**International Journal of Applied Mathematics**
**Volume 39 No. 1s, 2026**
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

# MOBILE CLOUD OFFLOADING FRAMEWORK

**Dr. Shubha Rao V[1], Dr. Sneha Girish[2], Ananthanarayanan Venkitakrishnan[3], Arjun A[4], Arjun P Chandra[5]**

[1]Department of Information Science and Engineering BMS College of Engineering, Bengaluru, India

e-mail: shubha.ise@bmsce.ac.in

[2]Department of MCA

K.S Institute of Technology, Bengaluru, India e-mail: snehagirish@ksit.edu.in

[3,4,5]Department of Information Science and Engineering BMS College of Engineering, Bengaluru, India

e-mail: ananthanarayanan.is22@bmsce.ac.in e-mail: arjun.is22@bmsce.ac.in

e-mail: arjunpc.is22@bmsce.ac.in

**Abstract**

Mobile Cloud Offloading (MCO) addresses resource limitations of mobile devices by migrating compute-intensive tasks to powerful remote servers. This paper presents a predictive MCO framework that incorporates a machine learning-based decision engine to intelligently select the optimal execution environment among local, edge, and cloud resources. The framework is evalu- ated using diverse workloads including matrix multiplication and image processing. Experimental results demonstrate that the ML-driven predictive approach consistently achieves lower execu- tion latency compared to static and reactive offloading strategies, validating the effectiveness of context-aware, proactive decision-making in mobile computing environments.

**Math. Subj. Classification 2020.** 68M20, 68T05, 68U05.

**Key Words and Phrases.** Mobile cloud offloading, edge computing, machine learning, latency opti- mization, mobile systems.

## 1 Introduction

The advancement of mobile computing has enabled sophisticated applications that demand substantial computational resources. Despite improvements in mobile hardware, devices continue to face con- straints in processing power, memory, and battery life. Mobile Cloud Offloading (MCO) addresses these limitations by transferring computationally demanding tasks to more powerful remote servers.

The effectiveness of offloading depends on network conditions, task complexity, and device state. Traditional approaches employ static or reactive mechanisms that often fail to achieve optimal per- formance in dynamic environments. This work develops a predictive MCO framework leveraging machine learning to make proactive, context-aware offloading decisions, selecting optimally among

Received: Aug 06, 2025

International Journal of Applied Mathematics
Volume 39 No. 1s, 2026
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

local, edge, and cloud resources. The implementation deploys the server component on both a Mi- crosoft Azure virtual machine (representing cloud infrastructure) and a laptop (representing edge infrastructure), demonstrating a hybrid edge-cloud architecture.

## 2 Objectives

The primary objectives of this research are: (1) to analyze limitations of existing offloading strategies,

(2) to design an ML-driven predictive offloading model, (3) to minimize execution latency through intelligent location selection, (4) to evaluate performance across local, edge, and cloud environments, and (5) to provide an extensible framework for future research.

## 3 Literature Review

Computation offloading has been extensively studied. Kumar et al. [1] provided a comprehensive survey of offloading techniques, emphasizing remote execution for overcoming mobile limitations.

The emergence of Mobile Edge Computing (MEC) represents a significant evolution. Mach and Běcvǎř [2] introduced MEC to reduce latency by positioning computational resources closer to mobile users. Dong et al. [3] investigated task offloading strategies for mobile edge computing, while Feng et al. [4] examined computation offloading techniques in MEC networks, categorizing approaches by offloading objectives and techniques.

Machine learning-based approaches have gained prominence due to their ability to adapt to het- erogeneous environments. Boukerche et al. [5] explored sustainable offloading strategies considering performance and energy consumption. The present work builds upon these advances by implementing and validating a predictive ML-based decision engine through actual execution in a hybrid cloud-edge environment.

