Context-Aware Offloading for IoT Application using Fog-Cloud Computing

Karan Bajaj[1]*, Shaily Jain[2] and Raman Singh[3]

[1]*Assistant Professor, Department of Computer Science & Engineering, Chitkara University School of Engineering and Technology, Chitkara University, Himachal Pradesh, India; karan.bajaj@chitkarauniversity.edu.in*

[2]*Professor, Department of Computer Science & Engineering, Chitkara University School of Engineering and Technology, Chitkara University, Himachal Pradesh, India; shaily.jain@chitkarauniversity.edu.in*

[3]*Lecturer, School of Computing, Engineering and Physical Sciences, University of the West of Scotland, Lanarkshire, Scotland, raman.singh@uws.ac.uk*

***Correspondence:** Karan Bajaj; karan.bajaj@chitkarauniversity.edu.in

ABSTRACT- It is difficult to run delay-sensitive applications and the cloud simultaneously due to performance metrics such as latency, energy consumption, bandwidth, and response time exceeding threshold levels. This is the case even though advanced networks and technologies are being used. The middleware layer of the Internet of Things (IoT) architecture appears to be a promising solution that could be used to deal with these issues while still meeting the need for high task offloading criterion. The research that is being proposed recommends implementing Fog Computing (FC) as smart gateway in middleware so that it can provide services the edge of the networks. Applications that are sensitive to delays would then be able to be provided in an efficient manner as a result of this. A smart gateway is proposed as solution for taking the offloading decision based on the context of data, which offers a hybrid approach in order to make decisions regarding offloading that are efficient and effective. A solution that uses machine-learning reasoning techniques to make offloading decisions, in multiple fog-based cloud environments. Feature selection, and classification are used as a learning method and are also ensembled as hybrid logistic regression-based learning to provide the best offloading solution. It works by learning the contextual information of data and identify the cases to make the decision of offloading. The proposed model offers a solution that is both energy and time efficient, with an overall accuracy of approximately 80 percent. With the proposed intelligent offloading approach, it is expected that Internet of Things applications will be able to meet the requirement for low response time and other performance characteristics.

Keywords: Internet of Things; Cloud Computing; Fog Computing; Offloading; Context-Aware; Logistic Regression.

ARTICLE INFORMATION

Author(s): Karan Bajaj, Shaily Jain and Raman Singh;
Received: 29/11/2022; **Accepted**: 31/01/2023; **Published**: 20/02/2023;
e-ISSN: 2347-470X;
Paper Id: IJEER-2911-11;
Citation: 10.37391/IJEER.110110
Webpage-link:
https://ijeer.forexjournal.co.in/archive/volume-11/ijeer-110110.html

Publisher's Note: FOREX Publication stays neutral with regard to Jurisdictional claims in Published maps and institutional affiliations.

1. INTRODUCTION

A scenario in which components share information through sensing devices, actuators, and processing units that communicate with one another in order to fulfil a meaningful purpose or complete work that requires a significant amount of intelligence with the minimum amount of human involvement is known as the Internet of Things (IoT) [1]. The goal of the Internet of Things (IoT) is to automate everything and connect all technologies. It is also meant to make all physical objects smart, so they can connect to each other and talk to each other, as well as make smart decisions on their own.

The processing of enormous amounts of data at the edge or in the cloud causes latency, and high energy consumption and also devalues other performance parameters that are unacceptable in some essential applications such as health situations, transportation management, and so on. It's clear that applications that do a lot of processing use more energy, drain batteries faster, and make solutions more expensive [2], but it's also true that some applications need more energy to do the processing.

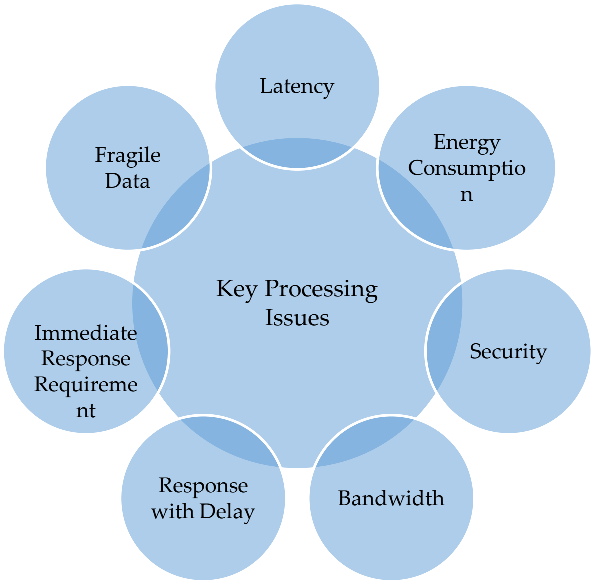


Figure 1: Key issues faced for processing data at cloud servers

Figure 1 highlights the key issues encountered when processing data solely through cloud data centers, mainly related with latency, delay and energy consumption, whereas when processing data solely through edge/fog computing architectures, including the challenges mentioned in *figure 1*, scarcity among resources and processing capacity constraints become a concern. Current research is focused towards latency, energy consumption and immediate response requirement challenges faced by the applications and providing the solution in the form of offloading strategy. Computation offloading is a strategy designed to reduce device energy consumption and processing time. One of the most difficult aspects of offloading is determining whether offloading is helpful. One of the primary issues is learning from the context of data or recognizing the context or scenario for computational and data offloading. The design and development of a context-aware offloading approach for computationally expensive Internet of Things applications is required in order to make effective optimal offloading decisions.

The intelligent offloading methodology with the proposed context-aware framework is supposed to achieve the requirement to meet the low response time, energy efficiency, and other performance parameters for IoT applications in different domains like healthcare, smart city, agriculture, etc. Because of the massive volume of data being generated, there is a lot of traffic of produced data that leads to the challenge of congestion among the network and results in high latency. The increase in time required for the round-trip causes delays in big data transmission volumes and high hop counts between IoT devices and cloud servers, which makes application data for providing services meaningless and unsuitable for end-users to understand.

1.1 Smart gateway as offloading solution

For a large number of applications that require a high amount of computations, there is a need for a third party to perform the processing and executions on behalf of the client devices and provide results after processing [3]. This methodology is being referred to as offloading, where an external entity performs the tasks, and can be referred to as outsourcing of operations. Computation offloading in the middleware layer, which is responsible for processing and storage, plays a significant role in supporting the delay-sensitive and context-aware services in IoT applications.

Despite usage of 5G networks and technologies, delay-sensitive applications and the cloud are unable to run simultaneously due to performance metrics such as latency, energy consumption, bandwidth, and response time exceeding threshold levels [4]. As a solution to counter the issues, the middleware layer of the IoT architecture appears to be a promising approach for dealing with these challenges while still meeting the requirement of high task offloading criteria.

Current research proposes a smart gateway deployment in middleware as it provides services to the network's edge, allowing delay-sensitive applications to be efficiently served. FC in itself suffers from a disadvantage, as Fog nodes, on the other hand, have a limited resource that aren't able to do all tasks, particularly those that require a lot of computing. Furthermore, fog is not considered as an alternative or substitute to the cloud but rather considered as an add-on, combining both the technologies to take advantage of their joint benefits. The proposed framework presents a smart gateway deployment that work as solution taking advantage of cloud and fog computing technologies together by making offloading decision based on the contextual information of the received data.

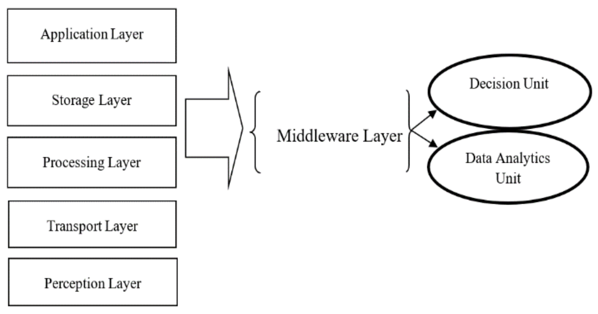


Figure 2: IoT architecture with middleware layer role for the proposed offloading method

Figure 2 shows the general five layers of basic IoT architecture. The bottom layer, perception & sensing, integrates devices and objects with sensors for data collection. It connects the real and digital worlds also termed as the physical layer. Real-time data is collected for processing. Transport layer moves data between devices and objects. Huge sensors generate a massive amount of data, requiring IoT systems to be flexible and high-performance to support different protocols among devices. IoT systems offer fast transactional services, context-aware applications, etc. Processing Layer which act as a middleware, analyses and processes transport layer data. This layer analyses and processes work with many technologies like edge, femto, fog, and cloud computing. Storage layers provide temporary data storage, duplication, and distribution. The application layer provides users with IoT application services. *Figure 2* shows the importance of middleware with its role as a decision and analytics unit, that can be played while performing the offloading.

