International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 04 | April - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

A Novel Framework for Secured Offload Intensive Computation Tasks in Smartphones from the Mobile Device to the Cloud

Dr. G. Vinoda Reddy, Mamatha B

Professor of CSE (AI & ML), CMR Technical Campus, Hyderabad, Telangana, India.
Assistant Professor, Dept of CSE (AI & ML), CMR Technical Campus, Hyderabad.

Abstract: In order to provide mobile customers scalable and resource-rich services, mobile cloud computing, or MCC, has arisen as a paradigm that blends the processing power of cloud computing with the widespread usage of mobile devices. However, there are a lot of efficiency and security-related issues when integrating mobile devices with cloud resources. In order to solve these issues, this study suggests a safe and effective architecture for mobile cloud computing that maximises resource utilization, improves data privacy, and guarantees secure communication between mobile devices and cloud servers. To accomplish its goals, the framework makes use of technologies including edge computing, encryption methods, and authentication systems. The efficacy and performance of the suggested framework in practical situations are shown by experimental assessments.

Keywords: Smartphones, mobile cloud computing, computation offloading, security, particle swarm optimization.

I.INTRODUCTION

We use smartphones extensively in our everyday lives. These gadgets do, however, have several drawbacks, including a short battery life, low processing power, a tiny memory capacity, and erratic network access. As a result, several strategies have been put out to lessen these restrictions and increase battery lifespan by using the offloading approach. This research proposes a unique paradigm to shift heavy computing operations from mobile devices to the cloud. This system makes use of an optimisation model to dynamically decide which tasks to offload depending on four key metrics: execution time, memory use, CPU utilisation, and energy consumption. Furthermore, an additional security layer is offered to ward against attacks on the transmitted data stored in the cloud. The outcomes of the experiments shown that the framework can choose an appropriate offloading choice while attaining a notable performance boost for various kinds of mobile application workloads. Furthermore, the framework may shield application data from danger, in contrast to earlier methods.

II.RELATED WORK

Applications for smartphones are many and include voice recognition, video gaming, face identification, augmented reality, picture and video processing, and more. The complexity of these applications and the need for processing power are growing along. The length of battery life, however, continues to be one of the primary obstacles to enhancing processing needs via battery upgrades, even with the developments in smartphones [1].
Through the internet, cloud computing [2], [3] offers limitless resource access. Self-service provisioning, flexibility, wide network access, resource pooling, affordability, and simplicity of use are just a few benefits of cloud computing.

Therefore, to get over the restrictions of smartphone devices, mobile cloud computing [4] is created. A new paradigm called "mobile cloud computing" combines cloud computing with mobile devices to improve application speed and battery life. It has been suggested in recent research [5], [6] to offload all or a portion of mobile apps from mobile devices to the cloud for remote execution. When deciding which tasks to offload, these frameworks are made to balance a number of restrictions, including the mobile device's energy consumption, CPU utilisation, execution time, battery life left, and network data transfer volume. Nevertheless, the majority of these models do not take memory consumption into account while making the offloading choice. One of the primary resources used by mobile apps is

memory. Furthermore, security measures are not used to defend offloaded data against intrusions. Thus, the primary goal of this study is to create a new model that integrates the majority of the aforementioned limitations in order to enhance mobile application performance and safeguard application data against intrusions.

Specifically, we presented a unique architecture that offloads just the computationally demanding activities of mobile apps via computation offloading. We developed an optimisation model that makes the choice about offloading.

The main findings and contributions of this paper is, this paper suggests a unique architecture that reduces the amount of network connection needed by offloading just heavy workloads rather than all apps. An optimisation model is developed to dynamically decide whether to offload a job at runtime based on four primary constraints: task execution time, CPU utilisation, memory utilisation, and energy consumption. Using the AES encryption approach, a new security layer is created to encrypt the task's data before it is sent to the cloud. The experimental tests use three distinct kinds of mobile apps to evaluate this framework and demonstrate how to choose an appropriate offloading choice for enhanced application performance.