## 4 Methodology

### 4.1 System Architecture

The proposed mobile cloud offloading framework is designed as a distributed system comprising three primary components, each serving distinct functions in the task execution pipeline:

- **Mobile Device**: Serves as the task initiation point, responsible for generating computation tasks such as matrix multiplication and image processing operations. The mobile device collects contextual information including network conditions, battery status, and device characteristics, which are used to inform offloading decisions.

- **Edge Server**: Comprises laptop-based servers deployed in close network proximity to mobile devices, providing low-latency computation capabilities. Edge servers offer intermediate com- putational resources with reduced network overhead compared to cloud infrastructure, making them suitable for tasks requiring rapid response times.

- **Cloud Server**: Utilizes Microsoft Azure cloud services to provide high computational capac- ity with increased latency. Cloud resources offer virtually unlimited scalability and processing power, making them appropriate for computationally intensive tasks where latency

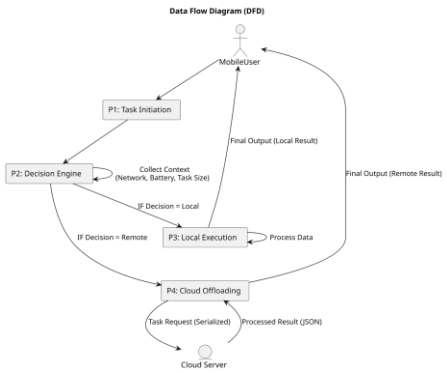
Received: Aug 06, 2025

**International Journal of Applied Mathematics**
**Volume 39 No. 1s, 2026**
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

is less criti- cal.

Figure 1 depicts the data flow between the three primary components. The mobile device acts as the task initiator and coordinator, while edge and cloud servers provide computational resources based on the offloading decision determined by the ML-based decision engine.

Figure 1: Data flow diagram showing the interaction between mobile device, edge server, and cloud server.




## 4.2 Offloading Decision Model

The core innovation of the proposed framework lies in its intelligent offloading decision model, which employs a machine learning classifier to predict the optimal execution location for each task. The model incorporates multiple input features that collectively characterize the execution context:

- **Task Characteristics**: Size and complexity of the computation to be performed
- **Network Conditions**: Current latency, bandwidth availability, and network type (WiFi, cellu- lar)
- **Device State**: Battery level, charging status, and current device load
- **Server Status**: CPU utilization and memory availability on edge and cloud servers

The primary objective of the decision model is to minimize total execution latency, which encom- passes both computation time and data transfer overhead. By considering these multifaceted factors, the model can make nuanced decisions that balance competing priorities and adapt to changing envi- ronmental conditions.

## 4.3 Workflow

The operational workflow of the proposed framework follows a systematic five-step process:

1. **Task Generation**: The mobile application generates a computation task based on user interac- tion or automated processes.
2. **Context Collection**: The framework collects current network metrics, device state information, and server load data to establish the execution context.
3. **Latency Prediction**: The ML model estimates expected execution latency for local, edge, and cloud execution based on the collected context and task characteristics.
4. **Location Selection**: The execution environment with the minimal predicted latency is

Received: Aug 06, 2025

International Journal of Applied Mathematics
Volume 39 No. 1s, 2026
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

selected, and the task is dispatched accordingly.

5. **Task Execution**: The task is executed either locally on the device or remotely on the selected server, with results returned to the mobile application.

# 5 Machine Learning Decision Model

The proposed machine learning (ML) decision model operates in two distinct phases: offline train- ing and real-time inference. This two-phase architecture enables the model to learn from historical execution data while maintaining the ability to make rapid decisions during runtime operation.

## 5.1 Model Selection and Justification

A Random Forest classifier was selected as the core predictive algorithm due to several advantageous properties. Random Forest ensembles are known for their robustness against overfitting, ability to handle heterogeneous feature sets without extensive preprocessing, and capacity to model complex non-linear relationships between input features and target variables. Additionally, Random Forest models provide feature importance rankings, enabling interpretation of which factors most signifi- cantly influence offloading decisions.