2. LITERATURE SURVEY

Edge computing stresses local computing and employing various surrounding devices/architectures as edge servers to provide timely and intelligent services. To fully realise edge computing in IoT applications, many challenges must be addressed, including how to efficiently distribute and manage data storage and computing, how to make edge computing collaborate with cloud computing for more scalable services, and how to secure and preserve system privacy. Offloading frameworks and models vary by offloading strategy (edges, fog, cloud) and working parameters (energy, duration, battery use). Many offloading frameworks and models have been studied to analyse the key issues and possibilities with the current approaches.

Some of the initial works are done by the MAUI offloads code to make cellphones last longer [5]. Here, authors show how system splits programs, profiles them, and formulates and solves programme partitioning to enhance energy advantages. This framework simplifies cloud migration of smartphone apps. Thinkair [6] uses cloud smartphone virtualization, dynamic resource allocation, and parallel execution for code offloading. Lin et al. [7] use a decision engine-based context-aware decision algorithm (CADA) to decide whether to offload a procedure to the cloud.

Another model Evidence-Aware Mobile Computational Offloading (EMCO) [8] is a unique computational offloading tools and framework. Dedicated computing infrastructure offloads computationally heavy operations to surrogates in a cloud-based model.

Context-Sensitive Model for Offloading System (CoSMOS) [9] methodology uses context-aware self-adaptive offloading for mobile cloud computing (MCC) systems. Decision-taking estimates is used to improve application execution time and energy consumption.

Junior et al. [10] present a machine learning-based Context-Sensitive Offloading System (CSOS) using J48, JRIP, IBK, and Naive Bayes reasoning. The authors implemented J48 and JRIP because they provide accurate offloading decisions. The most accurate offloading decision-making solution is chosen among multiple categorization methods. It processes context data into high-level context representation. Healthcare applications are latency-sensitive and demand quick processing, hence the Fog Computing (FC)-based analytical paradigm [11] is suggested. IoT data transmission requires lower communication, processing, and network latency than cloud-server architecture can provide. Data processing and analysis using edge structures reduces latency. Analytical model and hybrid fuzzy-based reinforcement learning technique using Neural Networks (NN) approaches in FC environment.

Many models and frameworks aim to save time and energy. Offloading frameworks include two broad categories [12]. Client-server and VM cloning frameworks. A full image transfer of the application, including the operating system, is made on the cloud server in VM clone. The client and cloned VM state are merged to resume execution after the task. Client-Server frameworks offload logic by pre-installing the application component on the server device. Local state is shared server-wide before task execution and synchronised with the client afterward [13]. Offloading frameworks differ mostly in their implementation methods.

Based on literature survey, it can determine that the Client-server model is a superior option to the VM model for the purpose of offloading data. This is because the Client-server model reduces the amount of bandwidth that is consumed and is closer to the real-world contexts that are required for data offloading. Additionally, the vast majority of frameworks are mobile-based; hence, mobile and gaming applications are most frequently chosen for the purpose of implementation when dealing with cloud-based offloading scenarios.

Based on the reviews of the relevant literature, it has been determined that very little work has been done on edge structures and edge-based cloud scenarios, and the working models and frameworks that are based on them do not concentrate on the smart requirements or context-based offloading approach.

3. DESIGN OF SMART GATEWAY ARCHITECTURE

Figure 3 demonstrate the smart gateway role for the offloading process, where smart gateway in middleware takes the offloading decision based on the received data from the applications. Under the current section, the proposed smart gateway receives the applications-based data from the sensory devices representing IoT applications and take the offloading decision accordingly.

The framework on middleware is composed of three major components: The Training, Decision Engine, and the Offloading. These components communicate with one another to perform context-sensitive offloading.

Materials and Methods should be described with sufficient details to allow others to replicate and build on published results. Please note that publication of your manuscript implicates that you must make all materials, data, computer code, and protocols associated with the publication available to readers. Please disclose at the submission stage any restrictions on the availability of materials or information. New methods and protocols should be described in detail while well-established methods can be briefly described and appropriately cited.

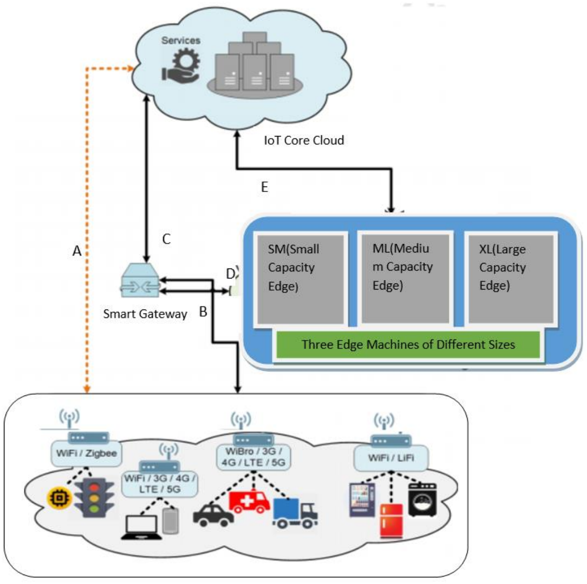


Figure 3: Proposed offloading framework architecture

In figure 3-line A demonstrate the traditional call structure data directly offloaded to the cloud, line B indicates that data first send to the smart gateway and gateway offloading the data to the fog structure or cloud for further processing represented by line C and D.

The smart gateway deployed will be having the learning inference engine, the feature of context identification, and rule generation by processing the data to make a further decision on the received data, either to process it themselves or to the cloud. Smart gateway will first analyze the incoming data and perform a selection of offloading parameters, for this training and testing of the data are done using different Machine Learning (ML) algorithms after classification of parameters, contextual information regarding the offloading of the data is generated using hybrid regression-based reinforcement learning inference system.

4. MATERIALS AND METHODS

The allocation and selection of data packets in an IoT–FC environment is accomplished through the use of a system that incorporates reinforcement learning as logistic regression with k-NN and decision tree evolution algorithms.

4.1 Data Set Description

Data set that is UCI Heart Disease Data, ECG sensor data collected from healthcare facilities were received through the online web-based UCI machine learning repository, it is a research centre dedicated to the study of intelligent systems and machine learning. In the simulated data set ECG sensor data is used for the analysis, which has 16 attributes and 920 instances, that makes up the data in the simulation. The dataset includes data from a patient who was suffering from heart disease as a result of a combination of conditions including high blood pressure, high blood sugar level, and high cholesterol. The data was collected in a continuous stream. ECG strips/records from a total of 920 participants were acquired from two leads in this study. Table 5 presents the attributes description of the dataset with 14 attributes as one attribute is “City” the other one is “ID” which simply means the city of collecting the dataset and the other is a serial number [14].

Data sets from the four different cities:
Cleveland: 303
Hungarian: 294
Switzerland: 123
Long Beach VA: 200
are used with total 920 values.

