Optimizing Energy Efficiency and Latency in Mobile-Edge-Cloud Systems via PPO, LSTM Caching, and DVFS

MSc Research Project
Cloud Computing

Deekshiya Seenivasagan
Student ID: x23268336

School of Computing
National College of Ireland

Supervisor: prof.Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Deekshiya Seenivasagan
Student ID:	x23268336
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	prof.Aqeel Kazmi
Submission Due Date:	15/09/2025
Project Title:	Optimizing Energy Efficiency and Latency in Mobile-Edge-Cloud Systems via PPO, LSTM Caching, and DVFS
Word Count:	8405
Page Count:	21

Signature:	Deekshiya Seenivasagan
Date:	15th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Optimizing Energy Efficiency and Latency in Mobile-Edge-Cloud Systems via PPO, LSTM Caching, and DVFS

Deekshiya Seenivasagan
x23268336

Abstract

The increasing demand for computationally intensive and latency-sensitive applications on mobile devices presents significant challenges in energy efficiency, battery life, and speed. This research project addresses these challenges through smart decisions made by the AI-driven framework for energy-efficient and latency-aware task offloading in mobile-edge-cloud (MEC) environments. The solution integrates three key components: Proximal Policy Optimization (PPO) which is a reinforcement learning model for intelligent task offloading decisions, Long Short-Term Memory (LSTM) network for proactive edge caching, and Dynamic Voltage and Frequency Scaling (DVFS) for adaptive power management. The framework is evaluated and implemented in a simulated environment using CloudSimPlus, with the Python-trained machine learning models has been integrated into the Java based simulator via ONNX Runtime. Various experiments were conducted across eight scenarios in an organized manner by enabling and disabling each optimization component. The results demonstrated that the integrated approach (PPO+LSTM+DVFS) achieves significant improvements in energy consumption, device battery lifetime, and overall task latency when compared to heuristic-based and other baseline approaches. The overall findings demonstrate that combining intelligent offloading, predictive caching, and dynamic power management provides better results for modern MEC systems. This study provides both theoretical understanding and practical guidance for designing intelligent and resource-efficient MEC infrastructures.

1 Introduction

The fast growth of computationally intensive and latency-sensitive applications on mobile devices such as augmented reality, video streaming, and real-time analytics has fundamentally transformed how users interact with digital services. However, this evolution presents significant challenges in energy efficiency, device battery lifetime, and service responsiveness. As mobile devices are inherently resource-constrained, traditional mobile cloud computing (MCC) architectures which offloads heavy workloads to cloud servers has become a key solution and offers greater computational capabilities, but this can lead to high network latency due to constant cloud interactions (Mao et al., 2017; Lin et al., 2014). To address these limitation Mobile Edge Computing (MEC) has emerged as a key solution for extending cloud capabilities to the network edge and enabling computation and caching closer to end users. By leveraging edge nodes for computation and caching, MEC reduces both latency and network load, enhancing the user experience for delay-critical applications (Mao et al., 2017; Zhang et al., 2024; Liang et al., 2023). However, the dynamic nature of mobile environments which has fluctuating workloads, heterogeneous resources, and unpredictable user demands requires intelligent, adaptive strategies for resource management.

Recent research highlights the importance of artificial intelligence (AI), particularly reinforcement learning (RL) and deep learning models can help make better decisions about where to run tasks (smarter task offloading), how to use resources (resource allocation), and caching policies in MEC scenerios (Zhou et al., 2024; Zhang et al., 2024; Demirci & Korcak, 2024). Algorithms such as Proximal Policy Optimization (PPO) have shown efficacy in making real-time offloading decisions that optimize energy usage and task latency delay (Zhang et al., 2024; Ma et al., 2025). In parallel, Long Short-Term Memory (LSTM) neural networks enable proactive and predictive edge caching by forecasting content popularity, thereby reducing redundant network traffic (Demirci & Korcak, 2024; Zong et al., 2023). In addition to intelligent offloading and caching, Dynamic Voltage and Frequency Scaling (DVFS) is a widely adopted technique for adaptive power management, by exploring the CMOS dynamic power realtion (V^2f) to achieve energy savings without compromising performance (Li et al., 2023; Florence et al. (2016)). However, these techniques offloading, caching and DVFS were explored in isolation or in limited pairs. This motivates an integrated approach that coordinates learning-based scheduling with caching and DVFS to improve end-to-end efficiency in realistic mobile-edge-cloud environments (Zhou et al., 2024).