III.LITERATURE WORK

In order to overcome the limitations of mobile devices, a number of strategies have lately been put forth that include shifting compute duties to cloud resources for distant execution [5]. Certain strategies shift only a single process from the mobile device to the cloud's cloned virtual machine (VM) [7]. Static analysis and dynamic profiling modules are combined in [7] to divide the application and choose which process is sent to the cloud.

Kosta et al. [8] employed an execution controller to monitor the methods' remote execution through the usage of a profiler module, and they built virtual machines (VMs) of a whole smartphone system in the cloud. The energy-intensive need for fundamental synchronisation with the cloud-based clone virtual machine (VM) is the primary disadvantage of and. Additionally, when being transferred to the cloud, application data are not secure from intrusions. By sending only the intensive services rather than the entire process to the cloud, addresses the synchronisation issue. Additionally, the authors develop a methodology to decide which services should be offloaded. This approach, however, always favours remote execution and is incredibly static and straightforward. Sometimes it's better to run services on a mobile device rather than dumping them to the cloud. Security measures of any kind must be used to safeguard the sent data.Additional frameworks are suggested in [11] that include splitting up the application and offloading expensive techniques. Like our framework, these frameworks make judgements about offloading using an integer linear programming paradigm. The choice to offload is made based on energy consumption limitations, total reaction time, and remaining battery life, without taking memory utilisation or security into account.

In comparison, the entire Android programme is offloaded to the cloud, which uses a lot of resources because of the volume of data that is sent over the network. Any security method should also be safeguarded because the programme that is transferred to the cloud has to be secure. Which solely offloads resource-intensive functions and uses the Software-as-a-Service concept to configure demanding services on the cloud server, aims to minimise both data transmission and energy usage. Like in [7] requires little synchronisation between the mobile device and the cloud server node, adding to the battery drain and increasing the attack surface of the offloaded data.

In order to provide a dynamic choice on the where, when, and how to offload the activities of the mobile application, a context-aware mobile cloud computing system with an estimating model was developed in [7]. Nevertheless, this framework required extra energy as it employed a discovery service to get the hardware details of the cloud resources every minute. Furthermore, there was no security safeguard for the sent data.

In order to reduce the energy efficiency cost incurred by the mobile device in executing the application, an iterative method merging resource scheduling strategy and dynamic offloading is presented in [8]. Task priority and completion time deadline were taken into consideration by the writers as the primary model constraints. The computation offloading selection, CPU clock frequency management in local computing, and transmission power allocation in cloud

computing were the three primary components of this approach. However, in choosing which tasks to offload, this system did not take memory use [9] into account or use any security measures to safeguard the transmitted data from intrusions.

Reducing overall energy usage while meeting time and reliability requirements has recently been studied in [2]. The paper suggested an energy-aware dynamic task scheduling method that employed the critical path assignment technique and directed acyclic graph (which displays the task precedence and communication cost) to determine the best execution order for each job that reduced the total energy consumption. But just the energy consumption statistic was taken into consideration by this model; other significant metrics, such memory use, CPU utilisation, and remaining battery life, were left out and are seen to be equally relevant.

Other works took into account memory use limits in their models, taking into account all of the previously described work. Nevertheless, no security method was used to safeguard the data that was offloaded to the cloud. This study presents a model that addresses four distinct limitations in the offloading decision-making process: memory use, execution time, CPU utilisation, and energy consumption. The offloading choice was decided dynamically at runtime by this model. To further safeguard the data that is offloaded to the cloud, we added a new security layer to this architecture.
Three distinct kinds of Android-developed mobile apps are used to evaluate the suggested framework.

