**International Journal of Data Informatics and Intelligent Computing (IJDIIC)**
Vol.2, No.2, June 2023, pp. 1~10
DOI: 10.59461/ijdiic.v2i2.57

# Intepretable Deep Gaussian Naive Bayes Algorithm (Idgnba) Based Task Offloading Framework for Edge-Cloud Computing

**Prabhdeep Singh**[1]
[1]School of Computer Applications, BBD University, Lucknow, Uttar Pradesh, India

**Article Info**

***Article history:***

Received May 18, 2023
Revised June 14, 2023
Accepted June 16, 2023

***Keywords:***

Internet of Things (IoT)
Edge computing
Task offloading
Interpretable deep Gaussian naive Bayes approach (IDGNBA)

**ABSTRACT**

When it comes to Internet of Things (IoT) applications and machine learning based computing, resource-restricted edge devices are inadequate due to the exponential growth of mobile information and the massive need for processing power. An edge offload, the migration of complex tasks from IoT devices to edge cloud servers, is a distributed computing paradigm that has the potential to overcome the IoT device resource limits, lessen the computational load, and increase the effectiveness with which activities are processed. However, due to the NP-hard nature of the optimum offloading decision-making issue, an efficient solution using traditional optimization techniques is difficult. Current deep learning algorithms still have a lot of problems, such as their slow pace of learning and limited ability to adapt to new environments. We provide a unique interpretable deep Gaussian naive Bayes technique (IDGNBA) for extremely fine offloading choices to address these issues. Through several simulation studies, we assess the efficacy of IDGNBA and find that it performs better in terms of offloading than traditional techniques. The model has strong mobility and can quickly adjust to a fresh MEC working atmosphere while taking offloading decisions in real-time.



***Corresponding Author:***

Prabhdeep Singh
School of Computer Applications
BBD University
Lucknow, Uttar Pradesh
India
Email: prabhdeepcs@gmail.com

## 1. INTRODUCTION

Due to its high cost-effectiveness and flexibility, which are made possible by consolidation—the centralization of processing, storage, and network management—cloud computing has seen significant development and application over the last several decades [1]. The last ten years have seen tremendous advancements in cloud computing, which have provided a wide range of advantages. Offloading computation is a feature of cloud computing that enables customers to move work that is computationally expensive to a faraway cloud that has a greater supply of resources [2]. The term "cloud computing" refers to a distant data center that is made up of a collection of supercomputers that collaborate and share resources to provide intensive resource computation and are managed by intelligent programs that include a software-defined network (SDN). Cloud computing is used by gadgets that have been unable to finish their computing work locally. These devices send their computing chores to the cloud. However, since there is a considerable distance among the cloud and the final devices, the network may have connection delays. This means that it is not suited for dependent on latency real-time applications. This is additionally to the backlog of traffic, which may cause the network to become overloaded [3].

Due to the poor connection and the great distance between them, offloading these jobs to faraway cloud servers causes a significant delay in the task offloading process. Using edge computing as a means to provide computer services at the peripheries of a network is one potential approach to the problem. Nevertheless, computing at the edge is inadequate and inefficient on its own to handle the spatially-temporally variable need for processing power. It is necessary to install a vast number of edge servers in fixed locations to provide complete geographical coverage with dependable quality provisioning. This will always result in substantial deployment costs, high operating expenditures, and an enormous amount of wasted resources [4]. These applications often produce a significant quantity of data and are frequently dependent on delays.

It is common practice for such services to be prioritized for computation at the edge of the infrastructure compared to in a distant cloud owing to the stringent latency requirements and high computation needs. As a result, edge computing and small molecules are compatible entities that have the potential to cooperate. A competitive approach for the offloading of mobile tasks is the use of small cells that are equipped with edge servers. Because these servers are located very close to the beginning of the tasks, they are better able to satisfy the stringent latency requirements [5]. The massive file upload and preliminary processing of offloading duties, and download processes are both reduced to a minimum by the design of edge computing, which adds to a shorter amount of time needed for the total service. However, there are significant issues involved in effectively managing the computational workloads of applications that are sensitive to latency and edge-cloud resource utilization [6,7]. Interpretable Deep Gaussian Naive Bayes (IDGNB) is a machine learning algorithm that can be used as a classifier in a system for job offload in edge-cloud computing. The IDGNB algorithm combines the interpretability of the Gaussian Naive Bayes (GNB) algorithm with the expressiveness of a deep neural network (DNN). The IDGNB algorithm is a promising machine learning algorithm for use in a System for job offload in edge-cloud computing. It combines the interpretability of the GNB algorithm with the expressiveness of a DNN, making it well-suited for handling high-dimensional feature spaces and providing insights into the decision-making process of the classifier.