Table 1: Attributes values and information of the dataset

Attribute Information		
Sr. No.	**Name**	**Description**
1	age	Age of patients in years
2	sex	Patient categorization based on gender : Male, 0: Female)
3	cp	Chest Pain Type -- Value 1: typical angina -- Value 2: atypical angina -- Value 3: non-anginal pain -- Value 4: asymptomatic
4	trestbps	Resting Blood Pressure (mm Hg)
5	chol	SerumCholestrol (mg/dl)
6	fbs	Fasting Blood Sugar > 120 mg/dl (1 : true, 0 : false)
7	restecg	Resting ECG -- Value 0: normal -- Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV) -- Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8	thalach	Maximum Heart Rate
9	exang	Exercise induced angina (1: true, 0:false)
10	oldpeak	ST depression induced by exercise relative to rest
11	slope	Slope of the peak exercise ST segment -- Value 1: upsloping -- Value 2: flat -- Value 3: downsloping
12	ca	Number of major vessels (0-3) colored by flourosopy
13	thal	Thal: (3 = normal, 6 = fixed defect, 7 = reversable defect)
14	(num) (the predicted attribute)	Diagnosis of heart disease (angiographic disease status) -- Value 0: < 50% diameter narrowing -- Value 1: > 50% diameter narrowing

4.2 Evaluation Setup

The experimental simulation was carried out with the help of Anaconda Python, which is a data science platform designed for data scientists, and business decision-makers. It is a collection of programming languages such as Python, R, and other languages. The proposed model is trained using the set of logistic regression algorithms, K-Nearest Neighbor (KNN), Naive Bayes, Decision Tree, Support Vector Machine (SVM), Multilayer Perceptron (MLP), and K-Nearest Neighbor (KNN) algorithms. The proposed model uses hybrid regression algorithms with deep learning where the logistic regression algorithm is one of the machine learning algorithms used to train the model.

4.3 Pre-processing of Datasets

Feature selection are included in the preparation of data. The process of automatically maintaining of a group of characteristics to a modest level so that these features can be modelled is referred to as feature selection.

4.4 Feature Selection

An algorithm is proposed for the smart gateway to take the offloading decision, the proposed algorithm leads to the design and development stages of a predictive model that is based on machine learning. Algorithm 1 shows the step-wise description of the different stages of the smart gateway before classification to prepare data to take the offloading decision.

```
# Algorithm 1
#Algorithm start here:
#Feature Selection
1: Correlation (Pearson correlation adds two objects' deviations from their means and divides by their squared differences)
2: Feature selection (Based on correlation dropping attributes “id, dataset, thalch”)
#Converting features to categorical values
1: slope Values (flat=0, upsloping=1, downsloping=2) && thal Values(reversable defect=0, normal=1, fixed defect=2)
2: Binary data attributes conversion
fbs,sex,exang (TRUE=1,FALSE=0)
#Handling Missing Value by Mean ()
1:Replace_Missing_Value_Mean (Heart Disease)
2: return Heart_Disease['slope','ca','thal', 'exang', 'fbs'].replace('0', mean())
#Splitting training and test dataset
1: Train_Test_Data_Split( train data 75% and test data 25%)
#Algorithm end here.
```

Feature selection refers to the process of selecting the input variables based on the correlation attribute evaluation.

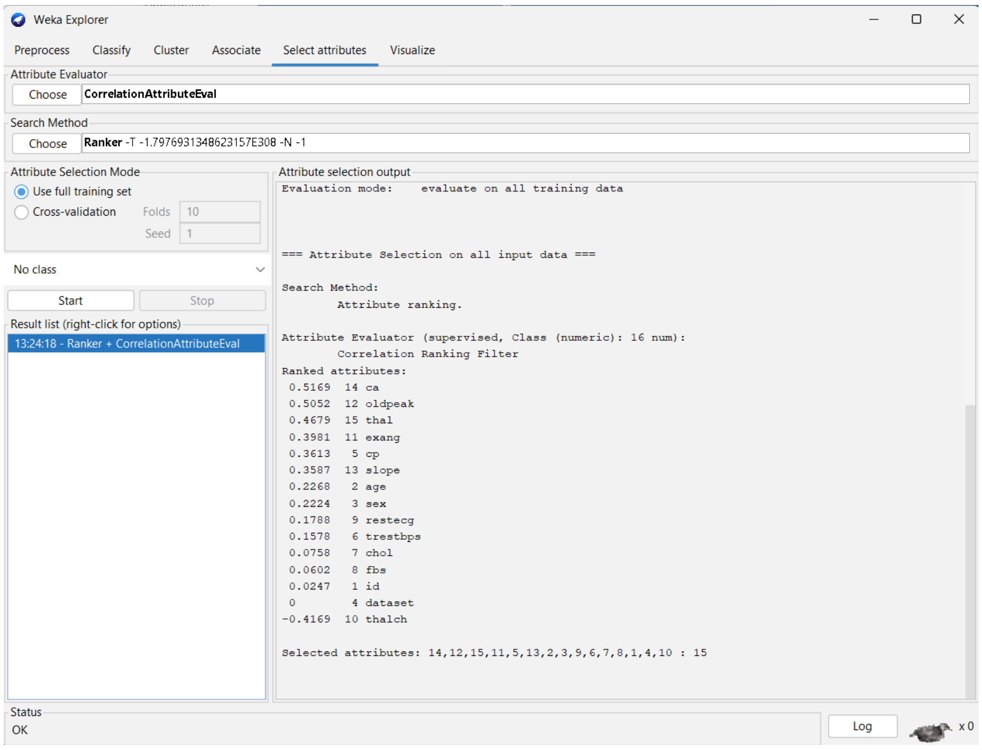
4.4.1. Correlation Attribute Evaluation

Using the correlation method, which is being presented here, it is possible to incorporate a reasonable number of necessary characteristics without affecting the accuracy of the model by employing an approach known as the correlation coefficient method. Weka is used for applying feature selection methods over the datasets.

Equation 1 shows the co-relation equation where X and Y show the corresponding attributes, calculating the mean at every step. Correlation measures the linear relationship between X and Y. Pearson correlation ranges from 1 (perfect negative correlation) to +1 (perfect positive correlation), with zero indicating no correlation between X and Y. A zero correlation does not indicate there is no linear relationship.

$$Z = \frac{\sum(X_i-\bar{X})\sum(Y_i-\bar{Y})}{\sqrt{\sum(X_i-\bar{X})^2\sum(Y_i-\bar{Y})^2}} \tag{1}$$

Weka 3.8.6 is used to implement correlation, which is attribute selection method and deployed over the data set 16 attributes as shown in *figure 4*. Results for the Heart Disease Data with correlation attribute evaluation are shown in *figure 4*.



Attribute Evaluator
Choose CorrelationAttributeEval

Search Method
Choose Ranker -T -1.7976931348623157E308 -N -1

Attribute Selection Mode
[✓ Use full training set]
Cross-validation Folds 10 Seed 1

No class

Result list (right-click for options)
13:24:18 - Ranker + CorrelationAttributeEval

Attribute selection output

```
Evaluation mode:    evaluate on all training data



=== Attribute Selection on all input data ===

Search Method:
        Attribute ranking.

Attribute Evaluator (supervised, Class (numeric): 16 num):
        Correlation Ranking Filter
Ranked attributes:
 0.5169  14 ca
 0.5052  12 oldpeak
 0.4679  15 thal
 0.3981  11 exang
 0.3613   5 cp
 0.3587  13 slope
 0.2268   2 age
 0.2224   3 sex
 0.1788   9 restecg
 0.1578   6 trestbps
 0.0758   7 chol
 0.0602   8 fbs
 0.0247   1 id
 0         4 dataset
-0.4169  10 thalch

Selected attributes: 14,12,15,11,5,13,2,3,9,6,7,8,1,4,10 : 15
```

Status
OK

Figure 4: Correlation attribute selector using WEKA

According to the findings of the correlation attribute evaluation from figure 4, attributes 1, 4, and 10 have the lowest correlation factor with the other attributes. The first attribute describes the id, which is just a serial number and is therefore less important in identifying the valuable information in the detection of heart problem. The fourth attribute describes the dataset that indicates the city, which is once again a very less important attribute related to the detection of patient health data. The tenth attribute which is "thalch" describes the detection method related to maximum heart rate of the patient, this value is additional because attribute 15th that is “thal” categories the patients in into different stages of categorization related to the ECG which makes it less important attribute. As a result of the correlation matrix study and the results that were obtained, these three attributes are removed from the test dataset as well as the train dataset. The optimized test and train datasets are now used over various classification processes to identify the best models and embed regression-based learning process to further optimize them.

4.4.2 Pre-classification Analysis Process

It is possible to do classification on both structured and unstructured data. Classification is the process of grouping together a certain set of data into distinct classes. The work carried out in [15,16,17] show the success of classification over the medical data.

In this step factors of offloading are defined to apply regression-based learning to offload the data.

4.4.2.1 Heart Disease Dataset Analysis

For the analysis and classification purpose, the complete implementation process is carried over the python 3.9 in the anaconda simulator, first, the reading and analysis of the train data set is carried out, for identifying the appropriate classification scheme.

