



REDUCED WORKLOAD ALLOCATION RESPONSE TIME IN EDGE-CLOUD ENVIRONMENT USING ARTIFICIAL INTELLIGENCE

Sarah A. Rafea ¹, Ammar D. Jasim ²

¹ Ministry of Trade, Department of Information Technology, Baghdad, Iraq

² Department of Computer Networks Engineering, College of Information Engineering, Al-Nahrain University,
Jadriya, Baghdad, Iraq

sara.ammar@coie-nahrain.edu.iq¹, ammar.alaythawy@nahrainuniv.edu.iq²

Corresponding Author: Ammar D. Jasim

Received:06/12/2023; Revised:13/03/2024; Accepted:30/04/2024

DOI:[10.31987/ijict.8.2.272](https://doi.org/10.31987/ijict.8.2.272)

Abstract- Edge-cloud computing paradigms increase the Quality of Experience (QoE) for real-time applications by offering many benefits, such as reducing the response time. Many strategies are proposed to handle the data in the edge cloud environment. Hence, where to execute the data generated by the end device is considered an important issue. In this paper, an Artificial Intelligence (AI) model is proposed based on a neural network for workload allocation decisions. The model deals with an AI application that runs on an edge server and a cloud center. The model was trained using a pre-generated dataset based on several features. The features considered are the data size, model complexity, application priority, edge server utilization, and delay of execution on the edge and the cloud. The model decides where to perform the task generated by the end device, either in the edge server or on the cloud. Four AI applications are considered. The model has been implemented using Tensorflow platform with the required libraries. The proposed model employee multi-feature with multi-application addressing workload allocation decisions in a hybrid edge-cloud environment by creating and utilizing a dataset based on proposed algorithm. The proposed model achieved accuracy reached 98.3% and reduced the response time of task execution compared to the based line approach considered in this paper.

keywords: Artificial intelligence, Workload allocation, Cloud computing, Edge computing, Neural Network, AI applications.

I. INTRODUCTION

Edge computing strongly emphasizes real-time computing close to the end devices in light of the rapidly developing user-end Internet of Things (IoT) [1]. A typical distributed cloud and edge computing hierarchical framework comprises three levels: cloud cluster, edge server, and end devices. Artificial intelligence (AI) technology is commonly employed in edge computing to support edge intelligence for better user experience and performance acceleration [2]. Most edge AI applications, such as autonomous cars, are latency-sensitive and have stringent end-to-end response time requirements. End-to-end response time in this context refers to the whole amount of time required, including processing time on various tiers, from the moment an end-user request is received until the end-users are informed of the outcome. Low response time is still a demand, even for edge applications without strict time constraints, like smart homes. Therefore, it has become crucial to figure out how to speed up the response time of these edge AI applications [3].

The benefits of edge computing go beyond reducing response time and include improving bandwidth utilization and enhanced privacy and security. However, edge devices often have limited computational resources, leading to difficulty in processing complex tasks. The general architecture of edge computing is presented in Fig. 1. The edge device layer

represents the end device that collects data by interacting with its environment. The edge server layer is located in the same location or near the end device to provide processing capability at the network's edge for workload processing. The edge server offers low response time while having low processing power and low storage capability. The cloud center layer provides high latency with high processing power and high storage. The location of the cloud server is far away from the edge server. It is generally used for storing critical data and processing complex tasks such as training complex deep learning models. Incorporating AI and edge devices offers several benefits, including advanced analysis capability, allowing the edge devices to process and analyze data in real-time, hence producing real-time intelligent decision-making at the edge layer [4]. The most important AI application benefits from the edge system are in the field of autonomous vehicles.

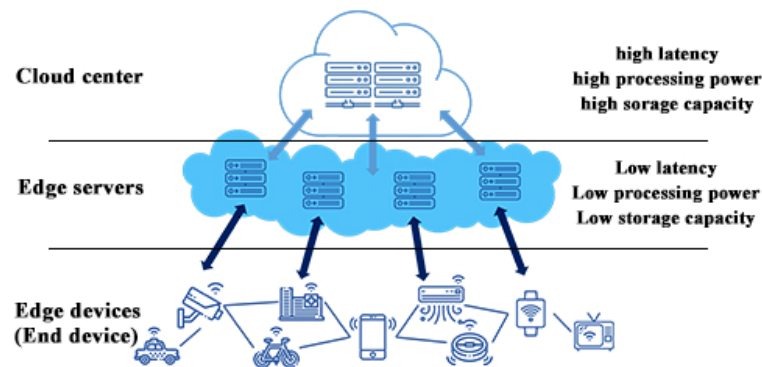


Figure 1: Edge computing system architecture.

The self-driving car in autonomous vehicles generates massive amounts of data that need to be processed and analyzed in real-time. Another important AI application benefits from edge system in smart cities. The data is generated from the IoT devices and sensors deployed in the city from various sources, such as environmental, surveillance, and traffic sensors. In this case, the data can be processed locally and in real-time, offering fast response time and proactive decision-making, hence enhancing overall efficiency.