The rest of the paper is as follows related works presents in section 2, the method describes in section 3, section 4 gives results and discussion, and section 5 depicts the conclusion of the paper.

## 2. RELATED WORKS

In the study [8], a novel compression as well, as security, and resources-aware task offloading architecture is suggested for the edge-cloud computing (ECC) system architecture to get over the constrained bandwidth and solve the possible security threats concern. To be more precise, we first add a layer of effective compression to shrewdly lessen the amount of data being sent across the channel. The goal of the research is to tackle the complicated issue that has been posed by suggested a low-complexity and dispersed offload framework. The framework is based on specific network limitations [9]. For an on-demand edge-cloud system for computing with several users and UAVs, the article developed a powerful resource distribution and offloading of computations methodology. The suggested solution is expandable and capable of handling rises in traffic on the network without deteriorating performance [10]. In the study [11], an IoT-Edge-Cloud computing scheme that uses blockchain with advantages from both mobile-edge computing (MEC) and mobile cloud computing (MCC) servers is proposed. MEC servers provide reduced latency services for computing, while MCC servers have more processing capability. In the paper [12], an innovative task offloading algorithm known as meta reassurance-deep positive reinforcement trying to learn-based offloading was proposed. It consists of two models: a meta-RL (meta-reinforcement learning) model that enhances the model's ability to migrate, and a DRL (Deep Reinforcement Learning) model that combines several level DNNs (Deep Neural Networks) to learn from previous task offloading situations. The paper [13] suggested the DCC task offloading structure, which has three layers: the device layer, the cloudlet layer, and the cloud layer. In DCC, the smallest area layer and cloud layer are used to offload jobs with high computational requirements. To learn the combined task offloading and resource allocation choice while lowering training costs and limiting privacy leakage during DRL training, a federal deep reinforcement learning (FDRL) architecture is suggested [14]. In a multi-user context, the article developed a delay-optimal task offloading strategy for multi-tier edge-cloud computing. To reduce the overall service time of UAVs (Unmanned Aerial Vehicles), the issue is framed as a model for optimization utilizing Integral Linear Programming (ILP) methods.

## 3. METHOD

In this section, we discuss in detail about Interpretable Deep Gaussian Naive Bayes Algorithm (IDGNBA). To improve performance and reduce latency, a task offloading paradigm for peripheral-cloud computing comprises the effective allocation of computing jobs between the edge equipment and cloud servers. The framework involves a set of techniques, protocols, and algorithms that enable seamless and optimal task offloading. The speed and effectiveness of edge-cloud systems for computing may be

considerably increased with a well-designed job-offloading architecture, enabling the deployment of more complex and resource-intensive applications on edge devices. Task offloading is the process of distributing computational tasks between edge devices and cloud servers, to maximize performance and minimize latency. By offloading tasks to the cloud server, edge devices can conserve their limited resources and execute other local tasks, resulting in increased efficiency and reduced energy consumption.

In this context, a task-offloading framework for edge-cloud computing is essential for the efficient distribution of computational tasks between edge devices and cloud servers. The framework involves a set of techniques, protocols, and algorithms that enable seamless and optimal task offloading. The suggested architecture for offloading tasks for edge-cloud processing seeks to solve the difficulties of performing computation-intensive applications on edge devices by leveraging the computing resources available on cloud servers. The framework considers factors such as task partitioning, offloading decision-making, resource allocation, task scheduling, data management, and quality of service management.

### 3.1. Problem Formulation and system model

We provide a summary of the system's model in this part, followed by definitions of the time delay models and the consumption of energy model. The optimization problem of compute offloading is then defined on this foundation.

#### 3.1.1. System Model

The schematic representation of a system model for IoT-edge-cloud computing systems' work offloading mechanism may be seen in Figure 1. The suggested architecture is made up of a server in the cloud an outside server, and several (IoT) devices. The process of the (IOT) devices may be completed locally or offloaded to a server located in the cloud or edge server.