## 5.2 Training Data and Features

The model is trained using comprehensive benchmark datasets that capture diverse execution scenar- ios. These datasets contain task metadata, device context information, and observed execution times for both local and remote execution environments. The training process considers the following key features:

- **Task Attributes**: Type of computation (matrix multiplication, image processing), problem size, and computational complexity
- **Network Characteristics**: Network type (WiFi, 4G, 5G), measured latency, and available bandwidth
- **Device Context**: Device model identifier, current battery level, charging status, and CPU uti- lization
- **Server State**: CPU load percentage and memory utilization on available edge and cloud servers
- **Performance Metrics**: Historical execution times for local and remote execution under similar conditions

Figure 2 presents a representative sample of the training dataset structure. The dataset includes categorical features such as task name and network type, numerical features including latency, matrix size, and server load, and target variables representing observed execution times for local and remote execution environments. This comprehensive data enables the model to learn complex patterns and make accurate offloading decisions across diverse operational scenarios.

## 5.3 Decision Mechanism

During operation, the model predicts whether offloading a task will reduce total execution latency compared to local execution. The prediction is based on the current runtime context, encompassing network conditions, device state, and server availability. By learning from

Received: Aug 06, 2025

**International Journal of Applied Mathematics**
**Volume 39 No. 1s, 2026**
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

historical execution data, the model can anticipate the performance implications of offloading decisions and select the execution environment most likely to minimize latency.

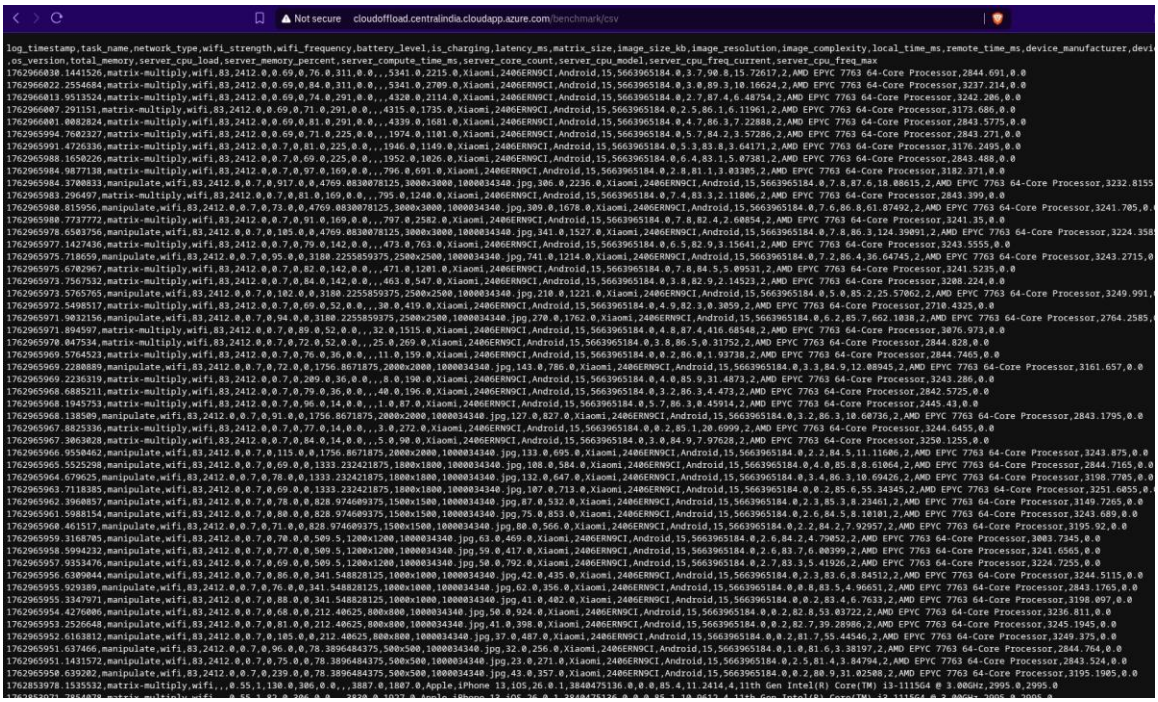



Figure 2: Sample dataset showing task metadata, network conditions, and execution times used for training the ML model.