IV.SECURED FRAMEWORK FOR MOBILE CLOUD COMPUTING

Mobile devices have become an integral part of our daily lives, offering a wide range of applications and services. However, the limited computational and storage capabilities of mobile devices often hinder their ability to execute resource-intensive tasks efficiently. Cloud computing provides a solution to this limitation by offering virtually unlimited computational resources and storage capacity. Mobile cloud computing (MCC) extends this concept by integrating cloud services with mobile devices, enabling users to access powerful computational resources and storage facilities from their mobile devices.

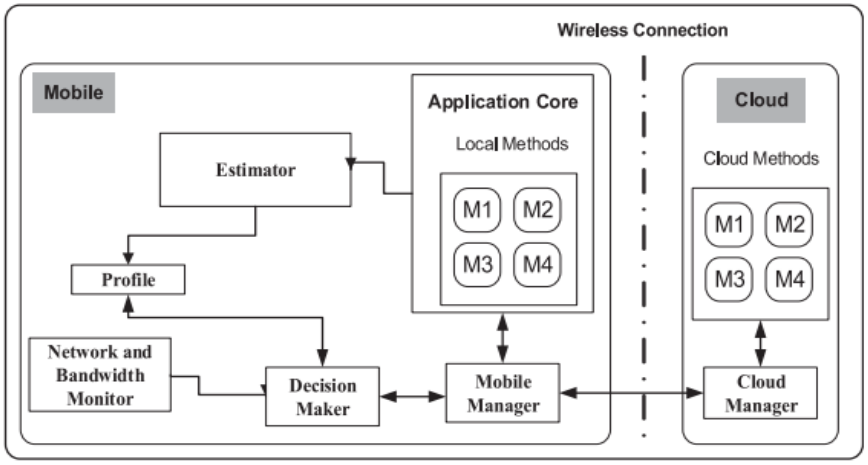

Fig. 1. Framework architecture.

This section explains the framework's architecture and demonstrates how its modules operate together to accomplish the system's design objectives. Furthermore defined is the linear optimisation model. A thorough analysis of the offloading choice is also displayed. Next, we present an algorithm that makes clear how this framework functions. Lastly, we determine the actions needed to include this framework in the development process. manager of ouds.

The framework operates first at the method level, requiring developers to put an annotation (@Remote) above each demanding method while they are still in the development stage. These techniques may be executed remotely from the cloud and should necessitate additional processing. These techniques cannot: a) rely on the user interface; or b) make

use of any mobile I/O device, including an accelerometer, camera, or GPS. A binary file containing these method codes and the libraries that go with them is then transferred to the cloud during the installation process. We'll talk about this area later.

Estimator: During the installation process, the estimator module is in charge of determining these techniques for both remote execution on the cloud and local execution on the mobile device, with varying input sizes (stored as a sample). The module then retrieves the values of CPU and memory utilisation, energy consumption, execution time, and energy consumption for each annotated method for these various input sizes (the energy consumption and CPU utilisation are measured using the minimum eye programme). Lastly, the values are conveyed and transmitted to the module for the profile.
Profile: For every annotated method, the profile module retrieves the values of execution time, memory use, CPU utilisation, and energy consumption from the estimator module. Next, the module stores these data in a new file that it produces for each method. These files serve as a history-based file for the decision maker module to employ for offloading, and they are updated following each process that runs.

Network and Bandwidth Monitor: This module merely keeps track of the network's current condition. It also collects data on the signal strength of both Wi-Fi and cell connections, as well as the state and bandwidth of each connection (using programming code). Subsequently, the decision maker module receives this data in order to facilitate the decision to offload.

Maker of Decisions: The decision make module, which forms the basis of the suggested framework, includes a decision-making algorithm and an integer linear programming model that forecast the runtime locations of the annotated methods' execution. Subject to specific limitations, the model seeks to identify an application partitioning strategy that minimises smartphone energy consumption, transfer data, memory use, and CPU utilisation. The decision-maker also considers all of the information gathered from the network and bandwidth monitor modules, as well as the profile.