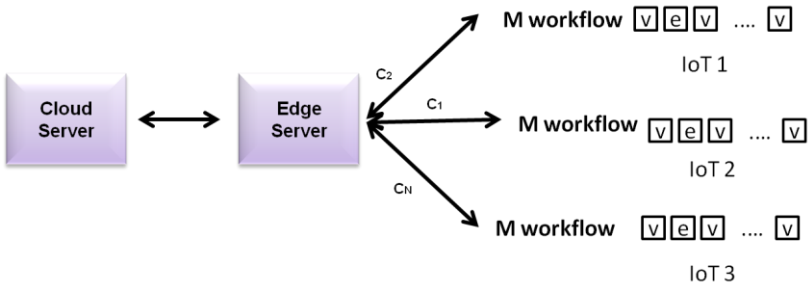


Figure 1. A system design for IoT-edge-cloud computing systems' work offloading mechanism

Within this architecture, edge cloud servers are located at close range to the various devices and provide high bandwidth. The device sends workflow information to the edge server, which then uses that information to make judgments about fine-grained unloading. The software for every device may be broken down into a series of processes in order of sequence. We will proceed on the assumption that the xth process is described as follows:

$$K_y = \{a_{0,}1, c_{1,}l_2, 2, c_2, \dots, w_j, a_{j,i,}c_i, \dots, l_{m-1,m,}a_{m-1}, m, c_{m,}a_m + 1\} \quad (1)$$

where $c_i$ is the i-th job in the process and $a_{j,i}$is an illustration of the collection of information flows through tasks $c_i$ and $c_j$. where $c_i$ signifies the j-th activity in the procedure.

Each process x can choose whether or not to unload its corresponding task $c_i$, and the unloading choice is represented by a variable:

$$p_{y,j} \in (p_0, p_1, p_2), \quad (2)$$

where a0 = [1 0 0]K denotes the choosing to work x will perform its j-th job local, a1 = [0 1 0]K denotes the choice that workflow x will offload its i-th work to the edge cloud server, and a2 = [0 0 1]K denotes the choosing to work x will offload its j-th job to the cloud servers.

### 3.1.2. Delay Model

Both the calculation delay and the process of transmission delay are included in the delay that is produced by computation offloading. We do not take into account the delay that occurs as a result of outsourcing decision-making since the amount of time needed to make the choice is quite minimal. As a result, the amount of time needed to complete job $c_j$ may be computed as follows:

$$D_j^v = \begin{cases} \frac{c_j}{V_0}, p_{y,j} = p_0, \\ \frac{c_j}{V_1}, p_{y,j} = p_1, \\ \frac{c_j}{V_2}, p_{y,j} = p_2, \end{cases} \quad (3)$$

where V0, V1, and V2 correspond, respectively, to the computational capacity of the Internet of Things, the computational both processing capacity of the cloud servers, and the edge server.
The following is the between task $c_i$ and $c_j$, there is a transfer lag:

$$D_{j,i}^v = \begin{cases} 0, p_{y,j=}p_{y,i} \\ \frac{a_{j,i}}{P_{0,1}}, p_{y,j=}p_0 \, , p_{y,i} = p_1 or \; p_{y,j=}p_1 \, , p_{y,i} = p_{0,} \\ \frac{a_{j,i}}{P_{1,2}}, p_{y,j=}p_1 \, , p_{y,i} = p_2 or \; p_{y,j=}p_2 \, , p_{y,i} = p_{1,} \\ \frac{a_{j,i}}{P_{0,2}}, p_{y,j=}p_0 \, , p_{y,i} = p_2 or \; p_{y,j=}p_2 \, , p_{y,i} = p_{0,} \end{cases} \quad (4)$$

where B0,1 represents the bandwidth that has been allotted to the IoT device's connection to the edge cloud server. The connectivity allotted for communication among the cloud servers and the edge servers is denoted by B1,2. Similarly, the bandwidth that an IoT device is given a server in the cloud is denoted by the notation B0,2.

The overall lag time for workflow x may be determined as follows:
$D_y = \sum_{j=1}^{M}(D_j^c + D_{j,j+1}^d)$, (5)
in which the process x is coupled with N different jobs.

### 3.1.3. Energy Consumption Model

One possible way to represent the electrical energy usage model of procedure x is as follows:
$A_y = A_y^{local} + \alpha A_y^{edge} + \beta A_y^{cloud}$, (6)

where a and b are weighted representing the amount of energy used by the edge server and the server in the cloud, respectively. Only the IoT device's use of energy is taken into account when = = 0 is met. We are going to overlook the amount of energy that is used during task transmission for the sake of simplicity.