1.1 Research Objective

Motivated by these challenges, this research addresses the following research questions: "How can the integration of Proximal Policy Optimization (PPO) for task scheduling, Long Short-Term Memory (LSTM)-based predictive caching, and Dynamic Voltage and Frequency Scaling (DVFS) be leveraged to optimize energy efficiency and latency in mobile-edge-cloud systems?"

- The main objective of this research is to develop and implement an integrated framework that combines Proximal Policy Optimization (PPO) for intelligent task scheduling, Long Short-Term Memory (LSTM) for proactive edge caching, and Dynamic Voltage and Frequency Scaling (DVFS) for adaptive, real-time power management for mobile-edge-cloud systems.

- Design and conduct simulation-based experiments using realistic synthetic workloads to evaluate the proposed framework.

- Implement and simulate the unified framework within the CloudSim Plus environment, integrating Python-trained machine learning models with Java-based simulation via ONNX Runtime.

- Evaluate the effectiveness of the proposed framework by benchmarking its performance using metrics such as energy consumption, task latency, and device battery life against conventional baseline approaches through scenario-based simulations.

1.2 Structure of the report

This report is organized as follows: Introduction which outlines the motivation and objectives, Related work reviews existing research and identifies gaps, Methodology describes the approach, data generation, and machine learning models used in the research, Design Specification will provide the details of system architecture, component specifications, and workflow of the proposed framework, Implementation will explain the development, integration, and simulation setup, Evaluation analyzes the experimental results and compares performance across different scenarios, Conclusion and Future Work will Summarize the key findings and suggests directions for future research.

2 Related Work

Mobile Cloud Computing (MCC) has become an essential paradigm that combines the computing capabilities of mobile devices with the vast resources of cloud servers. This setup allows mobile devices to offload heavy computational tasks, which helps conserve battery life and boost performance. Mobile Cloud Computing (MCC) has lots of benefits such as more processing power, better performance on mobile devices but it also comes with challenges such as high network latency and increased dependency on the cloud infrastructures. To tackle these problems, researchers have come up with a range of strategies. The thing is, most of these approaches are explored separately, not as part of a bigger, unified solution. Key among these are intelligent task scheduling, edge caching, and Dynamic Voltage and Frequency Scaling (DVFS). With the evolution of computing model, Multi-access Edge Computing (MEC) has been introduced, having computational resources closer to mobile users at the network edge. This advancement enables "Mobile-Edge-Cloud" systems, which combine the strengths of mobile devices, edge nodes, and cloud servers. This literature review examines various important studies that focus on these individual methods, and it highlights the importance of integrating them into a unified framework

2.1 Task Scheduling and Offloading in Mobile, Edge and cloud Environments

Task scheduling plays a important role in managing computational workloads across mobile, edge, and cloud layers. Effectively managing how tasks are distributed across mobile devices, edge nodes, and cloud servers is important for optimizing both performance and energy usage. Zhang et al. (2024) proposed a solution based on reinforcement learning, utilizing a method called Proximal Policy Optimization (PPO). Their approach enables dynamic task allocation within Multi-access Edge Computing (MEC) environments by continuously adapting to changing system conditions. They built a real MEC testbeds and trained on parameters such as CPU utilization, transmission delay, task execution time, task count. PPO performed well when compared to DQN/A2C on delay and energy. This aligns closely with the direction of the current research, reinforcing PPO as a solid method for intelligent taskscheduling. To evaluate learning based scheduling under controlled conditions, we follow common practice and use synthetic task traces in simulation, as done in PPO-based offloading/scheduling studies Fu et al. (2024) Xiong et al. (2022) they proposed a PPO-based RL offloading algorithm which models edge server and link energy. They achieved 22.69 percent average energy savings over baselines..