Moreover, AI can be implemented in the healthcare system at the edge, providing instance monitoring of health data and vital signs by collecting these data from medical sensors and wearable devices. Real-time analysis can be achieved in this collaboration between AI and edge devices in the healthcare system. It can be used to detect anomalies, predict potential health risks, and provide personalized healthcare recommendations. This enables timely interventions, reduces the burden on healthcare professionals, and ultimately improves patient outcomes [5]. The location of executing the workload generated by the AI application mentioned previously among edge servers and the cloud for each IoT application influences how quickly requests are processed due to the restricted processing or storage capabilities on the edge devices side and the size of the data that needs to be processed. The workload distribution becomes a crucial aspect influencing the Quality of Experience (QoE) of the user [6]. The AI application increased and integrated with edge computing systems in many ways to increase the application's performance, such as proposing workload allocation methods and lightweight AI models.

Many approaches have been adopted to increase the performance of the AI application, hence decreasing the response time, increasing the QoE, and increasing the application accuracy while decreasing the energy consumption, enabling these applications to work in a resource-constrained environment.

The rest of this paper is organized as follows: Section II presents related works. In Section III, the proposed workload allocation mechanism is introduced. The performance evaluation of the proposed method is presented in Section IV. Finally, the conclusion is presented in Section V.

II. RELATED WORKS

Many efforts have been introduced to solve the problem of workload allocation for various scenarios using AI methods. In 2020, a workload allocation mechanism was proposed by considering the end device, edge server, and cloud to minimize the response time of the emergency room or Intensive Care Unit (ICU) belonging to medical applications. The proposed algorithm optimizes the response time for several applications belonging to the medical area, such as short-of-breath alerts, patient phenotype classification, and life-death threats, while considering the priority and model complexity of the application. The result of the proposed algorithm compared to the basic allocation approach. The basic allocation approach includes executing the tasks on the edge server or the cloud. The proposed method achieved experimental results show that the allocation strategy will greatly influence the response time of the workload. Compared with other strategies, the results demonstrate the effectiveness and efficiency of the proposed workload allocation strategies [7].

In 2021, a workload allocation method was proposed to optimize the response time of several AI applications, considering model complexity and application priority. The proposed method considers six applications and is implemented in a real edge computing environment. The results indicate the effectiveness of the proposed algorithms in real-life applications compared to the basic allocation algorithm and achieve higher performance from 33% to 63% [8].

Other work in the same year focuses on minimizing the response time and energy consumption of IoT devices by proposing mathematical algorithms that consider the processing time, transmission time, and energy consumption in the decision process. The task offloading decision selects cloud or Mobile Edge Computing (MEC) server or IoT devices to achieve better execution time with lower cost. The author used different numbers of edge servers up to 5. The simulation results indicate that the proposed algorithm achieved up to 51.6% reduction in energy consumption and a task completion time reduction of up to 86.6% [9].

Other work focuses on proposing a workload allocation method aimed to increase the accuracy of the AI application in energy-constraint devices, hence, the IoT devices. The authors propose a method to decide if the prediction result is trustworthy or not, if not it will send the server for a more accurate result. The results indicate that the proposed method is implemented in several well-known deep neural network models. The experiment results show that the proposed method can achieve up to 3.93% inference accuracy enhancement under a specific power constraint compared to previous methods considered in the research [10].

A machine learning approach for workload allocation in Vehicular Edge Computing (VEC) was proposed in [11]. The proposed model considers the task completion success rate and service time by using several metrics, such as the upload/download sizes, computational footprints of the tasks, the LAN, MAN, and WAN network models, and mobility. The

implementation was carried out using EdgeCloudSim simulator and considering a high number of vehicles, up to 1800 vehicles. The results show that the proposed approach achieved better performance in terms of average serves time, average failure task, and average QoE compared to several methods such as game theory-based approach, random approach, Simple Moving Average (SMA) based approach, and the Multi-Armed Bandit (MAB) theory-based approach.

In 2022, deep learning-based algorithms called (DSLO) for offloading decisions were proposed for the MEC environment. Two models proposed in this work are based on the DSLO, which are CNN-DSLO and DNN-DSLO. The authors also propose batch normalization to speed up the model convergence process. The results show that the DSLO requires a few training samples and can quickly adapt to new MEC scenarios, achieving better utility compared to the Random algorithm and Greedy algorithm [12].

According to the previous work, some work focuses on vertical workload allocation, considering three layers of hierarchy: the device, the edge, and the cloud server. While other work focuses on the horizontal workload allocation and considers the allocation between edge servers with high mobility of the end device, hence MEC, Some authors who consider vertical workload allocation propose a mathematical approach for the AI application, including response time and model complexity, without considering the server utilization of the edge server. Some other authors also include energy consumption as a metric in the case of using three levels of decision (including the end device). Other authors employ the AI technique for MEC and scenarios with a large number of devices in the case of horizontal workload allocation. This paper considers the workload allocation between edge and cloud servers, regardless of the end device's capabilities. The proposed method is based on AI algorithms and unlike the previous work, considers several metrics belonging to AI applications, which are application priority, edge server utilization, execution time in the edge server, execution time in the cloud, and model complexity.