Calculating the energy needed to complete job v looks like this:

$$A_j = \begin{cases} c_j \, . t_{local,} p_{y,j} = p_{0,} \\ c_j \, . t_{edge,} p_{y,j} = p_{1,} \\ c_j \, . t_{cloud,} p_{y,j} = p_{2,} \end{cases} \quad (7)$$

where dlocal, dedge, and stand for neighborhood electricity per data bit, edge capacity per information bit, and cloud utilization of resources per communication bit, respectively. Therefore, the energy utilization design of process y may be stated by the following:

$$A_y = \sum_{j=1}^{M}[A_j, \alpha A_j, \beta A_j] . p_{y,j.} \quad (8)$$

### 3.1.4. The Formulation of the Problem

We begin by introducing a system function known as O(y, p), which can be described as a weighted average of the utilization of energy and delay in completing workflows. This is done to concurrently reduce both the amount of time it takes to finish all workflows and the amount of energy that is required to do so.

$$O(y,p) = \sum_{y=1}^{N}(D_y + \delta A_y)$$
$$= \sum_{y=1}^{N}\left[\sum_{j=1}^{M}\left(D_j^v + D_{j,j+1}^d\right) + \delta \sum_{j=1}^{M}[A_j, \alpha A_j, \beta A_j] p_{y,j}\right] \quad (9)$$

If there are a total of M processes, where each workflow has N-related jobs, and where represents the relative significance of utilization of energy and the amount of time it takes to do a task. The optimization issue may be recast as a a constraint-based reduction issue, denoted by P1, as seen in the following example:

$$(B_1): \min_{p} O(y,p) \tag{10}$$

$$g.d,: p_{y,j} \in \left(\begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}\right) \tag{11}$$

### 3.2. Interpretable Deep Gaussian Naive Bayes

Interpretable Deep Gaussian Naive Bayes (IDGNB) is a machine learning algorithm that can be used as a classifier in a framework for job dispatching in edge-cloud computing. The IDGNB algorithm combines the interpretability of the Gaussian Naive Bayes (GNB) algorithm with the expressiveness of a deep neural network (DNN). The IDGNB algorithm is based on the GNB algorithm, which assumes that the features of a sample are conditionally independent given the class label. However, the GNB algorithm is limited in its ability to model complex non-linear relationships between the features and the class label. This is where the DNN comes into play. The IDGNB algorithm combines the GNB algorithm with a DNN by using the GNB algorithm to estimate the conditional probabilities of the features given the class label and using a DNN to model the non-linear relationships between the features and the class label. The DNN is trained to learn the feature representations that are most useful for predicting the class label.

The IDGNB algorithm has several advantages in the context of a task-unloading framework for edge computing. First, the algorithm is interpretable, which means that it can provide insights into the decision-making process of the classifier. This is important in scenarios where it is necessary to explain the reasons behind the decision to delegate a job to the cloud server. Second, the IDGNB algorithm can handle high-dimensional feature spaces, which is important in the context of edge-cloud computing, where there may be many features that need to be considered while choosing to transfer a job to a cloud server. Finally, the IDGNB algorithm can be trained on small datasets, which is important in the context of edge devices, where the amount of training data may be limited due to the limited storage capacity.

The Naive Bayes Classifier, sometimes known as the GNB, is a straightforward critical classifier that utilizes Bayes' theory. It does this by constructing a plausibility model based on the category definition for every feature vector that is included in the training set. Make your classifications utilizing the Maximum A Posteriori judgment rule as you are putting the test to the test.

#### 3.2.1. The Model

The purpose of any deterministic classifier is to, given a set of features ranging from x1 to xn and a set of classes ranging from c1 to ck, estimate the possibility that the features will appear in every class and then provide the class that is most likely to include those characteristics. Because of this, we need to be able to compute P(ci |x0,..., xn) for each class individually. The Bayes rule is what we make use of when trying to do this. To refresh your memory, the Bayes rule is as follows:

$$O(E|P) = \frac{o(P|E)o(E)}{O(P)} \tag{12}$$

$$(X = V|Y) = \frac{o(Y|X=V)o(X=V)}{\sum_{r=1}^{|V|} o(Y|X=V_r)o(X=V_r)} \tag{13}$$

Both the numerator and the denominator have the potential to become rather tiny. This is often the case because might be quite near to the null value and we calculate several of them with one another. One needs just to take the record of the operator to stop underflows from occurring. Because of this, the following should be done to avoid underflows: Using the MAP (maximum a posteriori) selection rule, we do not need to calculate the by value if our sole concern is to determine to which class (y) the input ($y = y_1,..., y_n$). In this scenario, we just consider it important to know what category the data being provided belongs to. To avoid underflows in the numerator, we may easily prevent them by taking the logarithm of the expression $\log(o(y|X = V)o(X = V))$.