Similarly, Lin et al. (2014) conducted a research on energy and performance aware task scheduling and achieved 3.1x reduction in energy consumption. Both the studies are less effective because rely on the static or rule based decision making. In addition to these works, several studies have examined energy-aware scheduling strategies for cloud and real-time systems. Ismail and Fardoun (2017) proposed the EATS framework, and showed how workload type and startup/shutdown behaviour will impact energy and also they did not report about energy savings from EATS. Furthermore, Bambagini et al. (2016)provided a comprehensive survey of energy-aware scheduling techniques in real-time embedded systems. They looked at many types of scheduling, like rule-based and adaptive methods, and showed that using smart, flexible scheduling is especially important for saving energy when working with strict deadlines.Building on this, the study by Chandrasiri and Meedeniya (2025) further illustrates the practical benefits of PPO in cloud-based environments. The researchers showed that by refining scheduling decisions through a tailored reward system, PPO could help reduce both total execution time and energy consumption. The demonstrated effectiveness of PPO across different platforms ranging from edge to cloud environments strongly supports its adoption within the current study's framework. Fu et al. (2024) and Ma and Tian (2025) has used the PPO applications in UAV-based and collaborative MEC systems respectively, highlighting the PPO capability in balancing latency and energy demands through intelligent scheduling, strongly aligning with the current research objective.

2.2 Predictive Edge Caching with Deep Learning Models

To address latency caused by frequent data retrieval from remote cloud servers, edge caching has proven to be a valuable solution. Traditional approaches like the Least Recently Used (LRU) algorithm, while straightforward, often struggle to adapt to rapid changes in user behavior. In response to this limitation, Demirci and Korcak (2024) introduced a forward-thinking caching method powered by Long Short-Term Memory (LSTM) neural networks. They reduced latency and improved the cache hit rates by 29 percent than LRU approximately. Their system was capable of accurately forecasting user content requests and proactively storing that data at edge locations. When generating the synthetic access logs for caching experiments, we assume Zipf content popularity which is a standard and supported model in edge caching Breslau et al. (1999). In a related study, Liang et al. (2023) examined the combined optimization of task offloading and content caching. Their findings indicated that coordinating these two processes can give noticeable improvements in energy consumption. However, their framework did not incorporate any form of predictive modeling like LSTM, which could limit its responsiveness under changing user demand. Addressing this limitation, Zong et al. (2023) proposed an ensemble learning method which can dynamically predict content popularity trends, reinforcing the advantage of intelligent caching strategies in MEC.Wu et al. (2021) has provided a comprehensive overview of caching strategies, which highlights the static approaches vs learning based approaches. While these static methods are easy to implement, the authors noted that they do not perform well under dynamic and unpredictable content request patterns. While deep learning is becoming more popular for predicting what to cache, some studies still focus on simple and practical caching methods at the network edge. For instance, Azeem et al. (2022) looked at how to set up and manage caches for mobile devices in real-world networks. They found that effective caching at the edge can make things faster and reduce the amount of data sent over the main network, even when devices have limited resources. This suggests that combining advanced prediction models like LSTM with basic caching methods can make systems work even better.

