt federated learning [16] or methods based on hardware encryption [17]. The second challenge is the gap between the demand for memory, network bandwidth, and other resources required for model training and the resources provided by the cloud computing platform. In the ML training process, the memory consumption increases with the model size and the activation size, and the latter is proportional to the batch size. Today's serverless platform, e.g., AWS Lambda, offers up to 10 GB of memory for a serverless function, often falling short for training with large batch sizes. The third challenge is that a large amount of intermediate data will be generated in the training process of the ML model, which requires high communication ability to transform the intermediate data. Serverless functions have minimal communication capability that does not meet the growing communication demand for training ML models. Moreover, serverless functions lack direct inter-function communication capability, making recent serverless-based training frameworks resort to two-hop communication via intermediary cloud storage such as Amazon S3.

Our work attempts to train machine learning models for object detection in blasting sites on a serverless platform to solve the above challenges. We solve the problem of data security and model volume through two main approaches, namely, data encryption and distributed training. In the process of data set partitioning, the image data is encrypted and then uploaded to the public storage service, which solves the data security concerns. Our critical insight is that we can split data into partitions and disperse them to multiple Lambda functions for training. Distributed training can effectively reduces

the platform's memory requirements and thus enable large-volume object detection model training. Moreover, data parallelism can also effectively reduce the communication burden in training. On the basis of distributed training, we implement a parameter server architecture for distributed training using serverless computing to address the communication challenge, ensuring a cloud-native machine learning training solution.

To measure the overall performance of our application, we configure relevant environments on AWS Lambda and AWS EC2. Then, we train different models on the platforms and record the indicators, such as the loss in the training process, the Intersection over Union (IoU) for the test set, and the estimated cost. We also verify the efficiency and security of our encryption algorithm when compared with other commonly used encryption algorithms. In short, we make the following main contributions:

- We design a distributed training application for object detection tasks in blasting sites, which can assist the blasting operation with cloud computing resources.
- We implement the training application in a serverless computing manner based on the AWS Lambda platform, which can realize data transmission and synchronized distributed training on Function as a Service (FaaS) and provides enough memory (up to 10 GB per Lambda function) for classical Deep Neural Networks (DNN) model training.
- We propose the idea of applying AES encryption for slice data when the data set is split and uploaded to the public bucket, which effectively reduces the risk of user-sensitive data being exposed when it is stored on a third-party platform.
- We extensively evaluate the overall performance of our application. The experimental results show that our application can train ML models more quickly, economically, and safely while approaching the performance of full data set training.

The paper will be presented in the following order: In Section 2, we will introduce the overall architecture of our designed application, the implementation of distributed training, and the application of encryption algorithms. In Section 3, we will introduce the evaluation of our designed method. In Section 4, we will summarize and analyze relevant studies. In Section 5, we will give a conclusion of this paper.

2 Design

We implement a prototype FaaS-based Machine-Learning application built on Amazon Lambda for object detection. In this section, we will first introduce the overall design of our application and then introduce two important components: an effective distributed method to train models on the serverless platform and an encryption algorithm to guarantee the security of the training data set.

2.1 Overview

Challenges. When developing the distributed Machine-Learning training application for blasting data models on a serverless platform, we mainly consider two challenges: (1) distributed training method and relevant techniques, and (2) data security.

The current FaaS infrastructure does not allow direct communication between stateless functions. It only allows functions to read/write intermediate state information generated during the iterative training by using specific storage channels. Therefore, we should design our distributed training method, consider the synchronization between functions, and realize intermediate data communication based on this situation.

The original blasting data is usually confidential information of enterprise or military background units. When conducting data training on a third-party serverless platform, encryption of the original data must also be considered.

Framework. Fig. 1 shows the framework of the application. Before executing the application, users need to open related services on AWS in advance and deploy the application on the AWS lambda platform accordingly. The user first submits the specified configuration (data set location, common bucket location, training model, model parameters, etc.) to the trigger function. Then, the trigger function will split the data according to the number of workers and start the workers' function to conduct distributed training and aggregation of the net. Each running instance is a function running on AWS Lambda. The training data is partitioned and stored in S3, a distributed storage service in AWS. Each worker has a copy of the partition of the training data set in its local temporary storage. After training its network weights locally, a worker will upload it to the public bucket. Specific workers pool and aggregate the network weight according to the data set size.

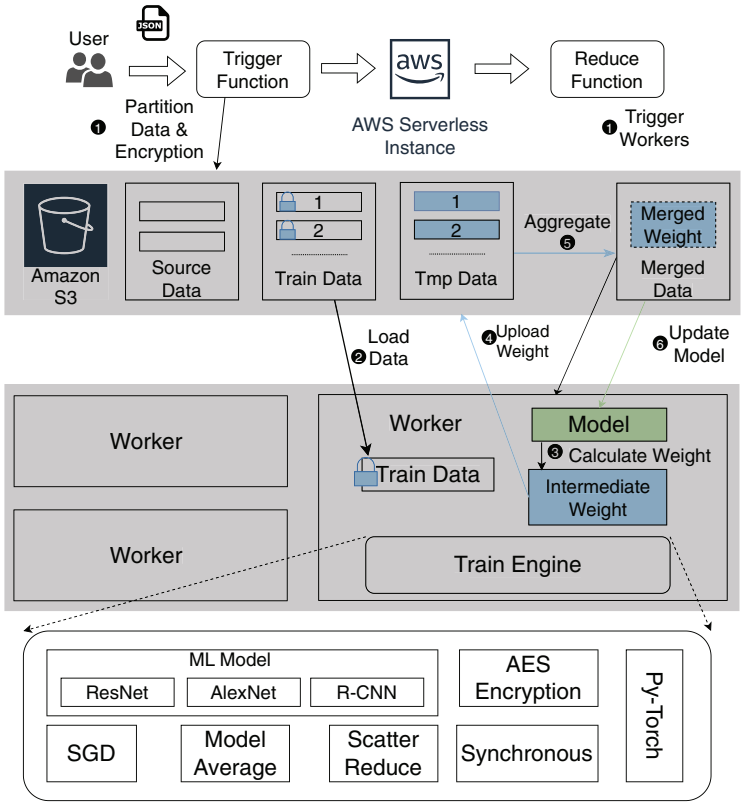


Figure 1: The framework of the application. The training process corresponds to the job execution part. The Train Engine is the corresponding functional module of the distributed training we wrote for the workers in the application