The data set is categorised under the four categories or classes having values

df['num'].value_counts()   # Python Code

Class	Attribute Count	Specification
0	411	#Normal Patient Class data
1	265	#Low Risk Patient Class data
2	109	#Low to Medium Risk Patient Class data
3	107	#Medium to High Risk Patient Class data
4	28	#High Risk Patient Class data

Based on different classes of patient health data, it can be said that this data support multi-class classification, and also offloading scenarios can be decided based on the criticality of the patients as:

#0 Normal Patient Categorization-wise data can be offloaded to the cloud.
#1 Critical Patient Categorization-wise data must be offloaded to the fog only with a small
computing capacity.
#2 Critical Patient Categorization-wise data must be offloaded to the fog only with medium
computing capacity.
#3 Critical Patient Categorization-wise data must be offloaded to the fog only with a large
computing capacity.
#4 Critical Patient Categorization-wise data must be offloaded to the fog only with a large computing capacity.

4.4.3 Handling Missing or Null Values

After analyzing and categorization of data for offloading purposes, null or missing value analysis is being done as this is the issue under the classification and affect the accuracy of overall model. If there are no data on a particular variable for any of the cases, then it is referred to that variable as being latent or unobserved. On the other side, there is unit non-response when data are absent on all variables for some of the cases [18].

df.isna().sum() # python code to check null values shown in *table 2*.

Attribute-wise results in the training and test set are given in *table 2*, which shows that a large amount of data is not present in the dataset. As its clear from the result analysis of missing values that the data set contains missing values in both train and test sets, therefore handling missing values is required. Mean and median are two of the basic strategies to handle this issue, especially in the case of a dataset having a small size [19], Even with a big data set, the mean criterion calculation is relatively quick. Because of these qualities, the mean criterion is an effective method of selection, particularly in situations in which a large number of values are available and only random values are absent [20]. However, when dealing with unsupervised data, utilizing the mean approach for tuning is an incredibly challenging task; hence, the mean method is appropriate for use in the present experiment scenario when it comes to filling in missing values. Based on the study mean approach is selected as part of the proposed algorithm during the pre-processing stage, *table 2* presents the case of results before and after applying the mean approach to both data sets.

Table 2: Missing value analysis after applying mean method over both train and test data

Train Data Before Applying Mean Method		Train Data After Applying Mean Method		Test Data Before Applying Mean Method		Test Data After Applying Mean Method	
age	0	age	0	age	0	age	0
sex	0	sex	0	sex	0	sex	0
cp	0	cp	0	cp	0	cp	0
trestbps	0	trestbps	0	trestbps	59	trestbps	0
chol	0	chol	0	chol	3	chol	0
fbs	0	fbs	0	fbs	9	fbs	0
restecg	0	restecg	0	restecg	2	restecg	2

exang	0	exang	0	exang	55	exang	0
oldpeak	0	oldpeak	0	oldpeak	62	oldpeak	0
slope	1	slope	1	slope	38	slope	38
ca	5	ca	0	ca	66	ca	0
thal	3	thal	3	thal	483	thal	483
num	0	num	0	num	0	num	0

As *table 2* shows that still large amount of data remains missing after applying the mean method, this data belongs to the textual information. Conversion of textual information to numeric is done for attributes “slope” and “thal” during identifying the attributes *table 3* shows the information and assigned constant numerical values, under the experimentation setup this becomes the pre-step before applying the mean method to the train and test data set.

Table 3: Text attributes values and their replaced numerical values

Slope Values	Assigned Constant Value	thal Values	Assigned Constant Value
flat	0	**reversable defect**	0
upsloping	1	**Normal**	1
downsloping	2	**fixed defect**	2

After applying the mean method, there is still a requirement of changing some binary data attributes, which are present in the form of TRUE and FALSE like ‘sex’, ‘fasting blood sugar’, and “exang” table 4 presents the attributes and the assigned values, in proposed algorithm this step is in parallel with table 3 as text values are changed into constant values there.

Table 4: Binary data attributes conversion

fbs Value	Assigned Constant Value	sex Value	Assigned Constant Value	exang Value	Assigned Constant Value
TRUE	1	TRUE	1	TRUE	1
FALSE	0	FALSE	0	FALSE	0

4.5 Classification Analysis Process

An algorithm is proposed for the classification and category-wise offloading decision purpose, Algorithm 2 gives the detailed step-wise description:

```
#Algorithm 2(Classification and Categorization)
#Algorithm starts here:
#Splitting dataset into Training and Testing
1: Traing_Split, Testing_Split = split (heart_disease_attributes, heart_disease_label)
2: return Traing_Split, Testing_Split
# Building Different Classifiers
3: Model-1: Logistic_Regession_Model(Traing_Split, Traing_label, Testing_Split)
4: Model-2: k-NN_Model(Traing_Split, Traing_label, Testing_Split)
5: Model-3: (Decesion_Tree)CART_Model( Traing_Split, Traing_label, Testing_Split)
# Building Ensemble Model LR-k-NN-CART-SEMod
6: metaClassifier_Stacking = 'Decision_Tree'
7: Ensemble_Model(Traing_Split, Traing_label, Testing_Split)
8: metaClassifier_Stacking = co-ncatenate( Model-1, Model-2, Model-3)
#Pre-Caching (Parallel with stacking caching model wise rules generated)
9:(Least_Recently_Used)LRU_Caching(Model-1 Rules->Model-2 Rules->Model-3 Rules)
#Output received in multiclass mode due to five different categories of data
10:Ensemble_Model_Predictions = metaClassifier_Stacking.predict(Testing_Split)
# Category wise decision of offloading
# Multi class output (five classes data {0,1,2,3,4})
# Offloading Process
11: Patient #0 Category Normal Cloud data
12: Patient #1 Category only data with low computing capacity should be offloaded to the fog.
13: Patient # 2 Category critical patient with average chance of heart disease data must be offloaded to the fog with medium computing power.
14: Patient # 3 Category critical patient with moderate to high chance of heart disease data must be offloaded to the fog with large computing power.
15: Patient # 4 Category critical patient with high risk of heart disease data must be offloaded to the fog with large computing power.
#Algorithm end here
```

The offloading process shown in algorithm 2 clearly shows that to improve the overall performance is improved by using the concept of the pre-caching process, which affects the computational capability by optimizing the rules generation and further by reducing the dimension during offloading accuracy can be increased.

First, for the purpose of this research, several different classification algorithms were evaluated with one another while considering the primary evaluation metrics. The python version of 3.9 with an anaconda development environment is used, and all the experimental setup is carried out in jupyter notebook that supports all the python libraries. For the analysis purpose, the data set has been split into train and test sets with the percentage

of 75% as the training and 25% as the testing set, results have been tested and evaluated using a cross-validation procedure that is 10 times more extensive. The different classification algorithms tested are Logistic Regression, MLP Classifier, Decision Tree Classifier (CART), Support Vector Machine (SVM), KNeighborsClassifier (k-NN), and Random Forest, all of these implementations are based on python, below category-wise training and test data is split is shown:
Distribution of target variable in the training set:

0	329
1	212
2	87
3	86
4	22

Distribution of target variable in test set:

0	82
1	53
2	22
3	21
4	6

A multiclass analysis of the classification algorithms is done, as data indicate five different scenarios in classes related to patient health data, a confusion matrix is created the quantity of each indicator is computed with the help of this matrix, and then the results are compared. A confusion matrix is a useful tool for examining how well a classifier is able to detect many classes that are distinct from one another.
Formulas are used to determine several evaluation metrics, like the true positive rate (recall or sensitivity), the false positive rate (FPR), the false-negative rate (FNR), and the precision and accuracy of evaluation categories shown in equations (2 -to- 5), and confusion matrix is generated for different categories of the predicted values.

$$Accuracy = \frac{t_p + t_n}{t_n} + f_p + f_n + t_n \qquad (2)$$

Accuracy involves identifying instances accurately using various learning methods. Equation 8 shows, t_p= true positive values, t_n= true negative values, f_p= false positive values, f_n= false negative values.

$$Precision = \frac{t_p}{t_p + f_p} \qquad (3)$$

Precession is accurately detected occurrences over total instances. Precision implies fewer false-positives.

$$Recall = \frac{TP}{TP + FN} \qquad (4)$$

The recall is the percentage of class occurrences successfully anticipated.

$$Score = 2 * \frac{Recall * Precision}{Recall + Precision} \qquad (5)$$

It's the average of accuracy and recall, thus it includes false positives and negatives, and is known as F-measure.