$$\hat{x} = \underset{r \in \{1,\dots\dots|V|\}}{argmax}\ o(V_r|y_{1,\dots\dots}y_m) \tag{14}$$

$$\underset{r \in \{1,\dots\dots|V|\}}{argmax}\ o(V_r)\prod_{j=1}^{m} o\,(y_j|V_r) \tag{15}$$

which changes when the log is taken:

$$\hat{x} = \underset{r \in \{1, \dots \dots |V|\}}{argmax\ log(o(V_r|y_{1,\dots\dots}y_m))} \tag{16}$$

$$\underset{r \in \{1, \dots \dots |V|\}}{argmax\ log\left(o(V_r)\prod_{j=1}^{m} o\,(y_j|V_r)\right)} \tag{15}$$

$$\underset{r \in \{1, \dots, |V|\}}{argmax}\left(\left(\log\,(o(V_r)) + \sum_{j=1}^{m}\log\,(o(y_j|V_r))\right)\right) \tag{17}$$

$$O(y_j|V_r) = \frac{1}{\sqrt{2\pi\sigma V_r^2}}a\ -\ \frac{(y_j-\mu_V\,r)}{2\pi\sigma V_r^2} \tag{18}$$

$* \sigma = standard\ deviation, \mu = mean$

### 3.2.2. Training

Assume for the moment that he has a database in which each line details the class to which an item belongs as well as the characteristics of that entity. To train the algorithm based on this data collection, we need to compute the value of all attributes of every class, the variation of every variable of every class, and the prior terms. Only then can we begin. Therefore, we are in a position to compute the Gaussian Bayes Probability since we now have all of the required data. Furthermore, to prevent numerous accesses on the informational array, which is not the best choice for deploying hardware, to calculate variance, we use a method that is presented below.

$$\sigma^2 = \frac{\sum(Y-\mu)^2}{M} = \frac{\sum Y^2}{M} - \mu^2 \tag{19}$$

### 3.2.3. Classification

Now that we've established a method for estimating the chance that a certain data point belongs to a particular category, we have to be able to utilize this information to make classifications. This is handled in a relatively straightforward way by the Naive Bayes algorithm; all that is required is to choose the ci that, given the characteristics of the data points, has the highest probability. The name for this kind of decision-making principle is the PDR (Posteriori decision rule). This is because, when we formulate the Bayes rule, we only utilize the likelihood and prior value, which are denoted by the words Q(A|B) and Q(B), respectively. This is the reason why this is the case.

## 4. RESULTS AND DISCUSSION

The theoretical analysis presented above is supported in this part by simulations and experiments. We suppose that every device is dispersed at random across a 100 by 100 $m^2$ circular region. Interpretable deep Gaussian naive Bayes technique is a type of probabilistic classification algorithm that can be used in machine learning applications. It is commonly used for text classification and other types of data analysis.

Task offloading framework is a technique used in edge-cloud computing, where tasks are distributed between edge devices and cloud servers to optimize resource usage and reduce latency. By combining the two techniques, researchers may have been able to develop a more efficient and effective method for task allocation in edge-cloud computing. The existing methods for comparing are the optimal matching algorithm [15], Greedy algorithm [16], and Fast matching algorithm [17]. Tasks may be transferred to the cloud, where there are abundant computer resources, and less energy is used during job execution. The performance of job execution under various offloading mechanisms is analyzed and contrasted in Figures 2 and 3. The vertical dimension in Figure 2 energy-saving rate depicts the ratio of the decrease in overall task execution consumption of energy under this offloading approach when compared to the local performance of all tasks.

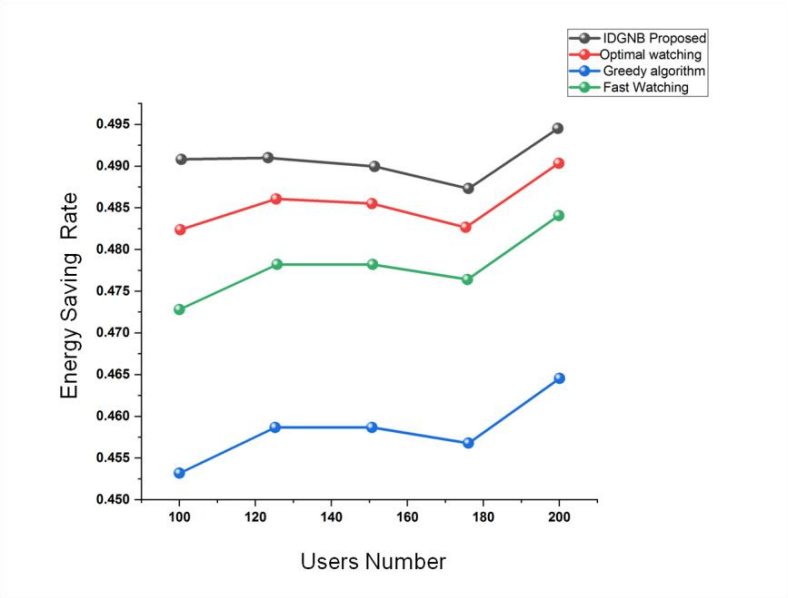


Figure 2. Energy saving ratio

Table 1.Results of energy saving rate

| Methods | Values |
|---|---|
| Optimal matching | 0.492 |
| Greedy algorithm | 0.464 |
| Fast matching | 0.487 |
| IDGNB (proposed) | 0.495 |