Job Execution. A training job in our application has the steps below:

1. Partition data and trigger workers. The trigger function reads the configuration parameters entered by the user, splits the original data set according to the number of workers, uploads the partition to the specified public bucket, and calls the functions of workers deployed on AWS Lambda by Function Invoke.

2. Load data. Each worker loads the corresponding partition of training data from S3.
3. Calculate weight. Each worker uses PyTorch to create the specified ML model and the allocated split training data and local model parameters to calculate the model parameters after this iteration.
4. Upload weights. Each worker uploads its model weight to the designated S3.
5. Aggregate weights. Based on the idea of average pooling, the model weights of all workers considered the intermediate state are aggregated by obtaining the average value of each worker's weights to generate the global state of model weights. Note that the aggregate progress is conducted in a synchronous manner because each worker has similar performance, and the data set is partitioned equally.
6. Update model. Each worker reads the merge status of model weights from S3 and updates the local model with this status. If the number of training iterations or loss requirements is not met, return to step 3 and run the next iteration.

2.2 Distributed Training

It is a popular research direction to deploy Distributed Deep Neural Networks (DDNN) to stateless serverless platforms for efficient and easy training. In the general process of deep neural network training, a large amount of intermediate data will be generated. However, an efficient distributed training method must be devised to operate within constrained function running time, involve temporary storage of data exclusively during function execution, and preclude communication between functions on a serverless platform. It is necessary to solve how to implement the optimization algorithm and complete the aggregation of data when realizing the distributed training of the deep learning model on a serverless platform.

Data Parallelism. For given data set D and loss function J, each iteration of the training process can be represented as:

$$\theta^{(t+1)} = \theta^{(t)} - \varepsilon J(\theta^{(t)}, D^{(t)}) \tag{1}$$

In Eq. (1), t represents the t-th training iteration, θ is the model weights, $J(\cdot)$ is the loss function, ϵ is the learning rate. Distributed training means that computing and storage requirements are distributed to multiple training devices, and data parallelism is a parallel strategy to solve this problem. The main principle of data parallelism follows the principle of Single Program Multiple Data, that is, the training task is divided into multiple processes (devices). Data set D is partitioned equally into n parts, and the i-th worker holds the partial data set $D_i (i \in [1, n])$, n is the number of workers. Each process maintains the same model parameters and the same computing task but processes different batch data. In this way, the data and calculation under the same global batch are split into different serverless instances, thus reducing the pressure of calculation and storage on a single instance. The t-th iteration of the i-th worker can be represented as:

$$\theta_i^{(t+1)} = \theta^{(t)} - \varepsilon J(\theta^{(t)}, D_i^{(t)}) \tag{2}$$

In the scenario of object detection, the loss function is set to IoU, which is computed as follows:

$$IoU = \frac{P \cap G}{P \cup G} \tag{3}$$

In Eq. (3), P is the predicted area, G is the real area. There are many ways to realize data parallelism. The data parallelism of this framework is based on Distributed Synchronous SGD, which

is the implementation method of data parallelism in the current mainstream deep learning training framework.

Distributed SGD. Due to the success of deep neural networks, stochastic gradient descent (SGD) may be the most popular optimization algorithm in the world today. When implementing SGD in a distributed way, we consider the following variant: gradient average (GA). We divide the training data evenly and let one worker take charge of one partition. Each worker runs a small batch of SGDs independently and in parallel while sharing and updating the global ML model in case of synchronization barriers set by users (for example, after one or several iterations). The update mode of the global model is the difference between the variant we designed and other distributed SGD modes. GA updates the global model in each iteration by collecting and aggregating updated gradients from Workers. GA updates the overall model weights as follows:

$$\theta^{(t+1)} = \frac{1}{n}\sum_{i=1}^{n}\theta_i^{(t+1)} \tag{4}$$

For most multi-layer ML models, this method of updating the global model is applicable.

A FaaS-Based Data Aggregation. We design a data aggregation communication scheme using the persistent storage service (S3) provided by the AWS platform as the storage location of intermediate data. The entire communication process contains the following steps:

1. Each worker is an aggregator, so the model weight of the iteration will be divided into the number of workers.
2. Workers upload the weights of the areas in their charge to S3 for temporary files.
3. Workers download the weights uploaded by other workers in their charge from S3.
4. All workers read the merged model weight data from S3.
5. All workers will refresh their local models with the information read from the merged file and then enter the next iteration or exit the training after reaching the set requirements.

2.3 Security Guarantee Algorithm

AES encryption algorithm [18] is the most popular computer encryption algorithm. Because it uses the same key during encryption and decryption, it belongs to the single-key encryption algorithm. AES algorithm has high security and can resist square attacks, side-channel attacks, penetration attacks, meet-in-the-middle attacks, statistical analysis attacks, energy analysis attacks, etc.