# 6 Algorithms

This section presents the formal algorithms that implement the ML-based offloading decision frame- work. Algorithm 1 describes the offline training phase in which the predictive model is constructed from historical execution data, while Algorithm 2 details the server-side inference procedure used to make real-time offloading decisions during task execution.

## 6.1 Offline Training Phase

The offline training process constructs a predictive model by learning patterns from historical ex- ecution data. The primary objective is to establish a robust mapping between contextual features (task characteristics, network conditions, device state, server load) and optimal execution decisions. The training process begins with loading a comprehensive benchmark dataset containing records of previous task executions under diverse conditions.

The target variable for classification is derived by comparing local and remote execution times: if remote execution completes faster than local execution, the target is set to 1 (offload recommended), otherwise 0 (local execution recommended). Feature engineering involves one-hot encoding of cate- gorical variables (task type, network type, device model) and standardization of numerical features to ensure compatibility with the Random Forest algorithm.

The dataset is partitioned into training and testing subsets to enable model validation. A Ran- dom Forest classifier is initialized with appropriate hyperparameters and trained on the training data. Model performance is evaluated on the held-out test set using metrics such as

Received: Aug 06, 2025

International Journal of Applied Mathematics
Volume 39 No. 1s, 2026
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

accuracy, precision, recall, and F1-score. Finally, the trained model, along with the fitted scaler and feature column meta- data, are serialized and persisted for subsequent use in real-time inference.

## 6.2 Real-Time Inference Phase

During runtime operation, the trained model is deployed to make real-time offloading decisions based on current task and system context. When a mobile device submits a task for execution, the request

**Algorithm 1** Training and Saving the Random Forest Decision Model

**Require:** Path to benchmark CSV file *data path*

**Ensure:** Saved artifacts: scaler.pkl, random forest model.pkl, feature columns.pkl

1: Load dataset $D \leftarrow$ read csv(*data path*)

2: Create binary target variable:

$y \leftarrow$ I[remote time ms < local time ms]

3: Define feature set

$F = \{$task name, network type, device model, battery level, is charging, latency ms, matrix size, image size

4: One-hot encode categorical features in $F$ to obtain feature matrix $X$

5: Split $(X,y)$ into training and testing datasets

6: Fit scaler on numerical features and transform data

7: Initialize Random Forest classifier

8: Train the model on the training dataset

9: Evaluate model performance on the test dataset

10: Save trained model, scaler, and feature column metadata

11: **return** success

includes task metadata and current device state information. The server-side inference process loads the persisted model artifacts and reconstructs the feature representation required for prediction.

Categorical features are encoded using the same one-hot encoding scheme applied during train- ing, ensuring feature consistency. The numerical features are transformed using the saved scaler to maintain the same scale as the training data. The model then computes the probability that offloading will result in lower execution latency. This probability is compared against a predefined threshold $\tau$; if the probability exceeds the threshold, the system recommends offloading, otherwise local execution is advised. The decision is returned to the mobile device along with the confidence score, providing transparency into the decision-making process.