The system's energy usage for ED will reduce as their number grows. The edge cloud platform will have more idle devices as more devices are added, offering the ED more possibilities to save energy consumption while doing the function. Along the vertical axis of Figure 3, the quantity of tasks that are not completed inside the time-delayed restriction is shown.

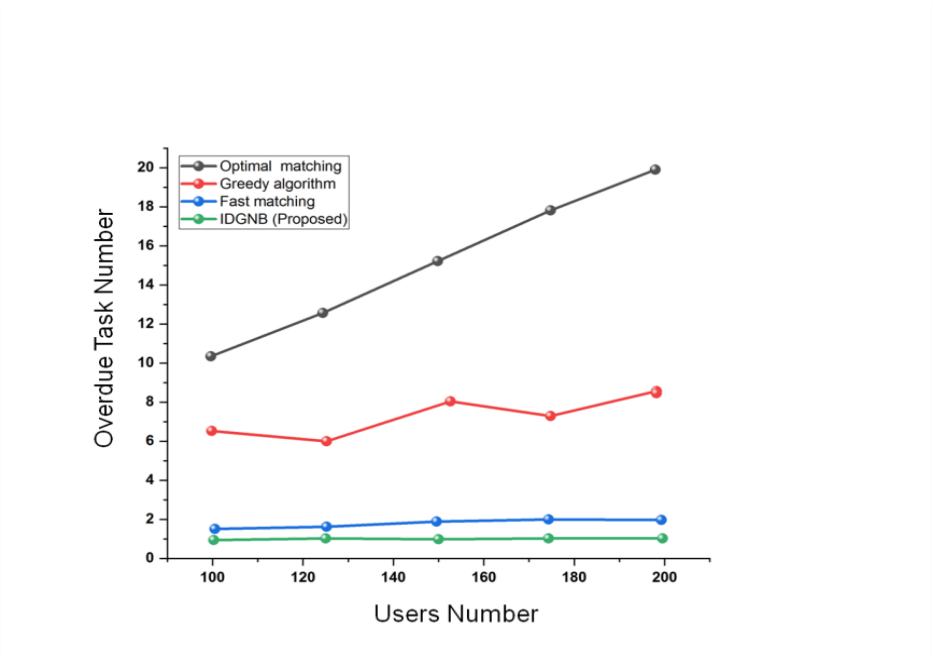


Figure 3. Overdue task for the user

Table 2. Results for overdue task

| Methods | Values |
|---|---|
| Optimal matching | 20 |
| Greedy algorithm | 8 |
| Fast matching | 2 |
| IDGNB (proposed) | 1 |

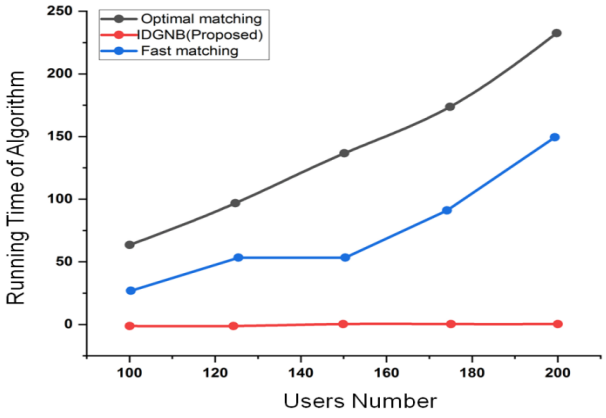


Figure 4. Algorithm's running time

Table 3. Results of running time

| Methods | Values |
|---|---|
| Optimal matching | 250 |
| Fast matching | 140 |
| IDGNB (proposed) | 10 |

When compared to many alternative methods, the optimum matching algorithm and the quick matching algorithm may successfully guarantee that the work is performed during the time delay restriction, as illustrated in Figure 3. The suggested method running times are simulated in Figure 4. The suggested approach may achieve almost ideal offloading performance with less complexity, as shown in the picture. This study mimics the impact of probabilistic generating tasks on task offloading performance. We presume that there are 150 devices in the system.