III. PROPOSED WORKLOAD ALLOCATION MECHANISM

The proposed workload allocation mechanism is based on one of the AI techniques, which is a neural network. The proposed mechanism is based on creating an AI model for deciding where to process the workload generated by the AI application. The goal is to select the layer for the execution that guarantees the minimum response time while simultaneously considering the priority of the application, the model complexity, and the edge server utilization. The AI application considered in this research is face recognition, action detection, patient phenotype classification, and short of breath alert. The model has been created and trained using a pre-generated dataset. Six features are used in the dataset generated using a Python program. Generally, the first step of the proposed workload allocation mechanism is creating a dataset by using a proposed algorithm, then design and train the AI model, finally use the AI model for inference process hence deciding the task location of execution.

A. Type of AI applications

There are numerous AI applications across various domains, such as Natural Language Processing (NLP) [13], Computer Vision [14], Healthcare and Medicine [15], Robotics [16], and Autonomous Vehicles [17]. In this work, the AI application considered is the short of-breath alert, and patient phenotype classification from health care application, face identification,

and action detection from computer vision application [8].

The Short of Breath alert application is one of the ICU patient monitoring applications to give an alert if the patient will have short of breath in the future. The Long Short-Term Memory (LSTM) model is used in this application. As a result, this application has a high priority. The Patient phenotype classification was used to conduct a 25 separate binary classification task using the LSTM model. The binary classification tasks are not urgent compared with the short of breath alert application, so a lower priority weight is set for this application. For the two above applications, the MIMIC-III [18] as the real-world dataset is used. MIMIC-III includes vital signs and other medical information for over 60000 ICU stays of 40000 unique ICU patients. Face recognition uses Labeled Faces in the Wild [19] as the dataset and FaceNet [20] as the network model. Action detection uses UCF101 [21] as the dataset and ResNet18 as the network mode.

B. Dataset creation

The lack of the dataset and the unavailability of a generalization approach since the data for each system are different in their features. The dataset for the system presented in this work is created based on several features, including task size, application model complexity, priority, utilization, and response time in the edge and cloud. The range of task size for each application is specified since the task size for heavy applications such as face identification and action detection are high compared to other applications considered in this study.

The priority of each application is set according to the importance of the application in the implementation process. For example, if these four applications are implemented for a hospital, then the more important application is the short-of-breath alert, so it takes 1 as a priority, while patient phenotype classification sets 3 as a priority since it is less important, and for the action detection and face identification priority is 2. Maybe in a high-security establishment such as a military facility, the action detection takes priority 1. The priority in this work ranged from 1 to 3. The utilization is an independent variable ranging from 1 to 100, indicating the edge server load at a specific time; hence, if the server is loaded, then the edge server considers sending the tasks to the cloud while the preference to execute the tasks on the edge server off course the other features also considered in the decision process in order to generate the labels for these features.

1) *The complexity of models and devices*: The model complexity is computed by computing the model's FLOPS (floating-point operation per second) [22]. The AI application is based on a different model, so the *FLOPs* of the convolution kernel (*FLOPs_C*) are computed by multiplying the number of parameters with the size of the feature map for the convolution kernel as presented in Eq. (1) as in [22], while the *FLOPs* of fully connected layer (*FLOPs_F*) is equal to the number of parameters as presented in Eq. (2) as in [22]:

$$FLOPs_C = 2 \times H \times W \times (C_{in}K^2 + 1) \times C_{out} \quad (1)$$

$$FLOPs_F = (2 \times I - 1) \times O \quad (2)$$

Here, $H, W, C_{in}, K, C_{out}, I$, and O represent the height, width, number of channels of input feature maps, kernel width, number of output channels, input dimension, and output dimension, respectively. The edge and cloud server computational ability are also considered and computed based on the devices' specifications. The specification and the complexity of the

devices (edge and cloud) are presented in Table I and measured in FLOPS [23]. The FLOPS of servers (FLOPS_S) are calculated according to Eq. (3) adopted from [23]:

$$FLOPS_S = \text{operating frequency} \times \text{processor core} \times \text{operations per cycle} \quad (3)$$

TABLE I
The specification of the servers

Server	Specifications	GFLOPS
Edge server	8 × 2.20 GHz Intel(R) Xeon(R) Silver 5220 CPU cores, 8GB DDR4 RAM	563.2 CPU-GFLOPS
Cloud center	16 × 2.20 GHz Intel(R) Xeon(R) Gold 5220 CPU cores, 1 GeForce RTX 2080 Ti with 4352 CUDA cores, 128GB DDR4 RAM	1126.4 CPU-GFLOPS 110 GPU-TFLOPS

2) *The delay time calculations:* The delay of execution of the tasks on the edge server and the cloud is computed based on Eqs (4 and 5) as in [8]. The delay of executing the task on the edge server represents the processing delay. In contrast, the delay of executing the task on the cloud represents the time needed for executing the task on the cloud and the transmission time from the edge server to the cloud [8].

$$Delay_{on_edge} = (datasize \times MC(FLOPs)) / FLOPS_S(FLOPs/sec) \quad (4)$$

$$Delay_{on_cloud} = (datasize \times MC(FLOPs)) / FLOPS_S(FLOPs/sec) + (datasize/BW) \quad (5)$$

Where MC refer to the model complexity, and its value could be either $FLOPs_C$ or $FLOPs_F$ depend on the model type.