AES Algorithm Encryption and Decryption. The encryption and decryption process of the AES algorithm is shown in Fig. 2. The AES algorithm operates in bytes, and the 16-byte data is represented as a $4 * 4$ matrix according to the specified byte arrangement. The whole encryption process can be understood as first performing AddRoundKey on the input $4 * 4$ plain text matrix P to get the state matrix S, then performing ten iterations of functions, and finally getting the $4 * 4$ ciphertext matrix C for output. Byte Subs, invShiftRow, Column obfuscation, and AddRoundKey are performed in turn in the first and ninth iterations. There is no Column obfuscation in the tenth round compared to the first nine iterations. The encryption and decryption process reverses the order of the round keys.

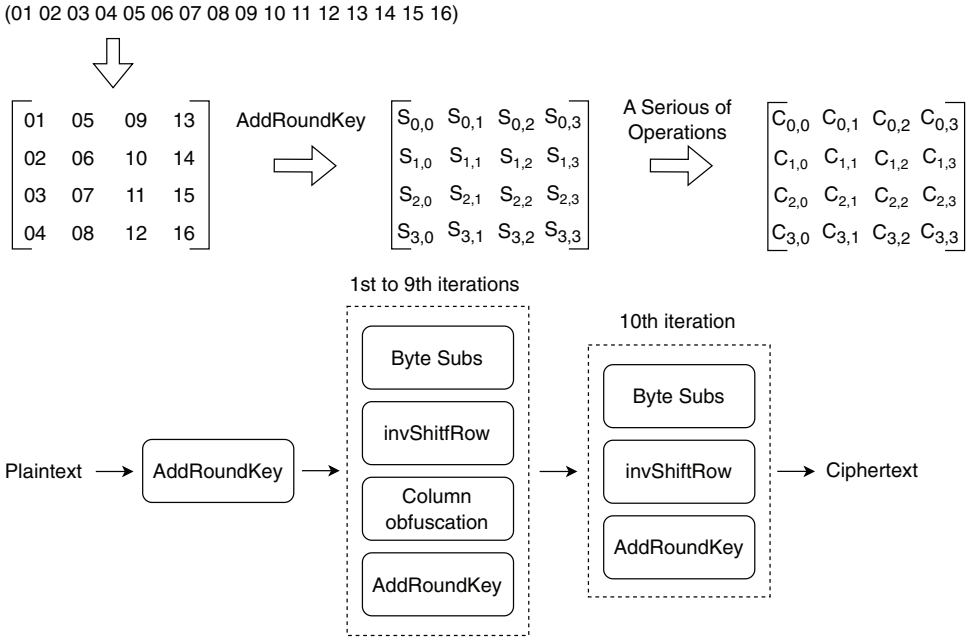


Figure 2: Details of encryption and decryption process of AES algorithm. The row vector represents 16-byte data. Three $4 * 4$ matrices represent the plaintext matrix, state matrix, and ciphertext matrix in turn. The flow chart at the bottom is the encryption process of Plaintext

Data Encryption. The application partitions the original data set and uploads them to the public bucket, and the split data set needs to be encrypted. The application program partitions the data with a block length of 128 bits and then encrypts the data based on The Cipher Block Chaining Mode (CBC). Every 128 bits of data is represented as a stream in Line 3. Unlike the traditional CBC mode, which uses the previous ciphertext block as the Initialization Vector (IV), we randomly generate an IV for each data block to minimize the retention of original features. Due to the openness of IV, we splice the generated IV with the ciphertext block and save it. After the AES encryption, the cipher data is encoded into ASCII codes with BASE64 for data transmission. Algorithm 1 shows how to encrypt an original image completely.

Algorithm 1: Image Encryption

Input: the original image *IMAGE*
Output: the cipher data *RESULT*
1: *RESULT* ← [ ]
2: **while** not end of *IMAGE* **do**
3: stream ← Read 128 bits of data
4: Randomly ← generate IV
5: cipher_data = AES_Encrypt(stream, IV)
6: tmp_data = BASE64_Encode(cipher_data + IV)
7: *RESULT* ←tmp_data
8: **end while**
9: **return** *RESULT*

Data Decryption. We use the AES encryption algorithm in CBC mode, hence the decryption process of the data in the application is basically the reverse process of encryption. After the worker downloads the responsible data block from the S3 bucket, the data is first Base64 decoded. Then, it decrypts the data every 144 bytes according to the key provided by the user. The first 128 bits are the original data, and the last 16 bits are IV. Finally, the decrypted data is spliced.

3 Evaluation

In this section, we mainly design two scenarios of experiments to verify the overall performance and data security of our designed application. The first one is training different models with different training configurations. The second one is evaluating the efficiency and security of different encryption algorithms.

3.1 Experiment Setup

Testbed. Our assessment uses the popular Function as a Service (FaaS) platform and Platform as a Service (PaaS) platform, AWS Lambda, and AWS EC2 ECS. AWS EC2 ECS serves a local training baseline that takes the full data set, while AWS Lambda is used to test the distributed training performance. In our evaluation, AWS Lambda provides a maximum of 3008 MBytes memory allocation for each serverless function. Its corresponding cloud storage service, S3, grants unlimited bandwidth to concurrent access. According to the official guidance of Amazon, every 1 GB of running memory allocated for a function on Lambda is equivalent to allocating 0.6v CPU. AWS EC2 ECS provides many different configurations to meet various needs. In our evaluation, the purchased model is the computing enhanced t3.xLarge, which is configured with 4v CPU and 16 GiB memory.