The suggested framework's execution flow is covered in this section. Algorithm 1 shows the intricate workings of the framework as well as the decision-making process involved in offloading the annotated procedure. This approach uses O(n) for its time complexity and doesn't need any more computing power from the smartphone. Initially, the procedures in the mobile application are divided into two categories during the development stage. The first category consists of computationally complex techniques that need greater processing power and are annotated by developers. As previously mentioned, the second technique is dependent on the hardware of the device and has to be run locally. Next, when the application is running, the decision maker module uses the network and bandwidth monitor module to verify the network state and reads the annotated method name (first kind of methods).

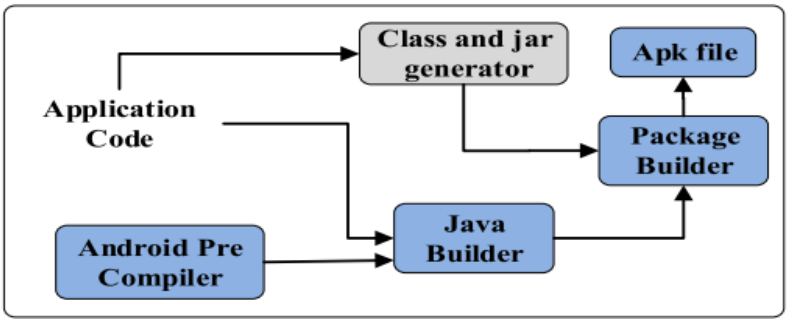

Fig. 2. Integration between builders in the building process.

The decision maker module reads the transfer data size, memory usage, CPU utilisation, and energy consumption through the profile module, where these values are stored at the installation step and updated during each run. If there is no connection or a failed connection, then all of the methods are performed locally on the mobile device. Next, the model of optimisation is solved. Which procedure is offloaded for remote execution and which is run locally on the mobile device are determined by the optimisation model. When offloading, the method data is first encrypted using

the AES technique before being sent to the cloud and run via communication between the mobile manager and cloud manager. Finally, the profile file is updated with new settings based on the operating process, regardless of whether the method is executed locally or remotely. The necessary procedures to incorporate the suggested framework into the mobile application are explained in this section. Our system operates at the method level, identifying the methods that may be offloaded using an annotation and Java reflection.

The developer must first put an annotation (@Remote) above each demanding method that can be delegated to a remote execution process during the development stage. Subsequently, every Android programme undergoes three primary builders in order to produce the APK installation file, as seen in Figure 2. The Android Pre-Compiler is the initial builder; it creates Java source files for any service interface and from your Android resources. The files produced by the first builder are then assembled by the Java Builder. The compiled files are then obtained by the package builder, which packs them into an APK file. The Class and Jar Generator is a new builder that our framework incorporates. Using this builder, a class with all the annotated code for remotely executable methods and the corresponding libraries needed to run these methods on the cloud are created. Next, when the installation stage occurs, the builder uses the constructed class and libraries to create a binary file that is uploaded to the cloud. The Package Builder combines the output from this builder with the output from the Java Builder to create an APK file.

V.RESULTS AND DISCUSSION

As indicated in Table 1, three distinct categories of mobile apps are used to assess the suggested framework. Four parameters are measured by the experimental results when the application methods are executed locally on a mobile device and when the framework is used to offload the methods to the cloud. These variables include memory use, battery consumption, CPU utilisation, and processing time. The assessment demonstrates how the suggested methodology for enhanced performance might help these applications.

Application	Description
Face Detection	Detect faces from an image and draw a rectangle on each face detected.
Gaussian Blur	Blurring an image by a Gaussian function.
Quick Sort	Sort a given set of integer array elements by using Quicksort

Table 1. Application used in experimental.