3) *Decision-making algorithm:* A decision-making algorithm used to decide where the task should be executed, on edge or the cloud, for creating the dataset is implemented using Python programming language and represents the label of the data. Algorithm 1 presents the decision process.

Algorithm 1: Execution Layer Selection Algorithm

Input: *priority, utilization, delay on edge, delay on cloud*

Output: *execution layer*

```

1      Diff = abs(delay on edge – delay on cloud)
2      Thr = threshold for selecting the difference in delay between edge and cloud
3      If ((Diff ≤ Thr) & (priority > 1))
4          If (utilization > 90)
5              Select the cloud
6          Select the layer with the smallest delay
7      Select the layer with the smallest delay
8      If ((priority == 1) & (utilization < 95))
9          Select the layer with the smallest delay
10     Else if ((priority == 1) & (utilization ≥ 95) & (Diff ≤ Thr))
11         Select the cloud
12     Select the layer with the smallest delay

```

C. Proposed model architecture

The proposed model architecture is presented in Fig. 2. The model contains 5 layers. The input layer contains 6 neurons, while the 3 hidden layer contains 8, 4, and 4 neurons, respectively. The output layer has one neuron with a sigmoid activation function. The proposed model is a binary classification model. The output of the proposed model gives a probability of execution in the edge server layer. The proposed model is implemented in the edge server. Hence, decide to execute locally in the server or send the task to the cloud. The model has been trained based on the 6 features presented in the dataset creation section. The network's topology, considered present in Fig. 3, consists of AI applications that send its task to

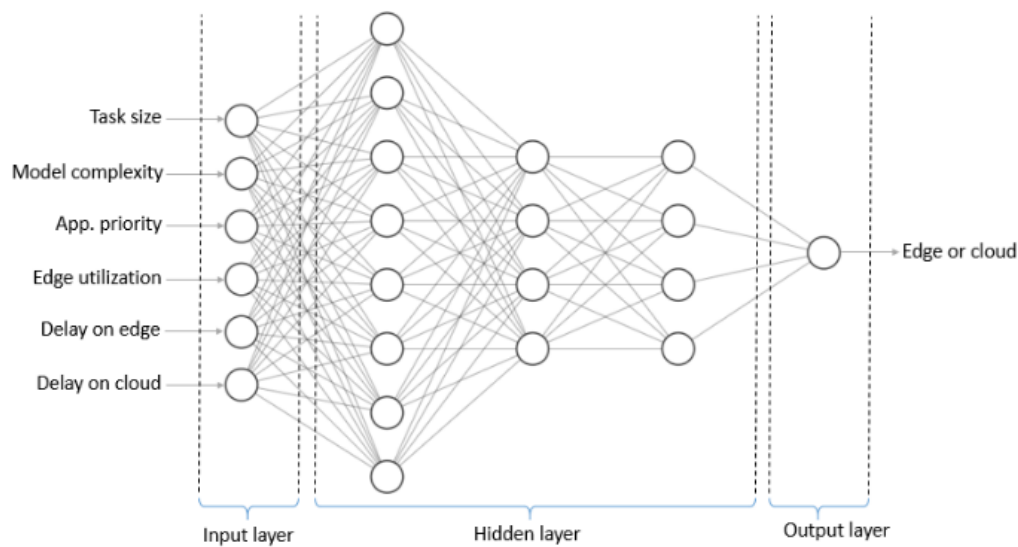


Figure 2: The proposed neural network model.

one edge server. The applications send all of their tasks or part of the task to the edge. The end device may enable the execution of tasks in the end device itself or part of the tasks. The decision on the end device was not considered in this research and did not affect the proposed mechanism. The edge server received all tasks from all applications and then used the proposed model to decide where to execute the tasks.

IV. SIMULATION TEST AND RESULTS

There are many tools and frameworks to implement AI models, such as Scikit Learn [24], TensorFlow [25], Caffe [26], Keras [27], PyTorch [28], and Google ML Kit [29]. In this research, TensorFlow framework and many libraries, such as Pandas and Numpy are used for training the model using Python programming language. TensorFlow is an end-to-end open-source machine learning platform to build and deploy AI models easily.

Table II shows the main parameter of the model. The neural network implemented consists of an input layer, three hidden layers, and one output layer for predicting the task execution layer. Many neural topologies were tested, and the more