Received: Aug 06, 2025

**International Journal of Applied Mathematics**
**Volume 39 No. 1s, 2026**
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

**Algorithm 2** Server-side Inference Using Persisted Artifacts

**Require:** Incoming request containing task metadata and runtime context $R$

**Ensure:** Decision: OFFLOAD or LOCAL, with confidence score

1: Load persisted artifacts: trained model, scaler, and feature columns

2: Construct feature dictionary from request $R$

3: One-hot encode categorical fields

4: Create feature vector $x$

5: Apply scaling transformation to $x$ 6: Compute predicted probability $p$ 7: **if** $p \geq \tau$ **then**

8: **return** OFFLOAD with confidence $p$

9: **else**

10: **return** LOCAL with confidence $(1 - p)$

11: **end if**

## 7 Results and Discussion

The proposed mobile cloud offloading framework was evaluated through comprehensive experimen- tation comparing four distinct execution strategies: local execution on the mobile device, cloud-only

offloading, dynamic offloading based on current conditions, and the proposed ML-driven predictive offloading approach. The experiments were conducted using computationally intensive workloads in- cluding matrix multiplication operations of varying sizes and image processing tasks such as flipping and grayscale conversion.

### 7.1 Performance Analysis by Task Type

Figure 3 presents the average execution times for local versus remote execution across different task types. The results indicate that image manipulation tasks execute faster locally due to network transfer overhead for the flip operation, while matrix multiplication tasks show varying performance depend- ing on size. Smaller matrix operations perform better locally when network overhead is considered, whereas larger matrices benefit from remote execution.

Received: Aug 06, 2025

**International Journal of Applied Mathematics**
**Volume 39 No. 1s, 2026**
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

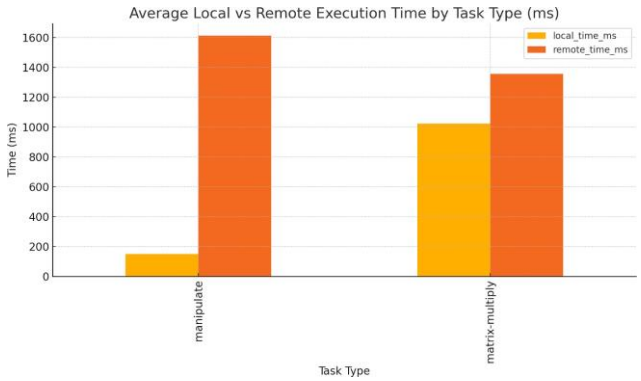



Figure 3: Average local vs. remote execution time by task type.

## 7.2 Matrix Multiplication Performance

Figure 4 illustrates the execution time comparison between local and remote execution for matrix multiplication tasks across varying matrix sizes. As matrix size increases, remote execution becomes increasingly advantageous due to the superior computational capabilities of cloud and edge servers. The crossover point occurs at approximately 200×200 matrices, beyond which offloading consistently outperforms local execution.

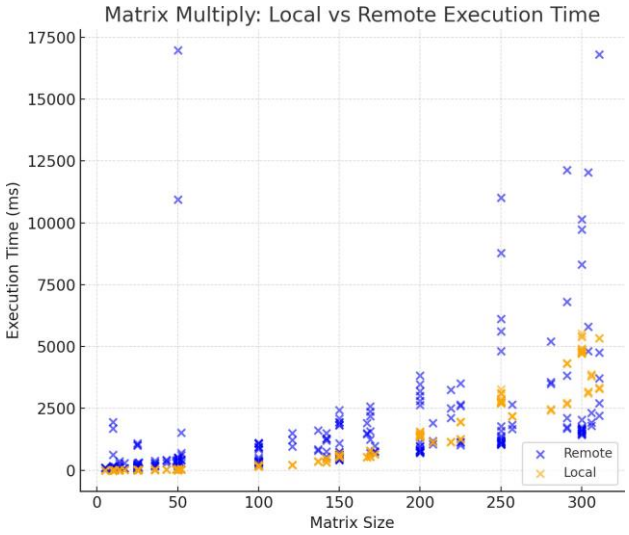



Figure 4: Matrix multiplication execution time: local vs. remote by matrix size.

## 7.3 Image Processing Performance

Figure 5 demonstrates the performance characteristics for image flip tasks across different image sizes. The results show that remote execution time increases compared to local execution as image size increases. This behavior occurs because the image flip operation has relatively low computational complexity, making network transfer overhead the dominant

factor. For computationally intensive image processing algorithms, offloading may provide benefits despite network costs, but for simple operations like image flipping, local execution is more efficient for larger images.