A server node, a Wi-Fi wireless network of radio type 802.11g, and a Samsung Galaxy S Plus mobile smartphone make up the experimental configuration used to evaluate the suggested architecture. Running Android 4.4.2, the Samsung Galaxy S plus GT-I9001 has a Qualcomm MSM8255T CPU, 512 MB of RAM, a 1650 mAh 3.7 volt battery, and an integrated WiFi interface. The server node is equipped with an Intel Core(TM) i5-2500 CPU operating at 2.4 GHz, 4 GB of RAM, a 600 GB hard drive, and a 100 Mbps network interface. It is running Microsoft Windows 7 Ultimate 64-bit. With a 54 Mbps physical layer data rate available, the mobile device connects to the wireless network via a Wi-Fi wireless network connection of radio type 802.11g. Processing time, CPU utilisation, battery consumption, and memory use are all measured during the evaluation using the Little Eye V2.41 programme.

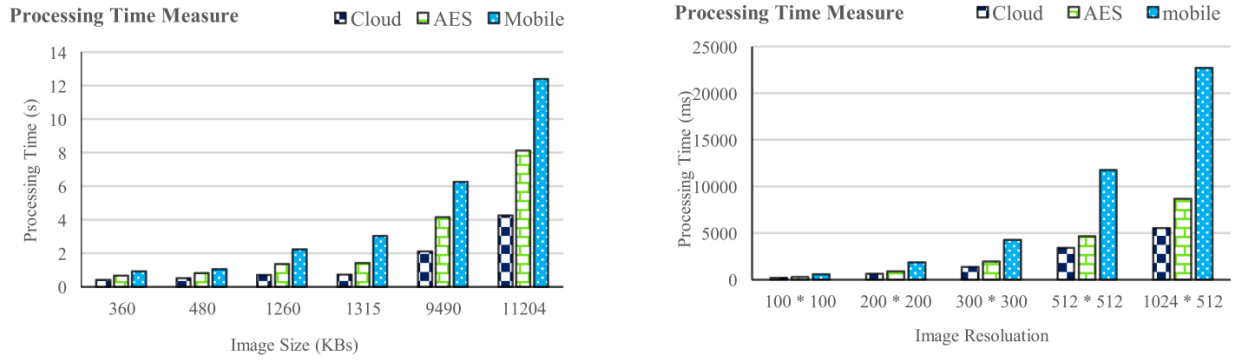

Figure 3. Usage of processing time for image resolutions.

Data input may be required for the application methods to function. In the event that this method is offloaded for execution, data are sent over the network and stored on the cloud side. These data are therefore open to assaults. Data and communication security requires the use of cryptographic algorithms. There are two types of cryptographic algorithms: symmetric key algorithms and asymmetric key algorithms. While asymmetric key algorithms, also known as public key, use a public (shared) key to execute encryption and use additional private key decryption processes, symmetric key algorithms, also known as single key, use a private (shared secret) key to execute encryption and decryption processes. The most well-known asymmetric algorithms are RSA, DSA, PGP, SSH, and SSL, whereas the most well-known symmetric algorithms include DES, TDES, AES, RC6, Twofish, Blowfish, Serpent, and MARS.

To secure the security of this data and reduce energy usage, symmetric key algorithms are chosen since they are quicker and use less energy than asymmetric algorithms. To safeguard the transmitted data of the apps, the AES algorithm is chosen and put to use as an encryption mechanism.