A multi-dimensional confusion matrix is built that is used to describe the performance of the multiple classifiers that are created over the experiments here from the dataset.

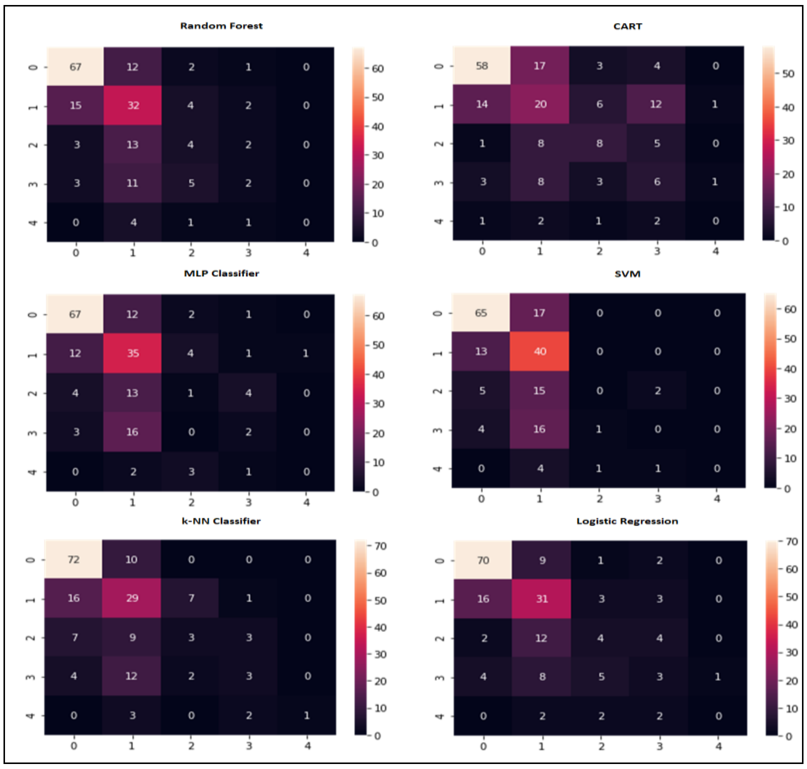


Figure 5: Presenting the matrix representation of different classifiers implementation

Figure 5 shows the resultant matrix of different classifiers with five different classes, table 5 shows that classifiers can identify the patient's health in different classes, and out of the accuracy achieved from these classifiers' models, a maximum of 60% accuracy is achieved.

Table 5: Accuracy percentage of different classifiers

Model	Accuracy	Precision	Recall	F1 Score
Random Forest	60.32%	0.38	0.37	0.37
MLP	51%	0.34	0.33	0.33
k-NN	58.69%	0.55	0.37	0.39
CART	50.54%	0.34	0.33	0.33
SVM	57.06%	0.24	0.31	0.26
LOGISTIC Regression	59.78%	0.36	0.36	0.35

Stacking is a set of techniques that requires training a second-level "metalearner" to determine the optimal combination of the base learners. It is also known as Super Learning or Stacked Regression. The purpose of stacking is to bring together formidable and varied groups of learning algorithms to work on the same problem as shown in algorithm 3.

```
#Algorithm 3 (Stacking Algorithm)
Step 1: Assemble the group.
    I.   List L-base algorithms (with a specific set of model parameters).
    II.  Specify a meta-learning algorithm.
Step 2: Train the group.
    I.   Train L-base algorithms on the training set.
    II.  Perform a split of test and train set on each learner and collect the L algorithms' cross-validated predicted values.
    III. N cross-validated projected values from each of the "L" methods can generate a new "N x L" matrix. This matrix and response vector are called "level-one" data. (N=training set rows).
    IV.  Train meta-learning on level-1 data. The "ensemble model" combines L-based learning models and the meta-learning model to generate test set predictions.
Step 3: Predict new data.
    I.   To create ensemble predictions, start with basic learners.
    II.  Feed those predictions into meta learner to generate an ensemble forecast.
```

From the study of *table 5* and the matrix generated in *figure 5*, three models are selected for stacking and building the proposed hybrid regression-based learning process to further optimize them, Logistic regression is selected because of its average accuracy under the classification of 60% as well as its average performance over all classes, the approximate time taken for execution 0.6336061954498291seconds, with the memory utilization of 2.1649879196623365 MB, and 0.000244 joules energy consumption, its selected with k-NN classifier as from *figure 6* we can see that this model works good for normal patients as well as class 3 and class 4 patients, in case of k-NN 0.6023867130279541seconds is time consumption, with the 2.1305145227729505 MB memory utilization having 0.000214 joules energy consumption and last CART as it has maximum performance for the detection of class 2 and class 3 patients, whereas the approximate time consumption by the model is 0.5623865127563477 seconds, 2.3044440407899387 MB memory utilization with 0.000916 Joules energy consumption. The category-wise improvement in performance is an important criterion of the experiment as it is necessary to take the offloading decision. These results are obtained by running the same model 20 times and taking the average for computation time, memory consumption and energy utilization.

```
Resultant Confusion Matrix:
              0  1  2  3  4
 Class 0: [[77 3  2  0  0]
 Class 1: [ 8  42 1  2  0]
 Class 2: [ 2  8  11 1  0]
 Class 3: [ 2  6  2  10 1]
 Class 4: [ 0  1  1  2   2]]

Detailed Classification Report:
            Precision   Recall   f1-score   Support
      0       0.93       0.94     0.95       82
      1       0.79       0.79     0.84       53
      2       0.50       0.50     0.46       22
      3       0.47       0.47     0.32       21
      4       0.33       0.33     0.20       6

   Accuracy                       0.77        184

Accuracy: 0.771739130435
```

Table 6 shows the comparison of the proposed algorithm with the other learning classifiers with overall average precision recall and F1 score achieved.

Table 6: Accuracy percentage comparison with different classifiers

Model	Accuracy	Precision	Recall	F1 Score
Proposed Regression learning with k-NN and Decision Tree Algorithm	77.17%	0.741212	0.814959	0.728214
Random Forest	60.32%	0.38	0.37	0.37
MLP	51%	0.34	0.33	0.33
k-NN	58.69%	0.55	0.37	0.39
CART	50.54%	0.34	0.33	0.33
SVM	57.06%	0.24	0.31	0.26
LOGISTIC Regression	59.78%	0.36	0.36	0.35

The proposed regression-based hybrid algorithm improves the accuracy to 77% by correctly classifying the data in different categories for taking the offloading decision with 0.93 precision to send the data to the cloud directly for normal patients having class 0, 0.79 to small size machine where category class is 1 depicting fewer chances of heart health issue, whereas category class 2 patients that have moderate chances of disease will be offloaded to the medium size EC2 machine with the precision of 0.50, whereas category class 3 and 4 data where the risk of patients health is medium to high and high is decided to offloaded to the large size EC2 machine, as it indicates the critical data and also category class 3 and 4 data have precision less than 0.50, and it can be seen referring confusion matrix and detailed classification report of the proposed algorithm, that category 4 data is misclassified into category 3. The decision of a common offloading machine boosts the precision value and thus accuracy by achieving a combined performance for class 3 and 4 patients.

The different performance parameters of the proposed model is given below:

- Approximate average computation time consumption 1.43442702293396 seconds
- Approximate average memory utilization 2.9475242632165926 MB
- Approximate average energy consumption 0.002587 joules

The results are obtained by running every experiment 20 times on the same machine and taking the average of them. From the results it can be clearly seen that due to hybrid taking three learning schemes together there is increase in computation time, memory utilization and energy consumption.