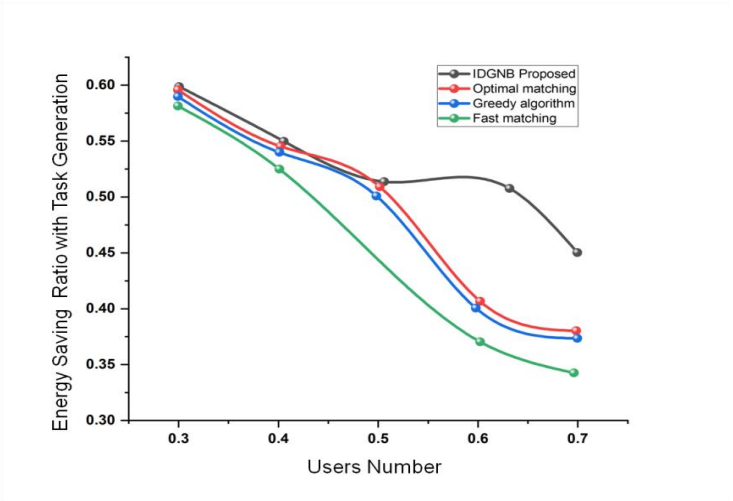


Figure 5. Task Generation for energy saving

Table 4. Energy saving ratio

| Methods | Values |
|---|---|
| Optimal matching | 0.392 |
| Greedy algorithm | 0.381 |
| Fast matching | 0.351 |
| IDGNB (proposed) | 0.473 |

The energy savings rate of the system will drop when task generation frequency rises, as illustrated in Figure 5 since there are fewer idle resources available in the edge and cloud. This makes it more challenging for task consumers to offload jobs sequentially for execution. As a result when compared to existing methods our proposed method is more efficient than existing methods.

## 5. CONCLUSION

In general, the benefits of offloading framework could include improved performance, reduced latency, and better resource utilization. Offloading certain tasks to more powerful or specialized computing resources can help improve overall system efficiency and responsiveness. However, the effectiveness of the offloading framework would depend on the specific implementation, the nature of the computing tasks being offloaded, and the characteristics of the computing resources involved. This research has suggested a unique IDGNB framework to address the task offload decision-making issue in heterogeneous edge and cloud collaborative computing environments. The problem of neural networks' inflexibility to change has been solved. It includes a task offload judgment model that is based on distributive GNB as well as a deep meta-learning-based training initial variable model. Both of these models can manage the challenge of making decisions regarding offloading tasks for corner-cloud computation and quickly adjust to a changing MEC environment.

IDGNB has a more positive impact on task offloading choices than binary offloading systems and traditional GNB-based partial offloading schemes, according to experimental data. Additionally, the model has improved portability and a quicker rate of environment learning thanks to the usage of meta parameters. The model may rapidly converge as the MEC environment evolves, and only a few learning steps are needed to determine the best offloading options at reasonable prices. Future studies will focus on improving the meta-learning method, specifically the starting parameters' ability to be adjusted automatically in response to environmental factors so that it is more capable of adapting to task offloading choices in large-scale MEC contexts. We will also concentrate on problems like resource distribution and bandwidth adjustment using serverless computing edge computing frameworks. The offloading model may also offer the relevant resource scheduling plans in addition to the results of task offloading decisions.

## REFERENCES

[1] Y. Ai, M. Peng, and K. Zhang, “Edge computing technologies for Internet of Things: a primer,” *Digit. Commun. Networks*, vol. 4, no. 2, pp. 77–86, Apr. 2018, doi: 10.1016/J.DCAN.2017.07.001.

[2] D. Wu, G. Shen, Z. Huang, Y. Cao, and T. Du, “A Trust-Aware Task Offloading Framework in Mobile Edge Computing,” *IEEE Access*, vol. 7, pp. 150105–150119, 2019, doi: 10.1109/ACCESS.2019.2947306.

[3] A. Naouri, H. Wu, N. A. Nouri, S. Dhelim, and H. Ning, “A Novel Framework for Mobile-Edge Computing by Optimizing Task Offloading,” *IEEE Internet Things J.*, vol. 8, no. 16, pp. 13065–13076, Aug. 2021, doi: 10.1109/JIOT.2021.3064225.

[4] H. Liao, Y. Mu, Z. Zhou, M. Sun, Z. Wang, and C. Pan, “Blockchain and Learning-Based Secure and Intelligent Task Offloading for Vehicular Fog Computing,” *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 7, pp. 4051–4063, Jul. 2021, doi: 10.1109/TITS.2020.3007770.

[5] A. K. Shukla and V. Suresh Kumar, “Cloud Computing with Artificial Intelligence Techniques for Effective Disease Detection,” *Int. J. Data Informatics Intell. Comput.*, vol. 2, no. 1, pp. 32–41, Mar. 2023, doi: 10.59461/ijdiic.v2i1.45.