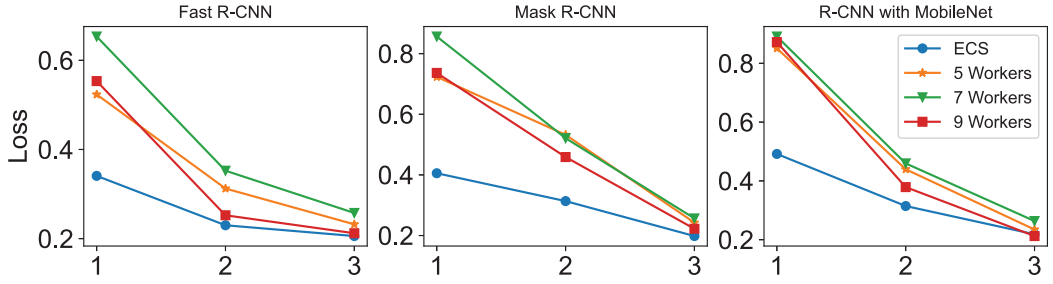
Dataset & Models. The data set used for our evaluation is the classic data set PennFudanPed [19] in the field of object detection. PennFudanPed includes the original image, mask, and annotation. The position and information of the person in the image are indicated in the annotation file.

We use the following ML models in our evaluation. Fast R-CNN is proposed based on R-CNN. It eliminates the SVM classifier and bbox linear regression, places them in the integrated network, and uses the ROI-pooling layer to convert RPs of different sizes into the same size to complete classification and regression at one time. Mask R-CNN is composed of Fast R-CNN and semantic segmentation algorithm FCN. The former completes the object detection task, and the latter can accurately complete the semantic segmentation task. R-CNN with the model skeleton of MobileNet v2. MobileNet v2 uses Inverted Research and Linear Bottlenecks and performs well in small-sample object detection models. The batch size of the overall performance experiment is set to one. In addition, we commit another experiment of R-CNN with MobileNet with different batch sizes to examine the performance. We set the optimal learning rate for each ML model at 0.005 and the stop condition of training as the number of iterations to compare the network performance under the same number of iterations. The epoch of the training process is set to 3, which is enough for the comparison of different configurations.

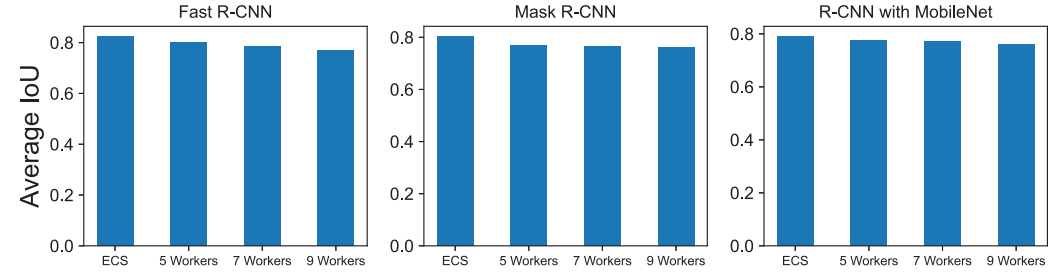
3.2 Overall Performance

We compare the overall performance by running distributed ML training applications on Lambda and training the ML model on ECS. Since the trained ML model involves Fast R-CNN, which consumes more memory, the running memory on Lambda is uniformly set to a maximum of 3008 MB to prevent memory shortage. The number of workers is set to 5, 7, and 9, which is equivalent to the computing power of a 3v CPU, 4.2v CPU, and 5.4v CPU.

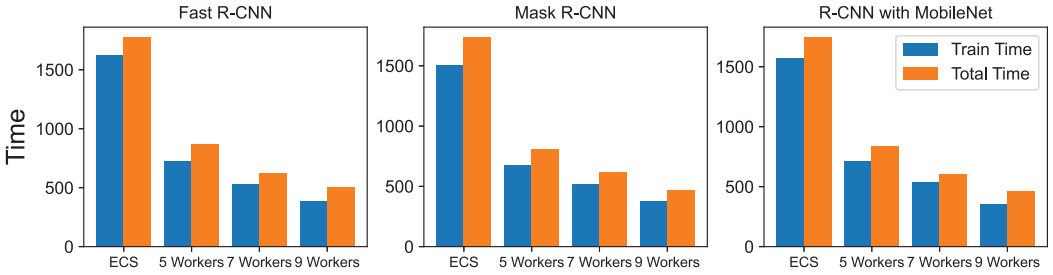
Loss. In Fig. 3a, we draw the training loss line graph of each epoch of the three ML models. The horizontal axis unit of the line graph is the epoch, and the vertical axis unit is the average loss. On ECS, due to a large amount of training data, the loss of each ML model tends to be stable after an epoch. On Lambda, the data set is partitioned, resulting in a relatively small data volume. The loss is relatively large in the early training period but decreases quickly. It can be seen from the image that the more workers, the greater the loss will be at the beginning of the training. However, as the iteration progresses, it will soon decrease. The cutting of the data set does not affect the training loss of the model. After multiple iterations, it can converge to the level of full data set training.



(a) Training loss of different workers



(b) Average IoU of different workers



(c) Training time of different workers

Figure 3: Comparison of overall performance of different configurations. The loss in Fig. 3a refers to the average loss of each batch. The specific criterion is the gap between the predicted value of the model and the actual value. The intersection-over-Union (IoU) in Fig. 3b is a concept used in object detection. They are the overlap rate of the generated candidate box and the original marker box, that is, the ratio of their intersection and union, and are commonly used standards to measure the confidence of the detection results. The evaluation index used in the experiment is the Average Precision when IoU is 0.50 to 0.95

Accuracy. In Fig. 3b, we draw the global IoU of the three models with confidence levels of 50% to 95%. Regardless of the model, the best IoU indicators are obtained through model training on ECS. This is not counterintuitive because ECS has a complete training data set. Each Lambda only has a partial training set, and the weight of the overall training model is the integration of all local models. In addition, the figure shows that the more workers, the less overall accuracy acquired. This is because, with the decrease of the data set scale, each worker trains the model with less data, which causes the degeneration of the overall accuracy performance. However, the test results on Lambda are not much different from those on ECS, which is acceptable.