2.3 Dynamic Voltage and Frequency Scaling (DVFS) for Energy Efficiency

At the system level, Dynamic Voltage and Frequency Scaling (DVFS) is now a well-known method for cutting down power consumption. It works by adjust the CPU's voltage and frequency depending on how heavy the workload is. This way, devices keep up performance while saving energy at the same time. Li et al. (2023) explored this idea in mobile edge computing. They combined DVFS with task offloading which aims to reduce the energy use. But here's the thing, their setup didn't use any kind of real-time learning. No adaptive logic either.Then came Liu et al. (2025). Their DVFS model was built for complex, multicore systems. It predicted workloads and prioritized tasks. Adjusted processor settings on the fly. More than 20 percent energy saved and hardly any missed deadlines. They were efficient, but even this model didn't deal with task offloading or edge caching. On the other hand, HajiKhodaverdian et al. (2023) added another layer. They brought reinforcement learning into the mix combined it with DVFS on mobile devices. The system learned over time. CPU settings were adjusted accordingly. But again, no caching. No prediction of content needs and no coordination across edge or cloud. Panda et al. (2023) further highlighted this integration, demonstrating deep reinforcement learning coupled with DVFS for energy-efficient computation offloading in time-critical IoT applications. Their results align with the proposed research methodology which highlights the benefits of combining DVFS with reinforcement learning. Lin et al. (2015) proposed a DVFS-based scheduling framework for mobile cloud computing environments. Their system dynamically adjusts voltage and frequency settings according to workload demands, resulting in minimized energy consumption while still meeting real-time processing requirements. Collectively, these works illustrate the practical effectiveness of DVFS as a core energy-saving mechanism, particularly when combined with intelligent task scheduling and adaptive system controls.In this research project, DVFS isn't working alone. It's tightly connected to the PPO-based scheduler. When the scheduler decides a task should run locally, DVFS steps in tuning the CPU based on the urgency and workload. Together, they save power and they don't slow things down. It's a more balanced, smarter system overall.

2.4 Comprehensive Perspectives on AI-Based Scheduling

To support these developments, Zhou et al. (2024) provides a broad analysis of various deep reinforcement learning (DRL) methods like PPO, DQN, and A3C. It categorizes different models by application context, performance metrics, and deployment challenges. Most importantly, it identifies the lack of comprehensive frameworks combining DRL with other policies and building flexible scheduling system, thus highlighting a critical research gap. This review strongly supports the need for an integrated approach and offers methodological guidance for combining DRL with systems-level optimization techniques. Additionally, Mao et al. (2017) provided detailed surveys and frameworks which highlights the importance of communication efficiency and load balancing in MEC. Complementing these Zhang et al. (2020) has presented a energy-aware scheduling framework for edge computing that explores energy Internet and energy harvesting for reducing energy use. These studies helps to justify further, that combining intelligent scheduling, caching, and energy management strategies to optimize system performance.

2.5 Identified Research Gap and Novel Contribution

Although each of the reviewed papers has demonstrated substantial benefits in improving energy efficiency, reducing latency, or enhancing resource allocation, they all focus on isolated or dual-component solutions. None of the studies integrates all three strategies PPO-based task scheduling, LSTM-based edge caching, and adaptive DVFS into a unified, real-time decision framework for mobile cloud computing environments. The current research addresses this clear gap by proposing a novel system that combines PPO-based task scheduling, LSTM-based predictive caching, and DVFS-based power management. This integrated approach highlights the strength of each methodology to dynamically optimize energy consumption, reduce latency, and enhance device battery levels in MEC environments. The resulting integrated framework is expected to demonstrate superior performance in energy savings, reduced latency, and overall system efficiency, thereby setting a new benchmark for future developments in this area.

3 Methodology

3.1 Synthetic Data Generation

This research generates two primary synthetic datasets: a Task Workload Dataset for PPO training and Content Access Logs for LSTM-based caching prediction which helps in evaluating the AI-driven scheduling and caching models in a realistic yet controlled environment

3.1.1 Task Workload Dataset

The task dataset was synthetically generated using Python. A total of 1000 tasks were created that resembles real mobile applications. Each task has several attributes such as:

- Arrival Time represents task arrival time in ms
- CPU Requirement represents processing power from 0.5 to 4.0 GHz
- Memory Usage represents memory required between 256 MB and 2048 MB.
- Execution time represents estimated time (10–300 ms) to complete the task.
- Deadline represents time limit for task completion between 100 to 200 ms
- Priority Level represents task priority from 1 (low) to 5 (high),
- Content ID: Links the task to certain content, used for caching decisions.