[6] X. Zhang, R. Zhou, Z. Zhou, J. C. S. C. S. Lui, and Z. Li, “An Online Learning-Based Task Offloading Framework for 5G Small Cell Networks,” in *49th International Conference on Parallel Processing - ICPP*, New York, NY, USA: ACM, Aug. 2020, pp. 1–11. doi: 10.1145/3404397.3404417.

[7] M. Gali and A. Mahamkali, “A Distributed Deep Meta Learning based Task Offloading Framework for Smart City Internet of Things with Edge-Cloud Computing,” *J. Internet Serv. Inf. Secur.*, vol. 12, no. 4, pp. 224–237, Nov. 2022, doi: 10.58346/JISIS.2022.I4.016.

[8] J. Almutairi and M. Aldossary, “A novel approach for IoT tasks offloading in edge-cloud environments,” *J. Cloud Comput.*, vol. 10, no. 1, p. 28, Apr. 2021, doi: 10.1186/s13677-021-00243-9.

[9] H. A. Alharbi, M. Aldossary, J. Almutairi, and I. A. Elgendy, “Energy-Aware and Secure Task Offloading for Multi-Tier Edge-Cloud Computing Systems,” *Sensors*, vol. 23, no. 6, p. 3254, Mar. 2023, doi: 10.3390/s23063254.

[10] K. I. Jones and S. R, "Information Security: A Coordinated Strategy to Guarantee Data Security in Cloud Computing," *Int. J. Data Informatics Intell. Comput.*, vol. 2, no. 1, pp. 11–31, Mar. 2023, doi: 10.59461/ijdiic.v2i1.34.

[11] S. Alhelaly, A. Muthanna, and I. A. Elgendy, "Optimizing Task Offloading Energy in Multi-User Multi-UAV-Enabled Mobile Edge-Cloud Computing Systems," *Appl. Sci.*, vol. 12, no. 13, p. 6566, Jun. 2022, doi: 10.3390/app12136566.

[12] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu, "EEDTO: An Energy-Efficient Dynamic Task Offloading Algorithm for Blockchain-Enabled IoT-Edge-Cloud Orchestrated Computing," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2163–2176, Feb. 2021, doi: 10.1109/JIOT.2020.3033521.

[13] Z. Zhang, N. Wang, H. Wu, C. Tang, and R. Li, "MR-DRO: A Fast and Efficient Task Offloading Algorithm in Heterogeneous Edge/Cloud Computing Environments," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 3165–3178, Feb. 2023, doi: 10.1109/JIOT.2021.3126101.

[14] A. M and S. M, "Dynamic Mobile Cloud Eco System Security - A Review," *Int. J. Data Informatics Intell. Comput.*, vol. 2, no. 1, pp. 62–69, Mar. 2023, doi: 10.59461/ijdiic.v2i1.44.

[15] Z. Cheng, Z. Gao, M. Liwang, L. Huang, X. Du, and M. Guizani, "Intelligent Task Offloading and Energy Allocation in the UAV-Aided Mobile Edge-Cloud Continuum," *IEEE Netw.*, vol. 35, no. 5, pp. 42–49, Sep. 2021, doi: 10.1109/MNET.010.2100025.

[16] J. Almutairi, M. Aldossary, H. A. Alharbi, B. A. Yosuf, and J. M. H. Elmirghani, "Delay-Optimal Task Offloading for UAV-Enabled Edge-Cloud Computing Systems," *IEEE Access*, vol. 10, pp. 51575–51586, 2022, doi: 10.1109/ACCESS.2022.3174127.

[17] X. Zhang, H. Zhang, X. Zhou, and D. Yuan, "Energy Minimization Task Offloading Mechanism with Edge-Cloud Collaboration in IoT Networks," in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, IEEE, Apr. 2021, pp. 1–7. doi: 10.1109/VTC2021-Spring51267.2021.9449054.

## BIOGRAPHIES OF AUTHORS

**Prabhdeep Singh** is an assistant Professor at School of Computer Applications department, Babu Banarasi Das University, Lucknow. He pursued B.Tech from Saroj Institute of Technology and Management, Lucknow (U.P.T.U.) and M.Tech CMJ University , Shillong. He is also pursuing Ph.D. (Part Time) from Amity School of Engineering & Technology, Lucknow. He has over 13 years of experience in technical education inclusive one year as a software engineer. He has published numerous research papers in international journals. He has also published two patents. He can be contacted at email: prabhdeepcs@gmail.com