Time. As shown in Fig. 3c, the performance of distributed ML training applications is much better than that of ECS in both training time and total time. Theoretically, the shortened time is linearly related to the number of workers. Considering the aggregation time, synchronization time, and other necessary start-up times, using distributed training does not reduce the running time linearly. However, it is not difficult to see that distributed training still significantly reduces training time and total run time. With the increment in the number of workers, more overall computing resources can be utilized to train the model, leading to a minor training time and overall time.

Cost. According to the official pricing manual, our estimated cost is shown in Table 1. Cost on Lambda mainly refers to the pricing of temporary storage and runtime pricing. When the running memory is 3072, the running time cost per 1 ms is 5.0×10^{-8} USD. The cost of temporary storage is 3.7×10^{-8} USD per GB per second. At present, Amazon does not charge for the storage space and communication of S3 bucket, so it does not need to be included in the bill. For the t3.xlarge EC2 instance we rented, its price is 0.2176 USD per hour. The traffic cost of EC2 instances is USD 0.01 per GB for the first 10 TB, and the cost of universal SSD storage is USD 0.08 per GB per month. The size of the data set is 50 MB, and the storage volume we set is 10 GB. Since we can focus on the implementation of a specific function on Lambda without paying for other hardware facilities, our distributed application achieves better performance in terms of cost performance.

Table 1: Cost of different scenarios

Object	Fast R-CNN		Mask R-CNN		R-CNN with MobileNet	
	Duration (s)	Cost (USD)	Duration (s)	Cost (USD)	Duration (s)	Cost (USD)
5 workers	289.32 ∗ 5	0.1258	269.66 ∗ 5	0.1173	279.75	0.1217
7 workers	207.54 ∗ 7	0.1532	205.98 ∗ 7	0.1520	201.33	0.1486
9 workers	169.44 ∗ 9	0.1789	155.04 ∗ 9	0.1637	153.48	0.1621
ECS	1773.28	0.1770	1734.19	0.1731	1743.69	0.1740

Batch Size. As shown in Fig. 4, with the increment of batch size, the training process converges slower, which is consistent with the theory. Besides, Fig. 4 shows that the training process with a larger data set has a better loss performance, which corresponds with the analysis in the overall performance experiment. Note that in the experiment with batch size set to 4, the demanding memory of ECS is beyond the memory constraint of AWS Lambda, so the result is set to 0 in the figure. This is intuitive because with the number of workers decreases, each worker needs more memory to storage the growing partial data set.

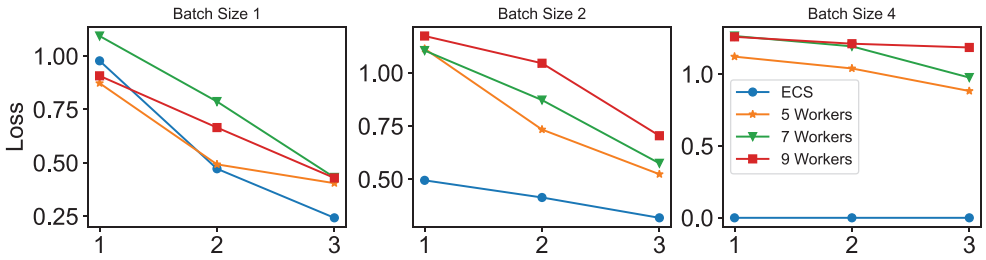


Figure 4: Loss of training R-CNN with MobileNet with different workers under different batch size configurations

3.3 Data Security

Efficiency. In order to verify the advantages of our AES encryption algorithm in terms of encryption and decryption efficiency, we test the speed of the commonly used RSA-1024 algorithm, a widely used security encryption algorithm based on the principle of large prime product decomposition, DES-64 algorithm, which is a classical symmetric encryption method and AES-128 algorithm to encrypt and decrypt the same data, and the result is shown in Table 2. AES algorithm and DES algorithm are both symmetric encryption algorithms. The AES encryption algorithm exhibits a characteristic where the size of the ciphertext is twice that of the plaintext, consequently leading to a commensurate increase in the decryption time relative to the encryption process. In contrast, the DES algorithm maintains equivalent sizes for both ciphertext and plaintext, resulting in equivalent encryption and decryption speeds. It is noteworthy that RSA constitutes an asymmetric encryption algorithm. The speed of RSA algorithm decryption is relatively slow, as the decryption process involves exponential operation on large ciphertext, which requires more computational resources and time. In contrast, encryption speed is faster because encryption only requires one modular exponentiation operation. The speed performance of the RSA algorithm and DES algorithm for encrypting data is obviously inferior to that of the AES algorithm. Besides, with the increment of the file length, the decryption time grows exponentially, making it even slower. As for AES algorithm, it demonstrates notably swift encryption and decryption speeds comparing with AES and RSA, irrespective of whether applied to small or large file size.

Table 2: Comparison of algorithm encryption/decryption speed

Algorithm	Minimal data Consumption (ms)	30 kb file time Consuming (ms)	100 kb file time Consuming (ms)	200 kb file time Consuming (ms)
AES-128	**0.31/0.60**	**2.81/5.63**	**9.36/18.86**	**15.75/31.44**
RSA-1024	21/153	1515/13287	4291/74406	9218/369826
DES-64	1.90/1.94	12.68/12.69	33.14/33.10	57.72/57.65

Security. In order to measure and compare the security of encryption algorithms, we use the brute force attack method to test the resistance of the RSA algorithm and the AES algorithm. It can be seen from Table 3 that the performance of the AES algorithm is slightly worse than that of the RSA algorithm, which is known for its security. However, considering its encryption and decryption efficiency, these security losses are acceptable. The DES algorithm performs worst, making it unsuitable for our method's data encryption process.