To incorporate caching logic into the scheduling process, each task was linked to a content ID, simulating dependencies on content files (e.g., video, document) that might be cached at the edge. This dataset is stored in CSV format and is used to train the PPO model and evaluate task offloading decisions within the simulation.

3.1.2 Content Access Logs

The second dataset simulates user interactions with content over time. It stimulates 1000 hours of system activity involving 100 users and 300 content items. In each hour, 10 access events are simulated. In real life certain videos or files will be accessed more often, certain content will be more popular, in order to copy this behavior, a method called a Zipf distribution was applied, where a few contents receive significantly more requests than others. To make even more realistic, the top 50 content items receive a popularity boost, ensuring even more frequent selection during log generation. For each log entry, a timestamp, user ID, and content ID are recorded. All this data is stored in a CSV format, which contains over 10,000 access records. Each row represents one content request and access count of 1, which can be used to calculate how many times each content is accessed per hour. This dataset is later processed to train the LSTM model, which will learn from these pattern and predict which content will be popular in the next hour. This helps improve caching at the edge, saving time and reducing cloud usage.

3.2 Machine Learning Model Development

3.2.1 Proximal Policy Optimization (PPO) for Task Offloading

The core scheduling logic in this research is developed using Proximal Policy Optimization (PPO) model. PPO is a reinforcement learning algorithm known for its stability and performance in learning tasks continuously. In this research PPO was used to train the agent capable of deciding where to execute the tasks whether to execute it locally on the device, offload to the nearby edge server, or send it to the cloud. The PPO model was developed using Stable-Baselines3 library in python and they are trained within a custom Gym-compatible simulation environment.

- Observation space: For the agent includes normalized task features, system state (device battery level, bandwidth availability) and caching status.
- The action space: Comprises three discrete choices: (0 = execute locally, 1 = offload to edge, 2 = offload to cloud).
- Reward: The reward function is carefully shaped to penalize high energy and latency, while providing bonuses for meeting deadlines, utilizing cache, and avoiding actions that deplete battery or bandwidth.

The model was trained over 500,000 steps. Each PPO agent processes 50 tasks, updating the environment state after each decision. The training objective was to minimize energy and latency. Once trained, the model is exported to ONNX format using PyTorch's export utilities. The model was later used within the Java simulation to guide real-time task allocation.

3.2.2 Long Short-Term Memory (LSTM) for Predictive Edge Caching

Edge caching is enhanced through Long Short-Term Memory (LSTM) networks trained to predict the content popularity. The LSTM model was developed using Keras (TensorFlow backend). The LSTM model takes past 24 hours access logs as input and predicts the content for the next hour. The architecture includes: LSTM layer with 128 units, followed by a dropout layer (0.3 dropout rate), and a dense layer with ReLU activation. The LSTM was trained using an 80/20 training-validation split and in order to prevent overfitting, early stopping was enabled. Once the model was trained, it was exported using tf2onnx. The model was later used within the Java simulation. During simulation, the cache was updated every simulated hour using the LSTM predictions. The top-30 predicted contents will be stored in the edge cache for that hour.

3.3 Data Collection and Metrics

Metrics are collected during the simulation runs systematically, metrics includes latency in milliseconds , Energy consumption in Joules and battery consumption (battery level is measured after the task execution). Each scenario generates logs and the results are stored in CSV file for post simulation analysis. This research evaluates eight scenarios across combinations of LSTM caching (on/off), PPO offloading (on/off), and DVFS control (on/off), under different workload intensities. For each cloudlet, it collects latency in ms, energy consumed in Joule and battery level in CSV file and later imported into Python for analysis using Pandas and Matplotlib. The data were then used to create performance comparison plots across eight experimental scenarios.