Received: Aug 06, 2025

**International Journal of Applied Mathematics**
**Volume 39 No. 1s, 2026**
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

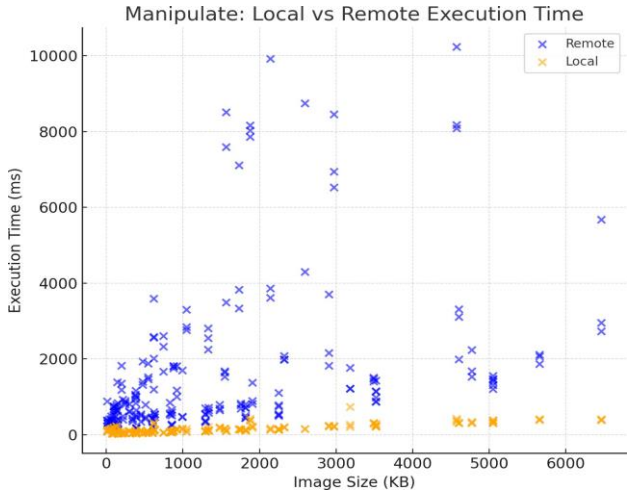



Figure 5: Image flip execution time: local vs. remote by image size.

## 7.4 Network Type Impact

Figure 6 examines the impact of network type on execution latency. The results show that WiFi connections provide significantly lower latency for offloaded tasks compared to cellular networks. However, the ML-driven predictive approach successfully adapts to network conditions, making op- timal decisions regardless of the connection type.

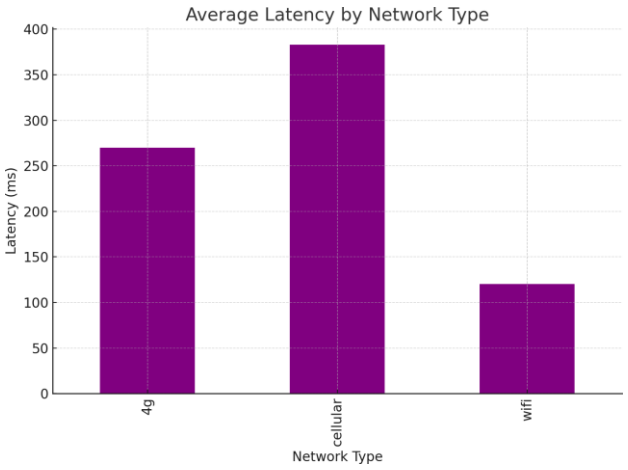



Figure 6: Execution latency by network type for different offloading strategies.

## 7.5 Quantitative Comparison

Table 1 presents a quantitative comparison of execution latency across the four strategies for tasks of varying sizes. The ML-driven predictive approach consistently achieves the lowest latency across all task categories.

| Task Size | Local | Cloud | Dynamic | ML Predictive |
|---|---|---|---|---|
| Small | 60 | 100 | 58 | 55 |
| Medium | 250 | 180 | 160 | 150 |

Received: Aug 06, 2025

Volume 39 No. 1s, 2026
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

| | | | | |
|---|---|---|---|---|
| Large | 600 | 450 | 400 | 380 |

Table 1: Execution latency comparison (ms) across different offloading strategies for various task sizes.