Table 3: Comparison of algorithm attacking speed

Algorithm	AES-128	RSA-1024	DES-64
Attack time consuming (s)	4.895	6.354	0.034

4 Related Work

Deep Learning for Blasting Applications. The extensive application of deep learning has extensively promoted productivity development in all walks of life, including some dangerous blasting industries. Literature [20] proposed to utilize deep learning models to remain time to close tap holes for blast furnaces. They adopt skip-dense layers to outperform the LSTM-based baselines in accuracy and performance. A deep learning model Mask R-CNN [21] trained through images captured from real blasting sites in Nui Phao open-pit mine is developed to evaluate the blasting results, expanding the possibility of the automated measurement of blast fragmentation. Literature [22] combined a hybrid deep learning-based computer vision method with a VMD algorithm for security detection of blasting furnace bearings, which achieves remarkable calculation speed and accuracy of bearing fault diagnosis. Literature [23] constructed a multi-hidden-layer neural network model and an LSTM neural network model based on PyTorch and Keras framework to predict the failure mode of the RC (reinforced concrete) columns under blast loading. Literature [24] adopted a convolutional neural network (CNN) for training models that can distinguish tectonic earthquakes and quarry blasts. They also apply different strategies due to different data sizes and yield high accuracy in seismic event discrimination with raw waveforms. Literature [25] decreased the blast-induced vibration in a tunnel excavation by deciding the initial setting of the MSP (multi-setting smart investigation of the ground and pre-large hole boring) machine with a deep learning-based prediction model. To avoid overfitting while training the model, they have applied several techniques like dropout, early stopping, and pretraining. Literature [26] performed five artificial intelligent algorithms with deep learning models to predict the flock phenomenon that is frequently appeared during the explosion in mining or construction projects. They also assess the performance of the five models, and Harris Hawks optimization-based MLP (HHO-MLP) achieves the best score. Our work aims to address the object detection tasks in blasting sites to improve the blasting performance. We design and implement a distributed machine learning training application for object detection models on a commercial serverless computing platform to achieve this goal.

Distributed Machine Learning. Due to the complexity of the deep learning model and the explosive growth of training data, the requirements of improving training speed and reducing model convergence time can no longer be met on a single GPU on a machine. Therefore, distributed machine learning is proposed to accelerate the convergence of the model by using the idea of parallel computing. Distributed machine learning can be divided into data parallelism, model parallelism in data partition, and model partition. A parameter server [27] is a typical representation of data parallelism. The training is split into partitions and dispersed across workers, and each worker keeps a complete copy of the model for local updates. The centric server is responsible for weight aggregation by collecting the local weights from workers. However, due to the memory limitation of a single GPU, some giant models cannot be trained on a single GPU, so it is feasible to divide the model into different machines or GPUs for training, which is why model parallelism [28,29] plays an important role. Federated learning has gradually attracted people's attention and has become a new paradigm of distributed machine learning for its privacy guarantee [30]. What's more, compared with the implementation in

the uniformly managed device cluster, building stronger computing power on the edge devices with low configurations and more numbers to realize the parallel training of the model has been favored by some researchers [31,32]. OSTTD [33] trains the task unloading model with the help of serverless service. OSTTD is a novel task-offloading method for multi-tier computing networks. This method addresses the challenges of offloading splittable tasks with topological dependence in complex and dynamic systems. Our work implements a distributed DNN model training application on a serverless computing platform, AWS Lambda, for the convenience of deploying and applying DNN applications on dangerous industrial sites.

Serverless Computing. In the past decades, thanks to the development of virtualization technology, cloud computing has gradually moved from the heavyweight Infrastructure-as-a-Service (IaaS) to the lightweight Function-as-a-Service (FaaS). At present, serverless computing has gradually attracted the attention of researchers and engineers. Its main advantages include simplified development, operation, maintenance processes, automatic scalability, and a "pay-as-you-use" billing mode. Serverless computing has been widely used in computing-intensive or traffic mutation tasks, such as machine learning training [14,34], video processing applications [35–38], etc. Sprocket [35] is a scalable serverless framework deployed on AWS Lambda for multistage video processing, including video encoding, decoding, and classification. Llama [36] extends Sprocket by enabling automatic parameter determination to optimize resource configuration and heterogeneous hardware support (GPU) to accelerate video processing. For machine learning, serverless computing can satisfy the requirements of machine learning applications that parallel computing with high computing performance. FaaShark [39] is an end-to-end network traffic analysis system based on a serverless computing platform that provides valuable insights into the use of serverless computing platforms for network traffic analysis. Siren [34] is a distributed machine learning training framework deployed on AWS Lambda and mainly utilizes reinforcement learning to guide the resource configuration, including the worker number and memory quota. Besides model training, serverless computing can be applied in model serving because of its automatic scalability to handle workload variation. SSC [40] is a pre-warming and automatic resource allocation framework designed explicitly for serverless workflows, which can reduce the cold start rate significantly. Batch [37] provides an adaptive batch size specification when serving inference requests to improve system throughput and resource utilization. Gillis [38] adopts model partition to serve large models because of the memory limitation of serverless platforms, like AWS Lambda and Aliyun Function. It provides two partition schemes to optimize execution time or cost, respectively. In addition, introducing cloud computing into IoT devices has become a new direction of current research. Our work applies AES encryption on data uploaded to a public bucket when utilizing the serverless computing method, which improves the security of data significantly.