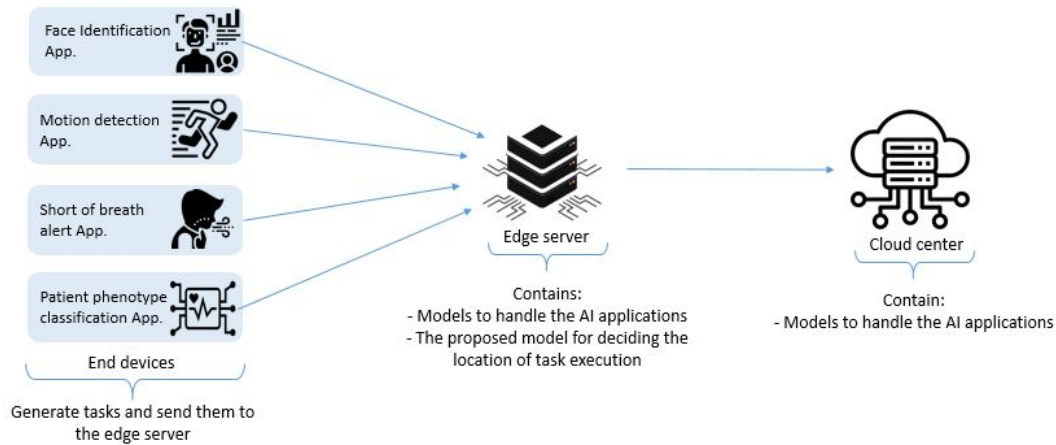


Figure 3: Network topology.

complicated network provided nearly the same result as the one tested in the research. In contrast, the less complicated neural network produces a relatively less accurate model. The activation function used in the last layer for classification is the sigmoid activation function [30], which produces a probability of being executed in the edge server or the cloud. The optimizer used is Adam optimizer [31] to reduce the Mean Square Error (MSE). The epoch is the total number of iterations of all the training data in one cycle for training set to 100, and the patch size indicates the number of samples taken to work through a particular model before updating its internal model parameters set to 32.

TABLE II
The main parameters of the model

Parameter	Specification or Value
Neural network layers	5 (1-input, 3-hidden, 1-output)
Activation function	Layer1,2,3,4-relu, 5-sigmoid
Optimizer	Adam
Dataset size	1200 records (60%training, 20% validation, and 20% testing)
Epoch	100
Patch size	32
Number of AI applications	4
Number of Edge server	1

A. Accuracy and loss of training and validation

The model performance measured by calculating accuracy, loss, precision, recall, and F-score [32]. The model's accuracy is presented in Fig. 4. As presented in the figure, the model converges fast and stable at around 98% accuracy. After 10 epochs, the model accuracy reached around 94% accuracy, and after 60 epochs, the accuracy reached 98% and stayed stable in this range. The validation accuracy is nearly the same as the training accuracy after 60 epochs, indicating that

the model converges well and there is no overfitting.

The loss of the model presented in Fig. 5, as present in the figure, the model converged fast until it reached 10% loss after 30 epochs and was stable at around 0.03% loss after 80 epochs. The loss of validation is nearly the same as the loss of training.

Fig. 6 presents the confusion matrix of the model, which predicts a categorical label for each input. The confusion matrix presents the number of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) produced by the model on the test data. The values of TP, TN, FP, and FN are 106, 130, 3, and 1, respectively. Table III. presents

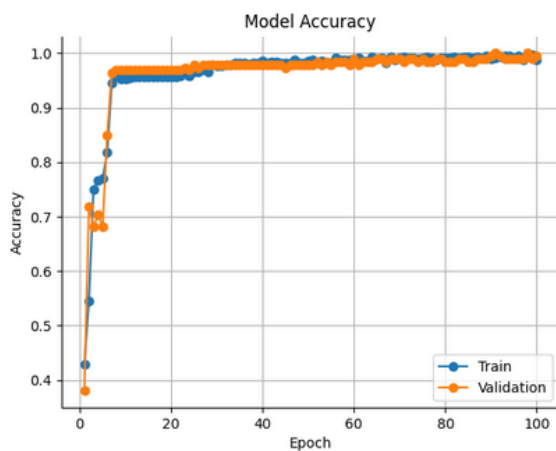


Figure 4: The accuracy of the proposed model.

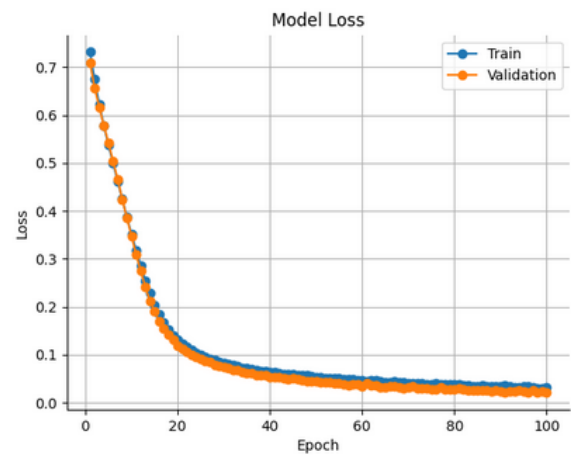


Figure 5: The loss of the proposed model.

106	3
1	130

Figure 6: The confusion matrix of the proposed model

the average results for accuracy, loss, precision, recall, and F-score. The average accuracy is 98.33, the average loss is 0.03, the precision is 0.9816, the recall is 0.9906, and the F1-score is 0.9860. the precision, recall, and F-score.

B. Task execution time

The task execution time for executing tasks belongs to all AI applications considered and calculated in more than one case. The first is if all tasks are sent to the cloud, the second one is in case all tasks are sent to the edge, and the final case is by using the proposed model.