Resultant Confusion Matrix with respect to offloading decision:

```
             0   1   2   3
Class 0: [77 3 2   0] # Offloaded to the cloud
Class 1: [8   42 1 2] # Offloaded to the small size EC2 machine
Class 2: [2    8    11 1] # Offloaded to the medium size EC2 machine
Class 3: [2   7   3   15] # Offloaded to the large size EC2 machine
```

Combining results of category class 4 from both row and column-wise, also class 1 and class 2 wrongly categorized data in class 3 will be offloaded to large size machine which will be considered as correctly categorized only as large size machine is having larger computation power and further boosting the performance in terms of accuracy and finally considered achieved matrix will be following

Resultant Confusion Matrix with respect to offloading decision on reduced dimension size:

```
             0      1    2     3
Class 0: [77     3     2     0] # Offloaded to the cloud
Class 1: [8     42    1     0] # Offloaded to the small size EC2 machine
Class 2: [2      8     11     0] # Offloaded to the medium size EC2 machine
Class 3: [2    7      3    18] # Offloaded to the large size EC2 machine

accuracy                       0.80      184
```

Accuracy: 0.8043478261

The offloading process decision model added with the proposed algorithm raised the accuracy to 80% by reducing the one dimension from the multi-dimension resultant matrix.

4.6 Pre-Caching Process

Next, to look at the improvements in response time and energy that were made possible in the proposed gateway design, caching concept is applied with staking to further reduce the amount of energy consumed and the reaction time. The evaluation is carried out by using the pre-caching concept to store the results of the classifier during the stacking and used in the next classifier, then offloading them, and then assessing the reaction time and energy consumption associated with each of these processes.

A Least Recently Used (LRU) caching strategy is used as part of the implementation, caching, when implemented properly, speeds up work and minimizes computational load by using less number of resources.

Steps to implement LRU:
1. Set the LRU cache capacity to a positive number.
2. toString(int key) Otherwise, return -1 if the key does not exist.
3. put(key, value): If the key exists, update the value of the key. Add the key-value pair to the cache if you don't need it.
4. The least recently used key should be removed if the number of keys exceeds the storage capacity.

The different performance parameters of the proposed model after implementation the caching are given below

Approximate average computation time consumption 0.7740482482452393 seconds
Approximate average memory utilization 3.8396332733711076 MB
Approximate average energy consumption 0.002153joules

The results are obtained again by running every experiment 20 times on the same machine and taking the average of them. From the results it can be clearly seen that due to hybrid taking three learning schemes together there is increase in computation time, memory utilization and energy consumption. The results shows with pre-caching computation time and energy consumption has been reduced when compared to the results without caching process, whereas memory utilization is increased around 1 MB.

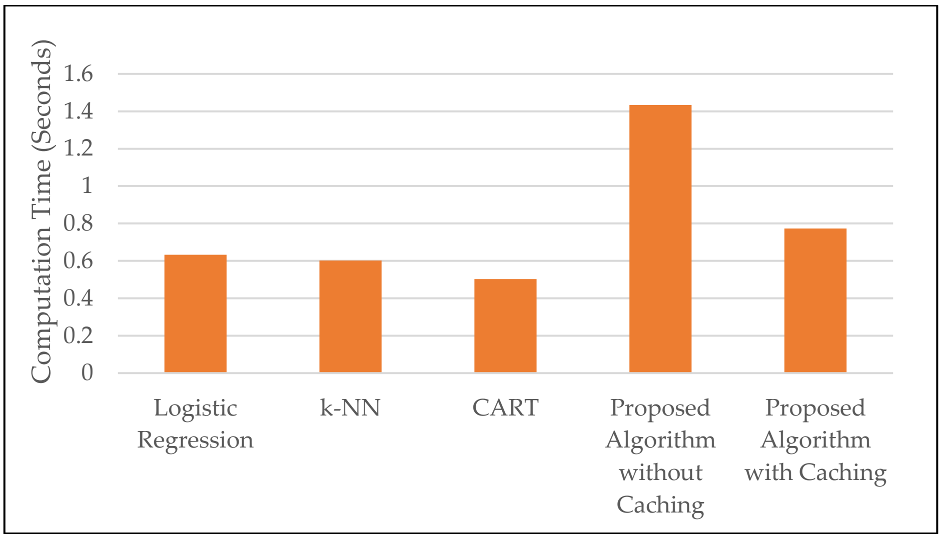


Figure 6: Time Consumption of the proposed algorithm

Figure 6 shows the computation time taken by the proposed algorithm with pre-caching when compared to other selected classifiers, figure clearly shows the average consumption time is approximately 0.774 seconds which is much more than the other compared learning schemes but individually no algorithm is able to correctly classify the data with more than 60% accuracy whereas proposed algorithm produces the results with around 80% of accuracy.

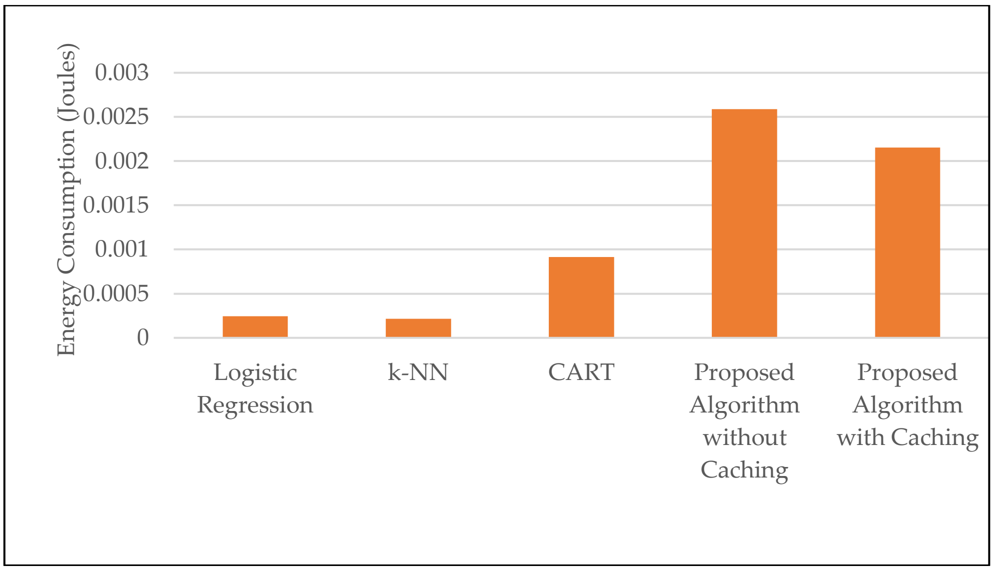


Figure 7: Energy Consumption of the proposed algorithm

Figure 7 shows the energy consumption by the proposed algorithm with pre-caching when compared to other selected classifiers, figure clearly shows the average energy consumption is while computation is 0.002153 joules much more than other implemented classifier algorithms but again in terms of accuracy the proposed algorithm is more precise to correctly classify the patient's data. The data from the healthcare IoT devices are separated into normal, average, and high-risk patients. Based on this classification, a choice on offloading is then made, as detailed in algorithm 3, which serves as a strategy for the allocation and selection of data packets in fog nodes.

5. RESULT ANALYSIS AND DISCUSSION

In this section a detailed analysis of different parameters of frameworks is done in the offloading scenario, depth calculations are performed to do the analysis.

Two different offloading methodologies are deployed using python language and compared with the proposed model implemented, accuracy, time, memory usage and energy consumption comparisons are made for the time start and end time calculations are done, for calculating the memory utilization "psutil" a cross platform library of python for retrieving the memory utilization are used. Energy calculations are done with the help of "pyJoules.energy_meter" a python library to measure energy consumption of python code. An approximate average calculation is done by running the same experiment 20 times and taking 20 different readings and then calculating the average.

5.1 FCFS Approach

A First Come First Serve (FCFS) approach is first adopted to send the data to the multiple machines (Cloud, SM, ML and XL) of different sizes. "Dask" a free open source python library is used to achieve the parallel computing, to allocate the tasks to the respective machines,

In terms of accuracy obtained data is presents

False	735
True	185

Name: Category

It means out of 920 value only 185 data were correctly classified and was directed to the right machine for the computation. Which is equivalent to only 20.10% data accurately classified which is to less and can't adopted for the critical data of patients.

5.2 Batch Processing

A batch processing approach is first adopted to send the data to the multiple machines (Cloud, SM, ML and XL) of different sizes. A batch of 50 entries are made using loop to send the data to each machine including cloud. Again "Dask" a free open source python library is used to achieve the parallel computing, to allocate the tasks to the respective machines.