5 Conclusion

We design a distributed ML model training application for object detection in the blasting field on FaaS, mainly including its communication mode, optimization algorithm implementation, data storage, and synchronization mode. We also encrypt the data in the training process, effectively reducing the exposure risk of user data stored on third-party platforms. We then implement our distributed ML training application on Amazon Lambda, following which we conduct a series of experiments to evaluate the overall performance. Our results indicate that our application can complete ML model training faster and more economically, with a performance close to full data set training. Our design of serverless computing DNN training framework is not only suitable for object detection in blasting sites but also can be extended to other DNN tasks in different dangerous circumstances, for example, image classification and image segmentation. Besides, with the improvement of services

from serverless computing vendors, the framework can support larger models with more parameters like YOLOv3 and SSD in the future.

Acknowledgement: The authors wish to express their appreciation to the reviewers for their helpful suggestions which greatly improved the presentation of this paper.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: T. Zhang, Z. Chen, M. Tian; software: T. Zhang, B. Ren; data collection: T. Zhang, Q. Xie; analysis and interpretation of results: T. Zhang, H. Guo, B. Ren; draft manuscript preparation: T. Zhang, Z. Chen; manuscript review: H. Guo, Q. Xie, M. Tian, Y. Wang. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Publicly available data sets were analyzed in this study. This data can be found here: https://www.cis.upenn.edu/~shi/ped_html/.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Gao, H., Qin, X., Barroso, R. J. D., Hussain, W., Xu, Y. et al. (2020). Collaborative learning-based industrial iot api recommendation for software-defined devices: The implicit knowledge discovery perspective. *IEEE Transactions on Emerging Topics in Computational Intelligence, 6(1),* 66–76.
2. Gao, H., Xu, K., Cao, M., Xiao, J., Xu, Q. et al. (2021). The deep features and attention mechanism-based method to dish healthcare under social IoT systems: An empirical study with a hand-deep local—global net. *IEEE Transactions on Computational Social Systems, 9(1),* 336–347.
3. Muthugala, M. V. J., Le, A. V., Cruz, E. S., Rajesh Elara, M., Veerajagadheswar, P. et al. (2020). A self-organizing fuzzy logic classifier for benchmarking robot-aided blasting of ship hulls. *Sensors, 20(11),* 3215.
4. Carmichael, M. G., Aldini, S., Khonasty, R., Tran, A., Reeks, C. et al. (2019). The ANBOT: An intelligent robotic co-worker for industrial abrasive blasting. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8026–8033. Macau, China, IEEE.
5. Le, A. V., Kyaw, P. T., Veerajagadheswar, P., Muthugala, M. V. J., Elara, M. R. et al. (2021). Reinforcement learning-based optimal complete water-blasting for autonomous ship hull corrosion cleaning system. *Ocean Engineering, 220,* 108477.
6. Padilla, R., Netto, S. L., Da Silva, E. A. (2020). A survey on performance metrics for object-detection algorithms. *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 237–242. Niteroi, Brazil, IEEE.
7. Feng, R., Liu, X., Chen, J., Chen, D. Z., Gao, H. et al. (2020). A deep learning approach for colonoscopy pathology wsi analysis: Accurate segmentation and classification. *IEEE Journal of Biomedical and Health Informatics, 25(10),* 3700–3708.
8. Chen, J., Ying, H., Liu, X., Gu, J., Feng, R. et al. (2020). A transfer learning based super-resolution microscopy for biopsy slice images: The joint methods perspective. *IEEE/ACM Transactions on Computational Biology and Bioinformatics, 18(1),* 103–113.
9. Gao, H., Fang, D., Xiao, J., Hussain, W., Kim, J. Y. (2023). Camrl: A joint method of channel attention and multidimensional regression loss for 3D object detection in automated vehicles. *IEEE Transactions on Intelligent Transportation Systems, 24(8),* 8831–8845.