4 Design Specification

4.1 System Architecture

The proposed framework follows a three-tier hierarchical architecture comprising of local (mobile device), edge servers, and cloud data centers. Each layer, represents a real-world computational resource and plays a very important role in processing the tasks and making decisions to ensure efficiency:

- Mobile Device Layer: Mobile device layer represents the typical smartphones. These devices are resource-constrained in terms of CPU capacity, memory, and battery life. The Mobile device layer generate tasks (cloudlets) and are more sensitive to battery drain and computation delays. These tasks can be executed locally or off-loaded based on the decision made by the PPO model.

- Edge Server Layer: Edge server layer simulates a more powerful server and they are located closer to the user. The edge can process the tasks offloaded from mobile devices and also maintains a cache, updated by AI predictions for rapid content access and process the tasks. These edge layer can process and execute offloaded tasks with lower latency than the cloud and energy efficient than mobile device.

- Cloud Datacenter Layer: Centralized servers with high computing power and unlimited storage but higher network latency. They are capable of executing large workloads but incurs higher network latency.

Each layer is represented as a separate Host object in CloudSim Plus, with resources reflecting their real-world values. These hosts are virtualized via VMs with corresponding capabilities. All these layers interconnected through virtual network links. The mobile devices produce tasks and these tasks are then scheduled either locally, to the edge, or to the cloud, based on AI-driven decision-making logic. This setup provides realistic evaluation of how task placement affects both latency and energy.

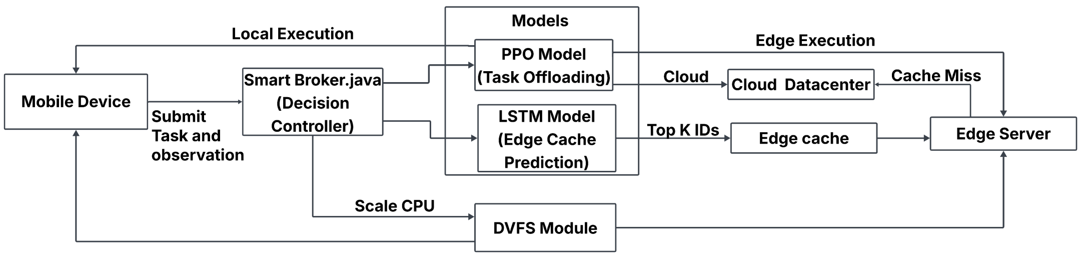


Figure 1: Architecture Diagram

4.2 Framework Overview

The proposed framework is AI driven which means it uses machine learning models in the decision-making process rather than relying on static heuristics which helps in optimizing energy consumption and task response time. The framework integrates:

- Proximal Policy Optimization (PPO) for task scheduling: A reinforcement learning model trained to decide whether to execute the task locally, offload them to the edge or send the task to the cloud. For executing the task it considers real-time system state and task attributes.
- Long Short-Term Memory (LSTM) for edge caching: LSTM neural network which is a time-series forecasting model trained to predict the most popular contents likely to be requested by users to cache at the edge to reduce network latency.
- Dynamic Voltage and Frequency Scaling (DVFS): Implements a hardware-level energy-saving strategy, which will automatically adjust the CPU frequency based on the current workload, thus optimizing for energy efficiency without significantly compromising performance.

Each component operate independently, but communicates via custom SmartBroker class that helps in managing task scheduling, update the device state(battery, bandwidth) and managing cache updates and DVFS logic. The modular design of the project allows each component to be enabled or disabled for experimental comparisons.

4.3 Component Specification

4.3.1 Mobile Device

For tasks executed locally, the device calculates execution time based on its MIPS value and the current operating frequency, which may be dynamically scaled using Dynamic Voltage and Frequency Scaling (DVFS). DVFS enables the device to reduce energy consumption by lowering the CPU frequency during less demanding workloads.