TABLE III
 The main parameters of the model

Parameter	Value
Accuracy	98.33
Loss	0.03%
Precision	0.9816
recall	0.9906
F-score	0.9860

Fig. 7 presents the delay time for executing the tasks belonging to the short-of-breath alert application in cloud, edge, and the proposed model. The size of the task is relatively small; hence, the execution time on the cloud takes more time because of the transmission time from the edge server to the cloud, while the edge server can process the tasks faster, producing a lower execution time for all tasks, which is also worth to notice that the proposed method select the edge server for the execution since the short of breath alert has the highest priority to be executed with the lowest execution time even if the edge server utilization is high.

Fig. 8 presents the delay time for executing the tasks belonging to the patient phenotype classification application in cloud, edge, and the proposed model. The task size is relatively medium since the execution time on the cloud takes less time because the resources of the cloud are higher than the edge server, and the data transmission time from the edge server to the cloud with the execution on the cloud is relatively shorter than the time of execution on the edge server. The main factors influencing task completion time are the device capability (edge server and cloud) and the transmission rate from the edge server to the cloud. The proposed method selects the cloud for execution in some cases and selects the edge server as the execution layer for other tasks. The target execution layer is affected by the application priority and server utilization along with the task size and delay time of completion task in both edge server and cloud. The patient phenotype classification is considered a low-priority application; hence, in cases where the edge server is loaded, the task will be sent to the cloud even if the execution time on the cloud takes more time than the execution time in the edge server in a small difference in time. Fig. 9 presents the delay time for executing the tasks of the Action detection application in cloud, edge, and the proposed model. The task size is relatively high; hence, the execution time on the edge server takes more time compared to the cloud, with a high difference, causing the target execution layer to be the cloud server. It's also worth noticing that the action detection application has medium priority. So, the task size is so high, which causing execution delay on the edge server to be unacceptable in some cases.

Fig. 10 presents the delay time for executing the tasks belonging to the face detection application in cloud, edge, and the proposed model. The task size is relatively large; hence, the execution time on the edge server takes more time than the cloud, with a high difference causing the target execution layer to be the cloud. The face identification application, similar to action detection, has medium priority. Still, the task size is so high, causing the delay of execution on the edge server to be unacceptable in some cases.

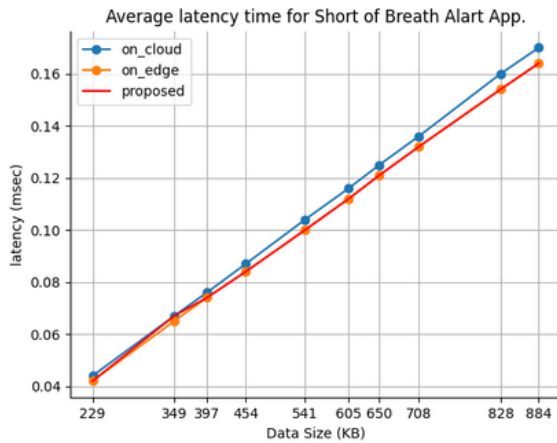


Figure 7: Average latency time for short of breath alert app.

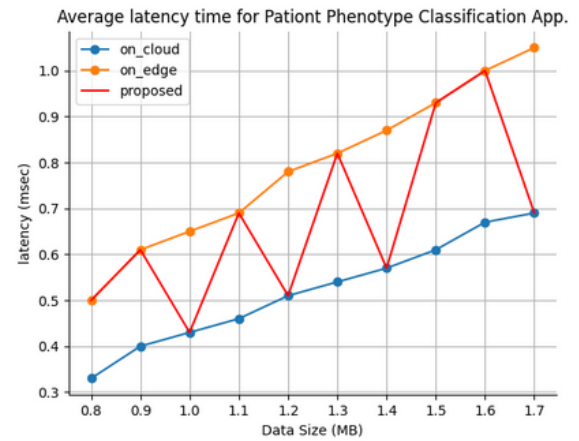


Figure 8: Average latency time for patient phenotype classification.

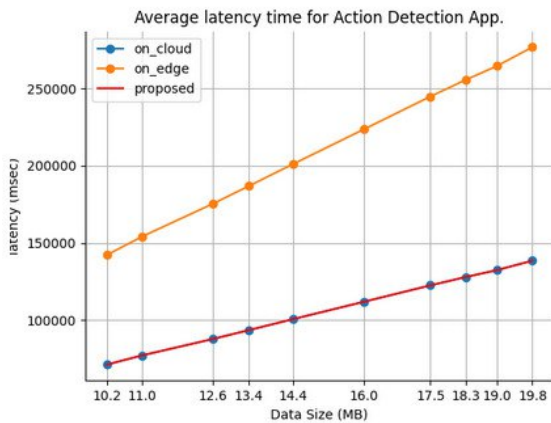


Figure 9: Average latency time for action detection app.