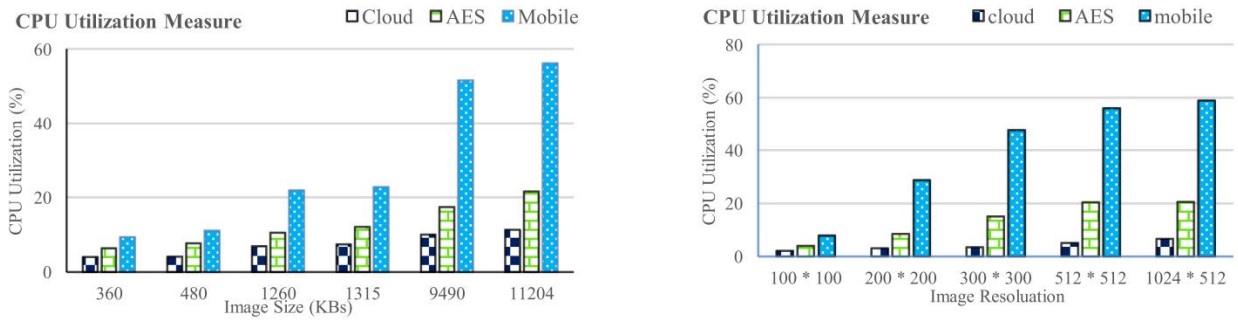

Figure 4. CPU utilization for MCC.

In the suggested framework, the linear model solves for the parameters of the techniques to determine the offloading option and chooses the right choice for each of the three applications. To support our findings, the framework is changed to enable the quick-sort application to operate remotely on the cloud.

The six photos utilised in the trial work for the face detection application have sizes of 360, 480, 1260, 1315 KB, and 9 and 11 MB. For the Gaussian blur application, five photos with resolutions of 100 ~ 100, 200 ~ 200, 300 ~ 300, 512 ~ 512, and 1024 ~ 512 are utilised. The quick-sort programme uses five arrays of 100, 500, 5000, 50000, and 100000 entries each. For every input, each programme is run 20 times, yielding the average values.

The results of the processing time measure for each of the three applications are displayed in Figure 3. Three distinct situations are used to build applications for Gaussian blur and face identification.

Two scenarios are used to run the rapid sort programme since offloading to the cloud takes more time. Therefore, when choosing local execution is desirable, there is no need to add a security layer.

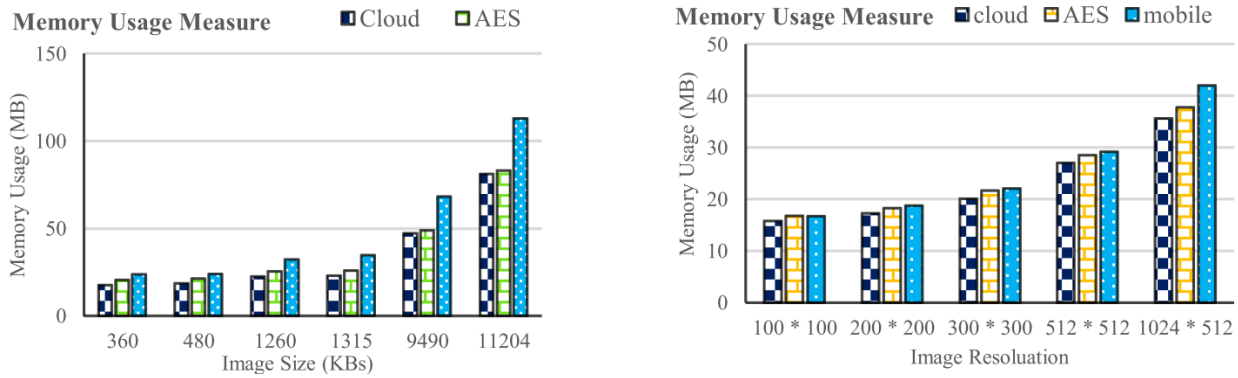

Figure 5. memory usage measure for cloud AES.

The processing time findings for Gaussian Bluer and face detection applications are displayed in Figs. 3a and 3b. The quick-sort application's processing time findings are displayed in Fig. 3c. The x-axis in Figure 3a represents the picture size in KBs, while the y-axis shows the processing time in seconds. The x-axis in Figure 3b represents picture resolution, while the y-axis shows processing time in milliseconds. In Fig. 3c, the y-axis represents processing time in milliseconds, while the x-axis shows the number of elements in the array. Without the suggested framework, the face detection technique or Gaussian blur approach takes an average of 5–8.5 s to execute, as Figs. 3a and 3b demonstrate. When the apps are run using the suggested framework, the average processing time is 1.5–2.5 seconds when no security is used, and around 2.8–3.5 seconds when the transmitted data is encrypted to the cloud using the AES security method. As a result, the suggested architecture can reduce the time needed to operate cloud applications by 65–73 percent. Fig. 3c, on the other hand, shows that local quick-sort method execution on the mobile device takes the least amount of processing time due to the fact that face identification and Gaussian blur apps require time for job completion and data computation. Hence, using the cloud to do these activities is helpful, while a quick-sort application with a straightforward objective might benefit from being executed locally on the mobile device.