In terms of accuracy obtained data is presents

False	719
True	201

Name: Category

It means out of 920 value only 201 data were correctly classified and was directed to the right machine for the computation. Which is equivalent to only 21.84% data accurately classified which is to less and can't adopted for the critical data of patients.

The proposed algorithm is compared with the other scheduling algorithms, that are FCFS and Batch Processing.

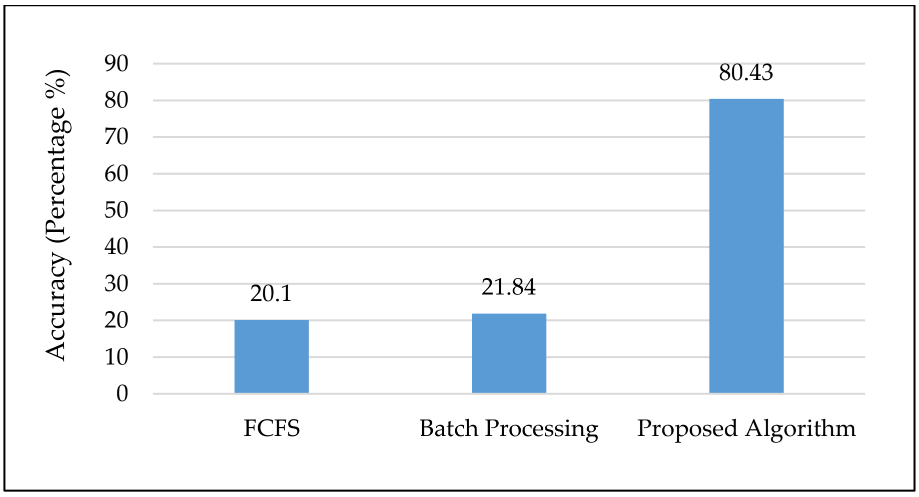


Figure 8(a): Accuracy of proposed algorithm in terms of percentage

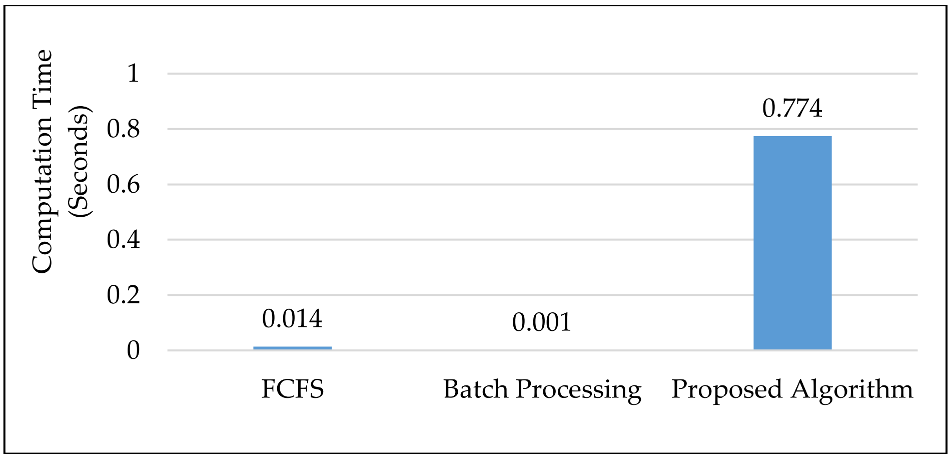


Figure 8 (b): Average computing time in seconds of the proposed algorithm

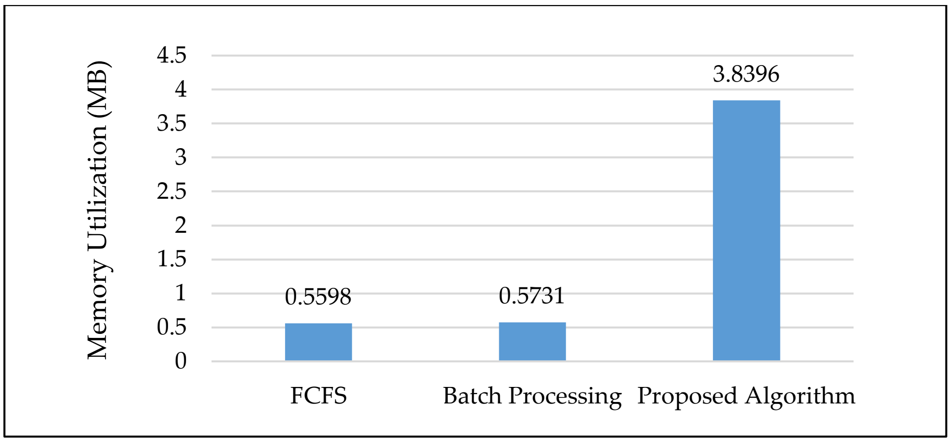


Figure 9(a): Memory utilization in Mega Bytes (MB) of the proposed algorithm

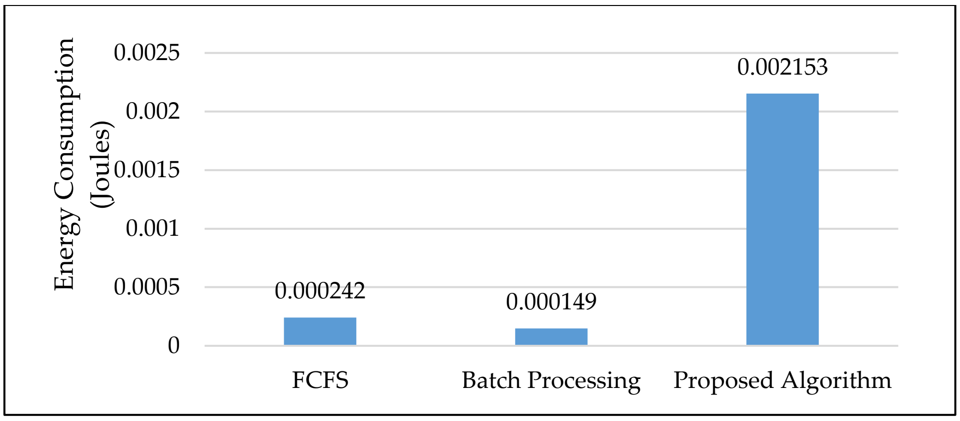


Figure 9(b): Average energy consumption of the proposed algorithm

Figure 8 and 9 clearly shows that the FCFS and Batch processing is more efficient in terms of time, memory and energy then the proposed algorithm, but there is mostly incorrectly classified data which leads to accuracy less than 25% which is not acceptable in case of critical data such as medical and health like presented scenario, whereas proposed algorithm and scheme leads to accuracy around 80% to process the data. For FCFS and Batch processing scenario of 4 classes for offloading is considered that makes the offloading as multiclass. The time, energy and memory calculations are the computation calculation that system takes (smart gateway) for taking the offloading decision for the appropriate machine or the cloud.

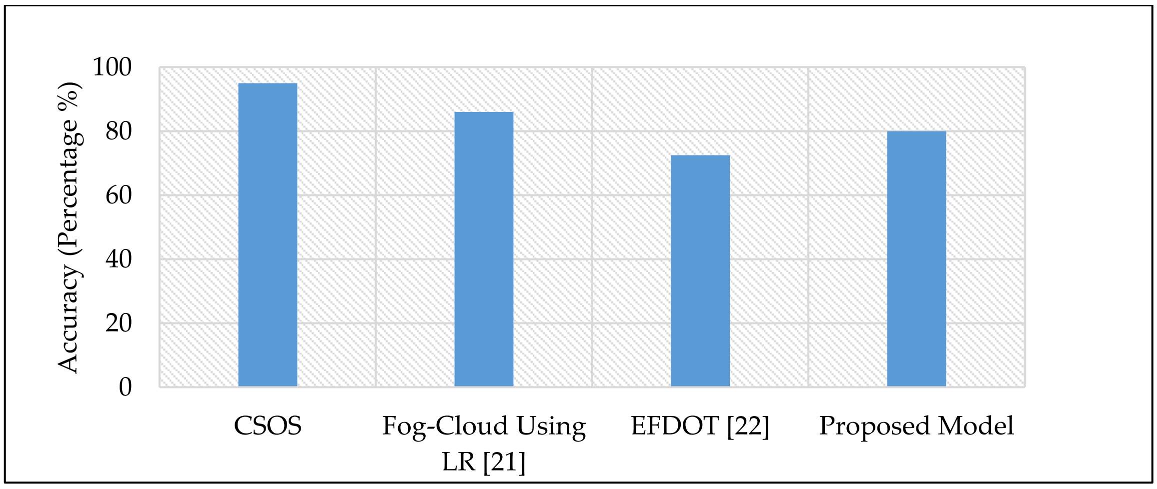


Figure 10: Accuracy comparison of proposed model

Most of the models and implemented frameworks haven't considered accuracy as the prime parameter in case of offloading as all the data will be getting processed at some stage or time, although the some of the main architectures that has considered accuracy are compared in fig 10. Although the proposed model is presenting the accuracy as less then CSOS and Fog-Cloud but representing the accuracy achieved over multiclass classification whereas other models simply work on binary classification where accuracy is more and in the scenarios like healthcare sectors patient data is critical and multiclass classification is more preferred and important.