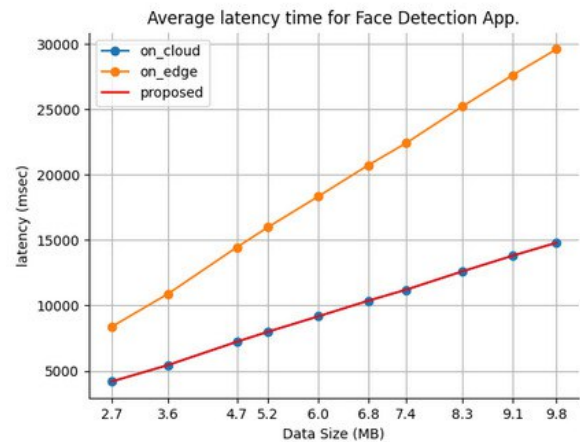


Figure 10: Average latency time for face detection.

C. Proposed model implementation

The predicted layer for execution tasks belonging to different AI applications is presented in Table IV. A Python program is used to randomly generate 15 tasks to send to the edge server. The model is already saved and used for predicting the layer of the execution. Generally, the tasks belong to the short of breath alert (tasks number, 1,2,7,12, and 14) executed in the edge since its task size are small compared to other application and also have the highest priority. The patient phenotype classification has the lowest priority and depends on the server utilization. The tasks may be executed in the cloud even if the time of execution on the edge server is lower than in the cloud (tasks number, 4,10, and 11). Face recognition (tasks number, 3,5,13, and 15) and action detection (tasks number, 6,8, and 9) are considered heavy applications, so generally,

all their tasks are sent to the cloud for execution.

TABLE IV
The specification of the servers

#	Task Size (KB)	App. Name	Delay on Edge (s)	Delay on Cloud (s)	Predicted Execution Layer
1	291	Short of breath alert	0.0542	0.0562	Edge
2	273	Short of breath alert	0.0509	0.0527	Edge
3	8138	Face recognition	24564.2755	12282.9515	Cloud
4	849	Patient phenotype classification	0.5237	0.3467	Edge
5	9266	Face recognition	27969.1051	13985.4791	Cloud
6	15352	Action detection	213706.8181	106854.9442	Cloud
7	820	Short of breath alert	0.1530	0.1585	Edge
8	14565	Action detection	202751.4204	101377.1667	Cloud
9	10788	Action detection	150173.8636	75088.0106	Cloud
10	890	Patient phenotype classification	0.5490	0.3635	Cloud
11	1648	Patient phenotype classification	1.0165	0.6730	Edge
12	353	Short of breath alert	0.0658	0.0682	Edge
13	5889	Face recognition	17775.7457	8888.4617	Cloud
14	510	Short of breath alert	0.0951	0.0985	Edge
15	2757	Face recognition	8321.9105	4161.2309	Cloud

V. CONCLUSION

An AI model for workload allocation decisions between the edge server and cloud center is considered in this work. A neural network is used in this paper and trained using a pre-generated dataset. The features in the dataset considered are the data size, AI model complexity, application priority, edge server utilization, delay of execution on the edge server, and delay of execution on the cloud. Four AI applications are considered: short of breath alert, patient phenotype classification, face identification, and action detection. The AI application is implemented on an edge server and cloud center. The task generated by the end device is sent to the edge server. The edge server runs the proposed AI model to decide where to execute the tasks on the edge server or the cloud.

The model achieved an average accuracy of 98.33%, and the average loss reached 0.03%. The model was tested with many scenarios and newly generated data for all AI applications considered in this work. The proposed model minimizes the latency time of all AI applications compared to the time required to execute the task just on the edge server or on the cloud center. As a future work, other features can be considered, such as the model's accuracy by implementing different models for the same application on edge and cloud, by putting the powerful model on the cloud and the lightweight one on edge. In addition, link reliability could be added as a feature in the dataset, and more than one edge server may be considered.

FUNDING

None.

ACKNOWLEDGEMENT

The author would like to thank the reviewers for their valuable contribution in the publication of this paper.

CONFLICTS OF INTEREST

The author declares no conflict of interest.