The results of the CPU utilisation measurement for each of the three apps are displayed in Figure 4. The smartphone applications, as illustrated in the figure, use an average of 30% of the CPU when run locally without the use of our framework (with the exception of the quick-sort application, which requires more processing to be offloaded than when run locally). When using the suggested architecture, smartphone apps consume an average of 7% of the CPU without any security layer and around 12% when an AES security layer is applied. Applications requiring significant calculation and modest data transport require a variance in CPU utilisation that rises.

VI.CONCLUSION

The efficiency of offloading computing from the mobile device to the cloud is improved by the unique, optimised, and secure architecture that is provided in this research. Only application methods that are heavy users of mobile resources may be offloaded by this framework. An integer linear programming model with a 0–1 coefficient is used to decide which tasks to outsource. Four constraints memory use, CPU utilisation, energy consumption, and execution time are used to dynamically make this selection at runtime. In the event of offloading, the framework incorporates an additional security layer that protects the method data using the AES algorithm before to transmission to the cloud. By lowering the use of mobile resources, such as processing time, battery consumption, CPU utilisation, and memory usage, the assessment results demonstrated that the suggested framework may enhance the performance of mobile applications. This investigation also demonstrates how the suggested algorithm might choose wisely while offloading. We ultimately come to the conclusion that using the suggested architecture to run resource-intensive mobile application methods remotely on the cloud preserves mobile resources, particularly when the application requires a lot of computation and little data transmission. To cut down on latency, we must use the same concept going forward on edge computing servers rather than central cloud computing. To possibly save execution time and energy usage, we also need to enable parallelization for the method's cloud execution.

REFERENCES

[1] N. Vallina-Rodriguez and J. Crowcroft, "Energy management techniques in modern mobile handsets," IEEE Commun. Surveys Tuts., vol. 15, no. 1, pp. 179–198, Jan-Mar. 2013.

[2] B. Sosinsky, Cloud Computing Bible. Hoboken, NJ: Wiley, 2010.

[3] G. Motta, N. Sfondrini, and D. Sacco, "Cloud computing: An architectural and technological overview," in Proc. Int. Joint Conf. Serv. Sci., vol. 3, 2012, pp. 23–27.

[4] Ravindra Changala, "Sentiment Analysis in Social Media Using Deep Learning Techniques", International Journal of Intelligent Systems and Applications in Engineering, 2024, 12(3), 1588–1597.

[5] Ravindra Changala, "Integration of IoT and DNN Model to Support the Precision Crop", International Journal of Intelligent Systems and Applications in Engineering, Vol.12 No.16S (2024).

[6] D. Kovachev, Y. Cao, and R. Klamma, "Mobile cloud computing: A comparison of application models," CoRR, vol. abs/1107.4940, 2011.

[7] A. U. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud omputing application models," IEEE Commun. Surveys Tuts., vol. 16, no. 1, pp. 393–413, 2014.

[8] M. Shiraz, A. Gani, R. H. Khokhar, and R. Buyya, "A review ondistributed application processing frameworks in smart mobile devices for mobile cloud computing," IEEE Commun. Surveys Tuts., vol. 15, no. 3, pp. 1294–1313, Jul.-Sep. 2013.