10. Li, Z., Guo, L., Cheng, J., Chen, Q., He, B. et al. (2022). The serverless computing survey: A technical primer for design architecture. *ACM Computing Surveys (CSUR), 54(10s),* 1–34.
11. Shafiei, H., Khonsari, A., Mousavi, P. (2022). Serverless computing: A survey of opportunities, challenges, and applications. *ACM Computing Surveys, 54(11s),* 1–32.
12. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C. C., Khandelwal, A. et al. (2019). Cloud programming simplified: A berkeley view on serverless computing. arXiv preprint arXiv:1902.
13. Xu, F., Qin, Y., Chen, L., Zhou, Z., Liu, F. (2021). λ*DNN*: Achieving predictable distributed DNN training with serverless architectures. *IEEE Transactions on Computers, 71(2),* 450–463.
14. Wang, H., Niu, D., Li, B. (2019). Distributed machine learning with a serverless architecture. *IEEE INFOCOM 2019—IEEE Conference on Computer Communications*, pp. 1288–1296. Paris, France.
15. Jiang, J., Gan, S., Liu, Y., Wang, F., Alonso, G. et al. (2021). Towards demystifying serverless machine learning training. *Proceedings of the 2021 International Conference on Management of Data*, pp. 857–871. New York, NY, USA, Association for Computing Machinery.
16. Shi, H., Ma, R., Li, D., Guan, H. (2023). Hierarchical adaptive collaborative learning: A distributed learning framework for customized cloud services in 6G mobile systems. *IEEE Network, 37(2),* 44–53.
17. Cai, Z., Ren, B., Ma, R., Guan, H., Tian, M. et al. (2023). Guardian: A hardware-assisted distributed framework to enhance deep learning security. *IEEE Transactions on Computational Social Systems, 10(6),* 3012–3020. https://doi.org/10.1109/TCSS.2023.3262289
18. Mahajan, P., Sachdeva, A. (2013). A study of encryption algorithms AES, DES and RSA for security. *Global Journal of Computer Science and Technology, 13(15),* 15–22.
19. Wang, L., Shi, J., Song, G., Shen, I. F. (2007). Object detection combining recognition and segmentation. In: Yagi, Y., Kang, S. B., Kweon, I. S., Zha, H. (Eds.), *Computer Vision—ACCV 2007*, pp. 189–199. Berlin, Heidelberg: Springer Berlin Heidelberg.
20. Kim, K., Seo, B., Rhee, S. H., Lee, S., Woo, S. S. (2019). Deep learning for blast furnaces: Skip-dense layers deep learning model to predict the remaining time to close tap-holes for blast furnaces. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 2733–2741. New York, NY, USA, Association for Computing Machinery. https://doi.org/10.1145/3357384.3357803
21. Vu, T., Bao, T., Hoang, Q. V., Drebenstetd, C., Hoa, P. V. et al. (2021). Measuring blast fragmentation at nui phao open-pit mine, vietnam using the mask r-cnn deep learning model. *Mining Technology, 130(4),* 232–243.
22. Yang, A. M., Zhi, J. M., Yang, K., Wang, J. H., Xue, T. (2021). Computer vision technology based on sensor data and hybrid deep learning for security detection of blast furnace bearing. *IEEE Sensors Journal, 21(22),* 24982–24992.
23. Zhou, X. Q., Huang, B. G., Wang, X. Y., Xia, Y. (2022). Deep learning-based rapid damage assessment of rc columns under blast loading. *Engineering Structures, 271,* 114949.
24. Zhu, J., Fang, L., Miao, F., Fan, L., Zhang, J. et al. (2022). Deep learning and transfer learning of earthquake and quarry-blast discrimination: Applications to Southern California and Eastern Kentucky. https://doi.org/10.1002/essoar.10511205.1
25. Kim, M. S., Lee, J. K., Choi, Y. H., Kim, S. H., Jeong, K. W. et al. (2020). A study on the optimal setting of large uncharged hole boring machine for reducing blast-induced vibration using deep learning. *Explosives and Blasting, 38(4),* 16–25.
26. Murlidhar, B. R., Nguyen, H., Rostami, J., Bui, X., Armaghani, D. J. et al. (2021). Prediction of flyrock distance induced by mine blasting using a novel harris hawks optimization-based multi-layer perceptron neural network. *Journal of Rock Mechanics and Geotechnical Engineering, 13(6),* 1413–1427.
27. Li, M., Andersen, D. G., Smola, A. J., Yu, K. (2014). Communication efficient distributed machine learning with the parameter server. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. (Eds.),

Advances in neural information processing systems, vol. 27. https://proceedings.neurips.cc/paper_files/paper/2014/file/1ff1de774005f8da13f42943881c655f-Paper.pdf

28. Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J. et al. (2019). Megatron-LM: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053.
29. Jia, Z., Zaharia, M., Aiken, A. (2019). Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems, 1,* 1–13.
30. Zhang, J., Hua, Y., Wang, H., Song, T., Xue, Z. et al. (2023). Fedala: Adaptive local aggregation for personalized federated learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, pp. 11237–11244. Washington DC, USA.
31. Do, T. N. (2022). Incremental and parallel proximal SVM algorithm tailored on the jetson nano for the imagenet challenge. *International Journal of Web Information Systems, 18(2/3),* 137–155.
32. Shi, H., Wang, H., Ma, R., Hua, Y., Song, T. et al. (2023). Robust searching-based gradient collaborative management in intelligent transportation system. *ACM Transactions on Multimedia Computing, Communications, and Applications, 20(2),* 1–23.
33. Zhang, R., Chu, X., Ma, R., Zhang, M., Lin, L. et al. (2022). OSTTD: Offloading of splittable tasks with topological dependence in multi-tier computing networks. *IEEE Journal on Selected Areas in Communications, 41(2),* 555–568.
34. Guo, H., Wang, H., Song, T., Hua, Y., Lv, Z. et al. (2021). Siren: Byzantine-robust federated learning via proactive alarming. *Proceedings of the ACM Symposium on Cloud Computing*, vol. 21, pp. 47–60. New York, NY, USA, Association for Computing Machinery.
35. Ao, L., Izhikevich, L., Voelker, G. M., Porter, G. (2018). Sprocket: A serverless video processing framework. *Proceedings of the ACM Symposium on Cloud Computing Machinery*, pp. 263–274. New York, NY, USA, Association for Computing Machinery. https://doi.org/10.1145/3267809.3267815
36. Romero, F., Zhao, M., Yadwadkar, N. J., Kozyrakis, C. (2021). Llama: A heterogeneous & serverless framework for auto-tuning video analytics pipelines. *Proceedings of the ACM Symposium on Cloud Computing*, pp. 1–17. New York, NY, USA, Association for Computing Machinery. https://doi.org/10.1145/3472883.3486972
37. Ali, A., Pinciroli, R., Yan, F., Smirni, E. (2020). BATCH: Machine learning inference serving on serverless platforms with adaptive batching. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. Atlanta, GA, USA, IEEE.
38. Yu, M., Jiang, Z., Ng, H. C., Wang, W., Chen, R. et al. (2021). Gillis: Serving large neural networks in serverless functions with automatic model partitioning. *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pp. 138–148. DC, USA, IEEE.
39. Zhao, H., Pan, S., Cai, Z., Chen, X., Jin, L. et al. (2023). faaShark: An end-to-end network traffic analysis system atop serverless computing platforms. *IEEE Transactions on Network Science and Engineering,* 1–12. https://doi.org/10.1109/TNSE.2023.3294406
40. Pan, S., Zhao, H., Cai, Z., Li, D., Ma, R. et al. (2023). Sustainable serverless computing with cold-start optimization and automatic workflow resource scheduling. *IEEE Transactions on Sustainable Computing,* 1–12. https://doi.org/10.1109/TSUSC.2023.3311197