6. CONCLUSION

Due to performance indicators such as latency, energy consumption, and response time exceeding threshold levels, it is difficult to run delay-sensitive applications and the cloud at the same time. This is true even when advanced networks and technologies are deployed. The Internet of Things (IoT) architecture's middleware layer appears to be a promising solution that might be used to address these challenges while still meeting the need for high task offloading criterion. A Context-Aware Offloading model is designed and developed as a proposed solution that makes offloading decisions in multiple fog-based cloud environments by using machine-learning reasoning techniques using hybrid logistic regression learning. Two algorithms are implemented for feature selection and classification techniques used to evaluate the model, time and energy consumption of the model is calculated with memory utilization. Although when compared with FCFS and Batch processing, proposed model consumes more memory, energy and time but due to contextual reading its accuracy in decision taking for offloading is more. When compared with other peer models, model that has been proposed provides a solution that saves both time and energy while maintaining an accuracy level that is approximately 80 percent. It is anticipated that Internet of Things applications would be able to meet the criteria for a low response time as well as other performance characteristics if the intelligent offloading approach that is context-based offloading that has been described is implemented. In future a framework will be proposed implementing the model and evaluated over different parameters like total computation time including the round-trip time and total energy consumption for the framework including the communication requirements.

Conflicts of Interest

Declare conflicts of interest or state "The authors declare no conflict of interest." "The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results"

REFERENCES

[1] Sethi, P. and Sarangi, S.R., 2017. Internet of things: architectures, protocols, and applications. Journal of Electrical and Computer Engineering, 2017. vol. 20, pp. 1–25. doi:10.1155/2017/9324035

[2] Li, Y., Björck, F., &Xue, H. Iot architecture enabling dynamic security policies. In Proceedings of the 4th International Conference on Information and Network Security, 2016 (pp. 50-54). ACM. https://doi.org/10.1145/3026724.3026736

[3] Li Y, Björck F, Xue H. IoT Architecture Enabling Dynamic Security Policies. In: Proceedings of the 4th International Conference on Information and Network Security [Internet]. New York, NY, USA: Association for Computing Machinery; 2016. pp. 50–4. (ICINS '16). https://doi.org/10.1145/3026724.3026736

[4] Bukhari, M. M., Ghazal, T. M., Abbas, S., Khan, M. A., Farooq, U., Wahbah, H., Ahmad, M., & Adnan, K. M. An Intelligent Proposed Model for Task Offloading in Fog-Cloud Collaboration Using Logistics Regression. Computational Intelligence and Neuroscience, 2022, 3606068. https://doi.org/10.1155/2022/3606068

[5] Poonam and Suman Sangwan (2022), Task Scheduling on Cloudlet in Mobile Cloud Computing with Load Balancing. IJEER 10(4), 994-998. DOI: 10.37391/IJEER.100440.

[6] Kosta, S., Aucinas, A., Pan Hui, Mortier, R., & Xinwen Zhang. ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. 2012 Proceedings IEEE INFOCOM, pp. 945–953. https://doi.org/10.1109/INFCOM.2012.6195845

[7] Ting-Yi Lin, Ting-An Lin, Cheng-Hsin Hsu, & Chung-Ta King. Context-aware decision engine for mobile cloud offloading. 2013 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), pp. 111–116. https://doi.org/10.1109/WCNCW.2013.6533324

[8] Nakahara, F. A., & Beder, D. M. A context-aware and self-adaptive offloading decision support model for mobile cloud computing system. In Journal of Ambient Intelligence and Humanized Computing. 2018, (Vol. 9, Issue 5, pp. 1561–1572). https://doi.org/10.1007/s12652-018-0790-7

[9] Kim, H.W., Park, J.H. and Jeong, Y.S., Adaptive job allocation scheduler based on usage pattern for computing offloading of IoT. Future Generation Computer Systems, 2019, Vol. 98, pp.18-24.

[10] Junior, W., Oliveira, E., Santos, A. and Dias, K., A context-sensitive offloading system using machine-learning classification algorithms for mobile cloud environment. Future Generation Computer Systems, 2019, 90, pp.503-520.

[11] Shukla, S., Hassan, M. F., Khan, M. K., Jung, L. T., & Awang, A. An analytical model to minimize the latency in healthcare internet-of-things in fog computing environment. PloS One, 2019, 14(11), e0224934. https://doi.org/10.1371/journal.pone.0224934

[12] Beloglazov, A., Abawajy, J., & Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. Future Generations Computer Systems: FGCS, 28(5), 2012, pp. 755–768. https://doi.org/10.1016/j.future.2011.04.017

[13] Benedetto, J.I., González, L.A., Sanabria, P., Neyem, A. and Navón, J., Towards a practical framework for code offloading in the Internet of Things. Future Generation Computer Systems, 2019, 92, pp.424-437.

[14] Andras Janosi WS, Matthias Pfisterer, Robert Detrano. UCI Machine Learning Repository 2018 (assessed on 03 Jan 2022). https://archive.ics.uci.edu/ml/datasets/heart+Disease

[15] Rahmani, A. M., Gia, T. N., Negash, B., Anzanpour, A., Azimi, I., Jiang, M., & Liljeberg, P. Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: A fog computing approach. Future Generations Computer Systems: FGCS, 78, 2018, pp. 641–658. https://doi.org/10.1016/j.future.2017.02.014

[16] Gállego, J. R., Hernández-Solana, A., Canales, M., Lafuente, J., Valdovinos, A., & Fernández-Navajas, J. Performance analysis of multiplexed medical data transmission for mobile emergency care over the UMTS channel. IEEE Transactions on Information Technology in Biomedicine: A Publication of the IEEE Engineering in Medicine and Biology Society, 2005, 9(1), pp. 13–22. https://doi.org/10.1109/titb.2004.838362

[17] Alarsan, F. I., & Younes, M. Analysis and classification of heart diseases using heartbeat features and machine learning algorithms. Journal of Big Data, 2019, 6(1), pp. 1–15. https://doi.org/10.1186/s40537-019-0244-x

[18] Wang, W., & Carreira-Perpinan. The role of dimensionality reduction in classification. In Proceedings of the AAAI Conference on Artificial Intelligence 2014, (Vol. 28, No. 1).

[19] Fira, M., Costin, H.-N., & Goraș, L. On the Classification of ECG and EEG Signals with Various Degrees of Dimensionality Reduction. Biosensors,2021, 11(5). https://doi.org/10.3390/bios11050161

[20] Chaudhuri, A., Kakde, D., Sadek, C., Gonzalez, L., & Kong, S. The mean and median criteria for kernel bandwidth selection for support vector data description. In 2017 IEEE International Conference on Data Mining Workshops (ICDMW) 2017, (pp. 842-849). IEEE.

[21] Bukhari, M. M., Ghazal, T. M., Abbas, S., Khan, M. A., Farooq, U., Wahbah, H., Ahmad, M., & Adnan, K. M. An Intelligent Proposed Model for Task Offloading in Fog-Cloud Collaboration Using Logistics Regression. Computational Intelligence and Neuroscience, 2022, 3606068. https://doi.org/10.1155/2022/3606068

[22] Ali, Z., Abbas, Z. H., Abbas, G., Numani, A., & Bilal, M. Smart computational offloading for mobile edge computing in next-generation Internet of Things networks. Computer Networks, 2021, 198, 108356. https://doi.org/10.1016/j.comnet.2021.108356