[9] Ravindra Changala, "UI/UX Design for Online Learning Approach by Predictive Student Experience", 7th International Conference on Electronics, Communication and Aerospace Technology, ICECA 2023 - Proceedings, 2023, pp. 794–799, IEEE Xplore.

[10] Ravindra Changala, "Optimization of Irrigation and Herbicides Using Artificial Intelligence in Agriculture", International Journal of Intelligent Systems and Applications in Engineering, 2023, 11(3), pp. 503–518.

[11] Ravindra Changala, Development of Predictive Model for Medical Domains to Predict Chronic Diseases (Diabetes) Using Machine Learning Algorithms And Classification Techniques, ARPN Journal of Engineering and Applied Sciences, Volume 14, Issue 6, 2019.

[12] D. Kovachev, T. Yu, and R. Klamma, "Adaptive computation offloading from mobile devices into the cloud," in Proc. IEEE Int.Symp. Parallel Distrib. Process. Appl., 2012, pp. 784–791.

[13] F. Xia, F. Ding, J. Li, X. Kong, L. T. Yang, and J. Ma, "Phone2cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing," Inf. Syst. Frontiers, vol. 16, no. 1, pp. 95–111, 2014.

[14] W. Zhang, Y. Wen, K. Guan, K. Dan, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," IEEE Trans. Wireless Commun., vol. 12, no. 9,pp. 4569–4581, Sep. 2013.

[15] Ravindra Changala, "Evaluation and Analysis of Discovered Patterns Using Pattern Classification Methods in Text Mining" in ARPN Journal of Engineering and Applied Sciences, Volume 13, Issue 11, Pages 3706-3717 with ISSN:1819-6608 in June 2018.

[16] Ravindra Changala "A Survey on Development of Pattern Evolving Model for Discovery of Patterns in Text Mining Using Data Mining Techniques" in Journal of Theoretical and Applied Information Technology, August 2017. Vol.95. No.16, ISSN: 1817-3195, pp.3974-3987.

[17] Ravindra Changala, Framework for Virtualized Network Functions (VNFs) in Cloud of Things Based on Network Traffic Services, International Journal on Recent and Innovation Trends in Computing and Communication, ISSN: 2321-8169 Volume 11, Issue 11s, August 2023.

[18] S. Guo, B. Xiao, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in Proc. IEEE INFOCOM, 2016, pp. 1–9.

[19] W. Z. Zhang, H. C. Xie, and C. H. Hsu, "Automatic memory control of multiple virtual machines on a consolidated server," IEEETrans. Cloud Comput., vol. 5, no. 1, pp. 2–14, Jan.-Mar. 2017.

[20] Y. Li, M. Chen, W. Dai, and M. Qiu, "Energy optimization with dynamic task scheduling mobile cloud computing," IEEE Syst. J.,vol. 11, no. 1, pp. 96–105, Mar. 2017.

[21] Ravindra Changala, Block Chain and Machine Learning Models to Evaluate Faults in the Smart Manufacturing System, International Journal of Scientific Research in Science and Technology, Volume 10, Issue 5, ISSN: 2395-6011, Page Number 247-255, September-October-2023.

[22] W. O. H. Rania and C. Babak, "A comparison of particle swarm optimization and the genetic algorithm," in Proc. Struct. Dyn. Mater. Conf., 2004, pp. 833–843.

[23] Ravindra Changala, A Novel Approach for Network Traffic and Attacks Analysis Using Big Data in Cloud Environment, International Journal of Innovative Research in Computer and Communication Engineering: 2320-9798, Volume 10, Issue 11, November 2022.

[24] Ravindra Changala, "Brain Tumor Detection and Classification Using Deep Learning Models on MRI Scans", EAI Endorsed Transactions on Pervasive Health and Technology, Volume 10, March 2024.

[25] B. Booba and T. V. Gopal, "Comparison of ant colony optimization and particle swarm optimization in grid scheduling," Australian J. Basic Appl. Sci., vol. 297, no. 4, pp. 230–235, 2014.