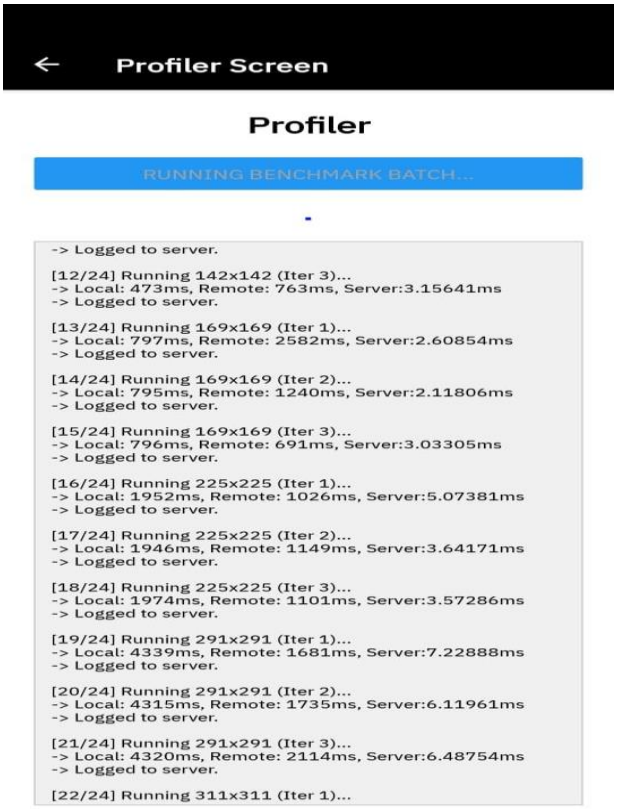
The experimental results demonstrate that the ML-driven predictive approach achieves perfor- mance improvements of 8-38% compared to individual strategies. By considering task characteris- tics, network conditions, and device state, the model consistently selects the execution environment that minimizes total latency, validating the effectiveness of the proposed framework.

## 7.6 Implementation

Figure 7 shows the profiler interface of the mobile application used for task configuration and execu- tion monitoring. The interface allows specification of task parameters such as matrix dimensions and image characteristics, enabling systematic evaluation across different computational loads.

The framework is implemented using a multi-technology stack optimized for each component's requirements. Python serves as the primary language for implementing the offloading logic and ma- chine learning model, leveraging libraries such as scikit-learn for the Random Forest classifier and Flask for the REST API server. The mobile client application is developed using React Native with JavaScript, handling task orchestration, server communication, and result presentation. Performance visualization and analysis are conducted using Matplotlib and Seaborn for generating graphs that depict latency comparisons and execution patterns across different strategies.

Figure 7: Profiler interface for task configuration and execution monitoring.




Received: Aug 06, 2025

**International Journal of Applied Mathematics**
**Volume 39 No. 1s, 2026**
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

## 8 Conclusion

This paper presented a predictive mobile cloud offloading framework that leverages machine learn- ing to make intelligent, context-aware decisions regarding task execution location. The proposed approach addresses limitations of traditional static and reactive offloading strategies by incorporating predictive analysis of task characteristics, network conditions, device state, and server availability.

Experimental evaluation demonstrates that the ML-driven predictive approach consistently out- performs conventional offloading strategies, achieving performance improvements of 8-38% across diverse task types and sizes. The hybrid architecture combining edge and cloud resources provides a flexible and scalable solution that adapts to varying computational requirements and network condi- tions.

## 9 Future Work

Future research directions include integration of deep reinforcement learning for adaptive policy learning, evaluation of multi-user scalability, implementation of enhanced security mechanisms in- cluding encrypted offloading, development of energy-aware offloading decisions, and extension to support additional execution environments such as fog computing and peer-to-peer networks.

### Acknowledgements

The authors would like to acknowledge their institution and mentors for guidance and support.

### References

[1] K. Kumar et al., A survey of computation offloading for mobile systems, *Mobile Networks and Applications*, 2013.

[2] P. Mach and Z. B ě c v ǎ ř , Mobile edge computing, *IEEE Communications Surveys & Tutorials*, 2017.

[3] S. Dong et al., Task offloading strategies for mobile edge computing, 2024.

[4] C. Feng et al., Computation offloading in mobile edge computing networks, *Journal of Network and Computer Applications*, 2022.

[5] A. Boukerche et al., Sustainable offloading in mobile cloud computing, 2019.

Received: Aug 06, 2025