REFERENCES

- [1] X. Zhang, Z. Cao, and W. Dong, "Overview of Edge Computing in the Agricultural Internet of Things: Key Technologies, Applications, Challenges," *IEEE Access*, vol. 8, pp. 141748–141761, 2020.
- [2] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-Edge AI: Intelligentizing Mobile Edge Computing, Caching, and Communication by Federated Learning," *IEEE Netw.*, vol. 33, no. 5, pp. 156–165, 2019.
- [3] A. Biswas, and H. Wang, "Autonomous Vehicles Enabled by the Integration of IoT, Edge Intelligence, 5G, and Blockchain." *Sensors*, vol.23, no.4, pp.1963, 2023.
- [4] S. Ammar and A. Dawood, "AI Workload Allocation Methods for Edge-Cloud Computing: A Review", *Al-Iraqia Journal for Scientific Engineering Research*, vol.2, Issue 4, pp. 115-312 ,2023
- [5] T. Hao, J. Zhan, K. Hwang, W. Gao, and X. Wen, "AI-oriented Medical Workload Allocation for Hierarchical Cloud/Edge/Device Computing," *arXiv*, Feb. 09, 2020. Accessed: Jul. 05, 2023. [Online]. Available: <http://arxiv.org/abs/2002.03493>
- [6] T. Hao, J. Zhan, K. Hwang, W. Gao, and X. Wen, "AI-oriented Workload Allocation for Cloud-Edge Computing," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, Melbourne, Australia: IEEE, pp. 555–564, 2021.
- [7] J. C. S. Dos Anjos, J. L. G. Gross, K. J. Matteussi, G. V. González, V. R. Q. Leithardt, and C. F. R. Geyer, "An Algorithm to Minimize Energy Consumption and Elapsed Time for IoT Workloads in a Hybrid Architecture," *Sensors*, vol. 21, no. 9, p. 2914, Apr. 2021.
- [8] Y.-W. Hung, Y.-C. Chen, C. Lo, A. G. So, and S.-C. Chang, "Dynamic Workload Allocation for Edge Computing," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 29, no. 3, pp. 519–529, Mar. 2021.
- [9] C. Sonmez, C. Tunca, A. Ozgovde, and C. Ersoy, "Machine Learning-Based Workload Orchestrator for Vehicular Edge Computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 4, pp. 2239–2251, Apr. 2021.
- [10] S. Yang, G. Lee, and L. Huang, "Deep Learning-Based Dynamic Computation Task Offloading for Mobile Edge Computing Networks," *Sensors*, vol. 22, no. 11, p. 4088, May 2022.
- [11] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.
- [12] W. Yu, F. Liang, X. He, W. Grant, C. Lu, J. Lin, X. Yang, "A Survey on the Edge Computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.
- [13] Y. Kang, Z. Cai, C.-W. Tan, Q. Huang, and H. Liu, "Natural language processing (NLP) in management research: A literature review," *J. Manag. Anal.*, vol. 7, no. 2, pp. 139–172, Apr. 2020.
- [14] A. Esteva, K. Chou, S. Yeung, N. Naik, A. Madani, A. Mottaghi, Y. Liu, E. Topol, J. Dean, and R. Socher, "Deep learning-enabled medical computer vision," *Npj Digit. Med.*, vol. 4, no. 1, p. 5, Jan. 2021.
- [15] D. Lee and S. N. Yoon, "Application of Artificial Intelligence-Based Technologies in the Healthcare Industry: Opportunities and Challenges," *Int. J. Environ. Res. Public. Health*, vol. 18, no. 1, p. 271, Jan. 2021.
- [16] D. Vrontis, M. Christofi, V. Pereira, S. Tarba, A. Makrides, and E. Trichina, "Artificial intelligence, robotics, advanced technologies and human resource management: a systematic review," *Int. J. Hum. Resour. Manag.*, vol. 33, no. 6, pp. 1237–1266, Mar. 2022.
- [17] Y. Ma, Z. Wang, H. Yang, and L. Yang, "Artificial intelligence applications in the development of autonomous vehicles: a survey," *IEEECAA J. Autom. Sin.*, vol. 7, no. 2, pp. 315–329, Mar. 2020.
- [18] A. E. W. Johnson, T. J. Pollard, L. Shen, L. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi and R. G. Mark, "MIMIC-III, a freely accessible critical care database," *Sci. Data*, vol. 3, no. 1, p. 160035, May 2016.
- [19] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, "Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments," 2008.
- [20] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA: IEEE, pp. 815–823, 2015
- [21] M. Ramesh, K. Mahesh, "Sports Video Classification with Deep Convolution Neural Network: A test on UCF101 Dataset." *International Journal of Engineering and Advanced Technology (IJEAT)*, Vol.8 Issue-4S2, April 2019.
- [22] S.-C. Hsia, S.-H. Wang, and C.-Y. Chang, "Convolution neural network with low operation FLOPS and high accuracy for image recognition," *J. Real-Time Image Process.*, vol. 18, no. 4, pp. 1309–1319, Aug. 2021.
- [23] R. Dolbeau, "Theoretical peak FLOPS per instruction set: a tutorial." *The Journal of Supercomputing*, vol.74 no.3, pp.1341-1377, 2018.
- [24] "scikit-learn: machine learning in Python — scikit-learn 1.2.2 documentation." <https://scikit-learn.org/stable/> (accessed Jun. 24, 2023).
- [25] "TensorFlow." <https://www.tensorflow.org/> (accessed Jun. 24, 2023).
- [26] "Caffe | Deep Learning Framework." (accessed Jul. 06, 2023).
- [27] "Keras: Deep Learning for humans." <https://keras.io/> (accessed Jul. 06, 2023).
- [28] "PyTorch." <https://pytorch.org/> (accessed Jul. 06, 2023).
- [29] "ML Kit," Google for Developers. <https://developers.google.com/ml-kit> (accessed Jul. 06, 2023).
- [30] A. D. Rasamoelina, F. Adjailia, and P. Sinčák, "A Review of Activation Function for Artificial Neural Network." *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, p. 281-286, 2020.
- [31] S. Bock and M. Weis, "A Proof of Local Convergence for the Adam Optimizer," in *2019 International Joint Conference on Neural Networks (IJCNN)*, Budapest, Hungary: IEEE, pp. 1–8, 2019.
- [32] Y. Li and Z. Chen, "Performance Evaluation of Machine Learning Methods for Breast Cancer Prediction", *Applied and Computational Mathematics*, vol.7, no.4, pp. 212-216, 2018.