- Configuration: 1 CPU core @ 1000 MIPS, 512 MB RAM, 1 Mbps bandwidth and Battery level normalized between 0 (empty) and 1 (full).
- Execution Time:

$$\text{execTime} = \frac{\text{taskLength(MI)}}{\text{localMips} \times \text{freqRatio}} \tag{1}$$

  where freqRatio is the ratio of current to maximum CPU frequency(f/fmax) (1.0 if DVFS is not enabled). localMips is 1000. This formulation directly follows widely used computation models in mobile edge computing Mao et al. (2017)

- Energy Calculation: For tasks executed locally, energy usage is calculated as:

$$\text{Energy\_local} = 0.25 \times (\text{freqRatio})^2 \times \text{cpuReq} \times \text{execTime} \tag{2}$$

  Where, freqRatio: Ratio of current to maximum CPU frequency. cpuReq is the number of CPU cores required for the task. 0.25 is a power coefficient in Watts, based on typical smartphone CPU measurements and execution time in seconds(tunable), so the energy in Joules.

- Battery Update: After each execution, battery is updated. If the battery level depletes to zero, it is resetted to 0.15, allowing continued simulation.

$$\text{batteryLevel} = \text{batteryLevel} - \frac{\text{Energy\_local}}{8.0} \tag{3}$$

- Bandwidth Update: After local execution, bandwidth is incremented by 0.01, simulating recovery due to reduced network usage.

4.3.2 Edge Server

The edge server proactively maintains a cache of the most popular content, using a Long Short-Term Memory (LSTM) neural network to predict upcoming demand. This reduces latency and network load by ensuring that frequently requested data is available locally at the edge. When a task is offloaded to the edge, the execution time is determined by the server's processing power and DVFS setting. A waiting time is added for network latency and possible cache misses.

- Configuration: 1 CPU core @ 2000 MIPS, 2048 MB RAM, 5 Mbps bandwidth. Caches top 30 most popular contents as predicted by the LSTM module

- Execution Time:

$$\text{execTime} = \frac{\text{taskLength(MI)}}{\text{edgeMips} \times \text{freqRatio}} \tag{4}$$

  where freqRatio is the ratio of current to maximum CPU frequency (1.0 if DVFS is not active). This formulation directly follows the widely used computational models in mobile edge computing Mao et al. (2017).

- Network Waiting Time: The base waiting time is 0.1 seconds; if the requested content is not cached, an additional 0.4 seconds is added as a cache miss penalty.

- Energy Calculation: For edge execution energy is Calculated as,

$$\text{Energy\_edge} = 0.12 \times (\text{freqRatio})^2 \times \text{cpuReq} \times \text{execTime} + 0.04 \times \text{waitingTime} \tag{5}$$

  Where, 0.12 is the energy coefficient for edge processing which is tunable (lower than mobile). 0.04 is idle power usage during network waiting (tunable). cpuReq is the number of CPU cores required for the task

- Latency Calculation: Latency is calculated as,

$$\text{totalDelay} = \text{waitingTime} + \text{execTime} \tag{6}$$

- Battery Update: After each edge execution the battery is decremented, to reflect lower impact of offloading on device battery.

$$\text{batteryLevel} = \text{batteryLevel} - \frac{\text{Energy\_edge}}{15.0} \tag{7}$$

- Bandwidth Update: Offloading to edge reduces bandwidth by 0.05.

4.3.3 Cloud Datacenter

Tasks offloaded to the cloud incur minimal device-side energy consumption, as the computation is handled remotely. Latency is higher due to greater network round-trip times.

- Configuration: 1 CPU core @ 1200 MIPS, 1024 MB RAM, 2 Mbps bandwidth
- Execution Time: Cloud frequency is fixed; no DVFS applied. CloudMips is 1200 and the freqRatio is always 1 for cloud since no DVFS is applied

$$\text{execTime} = \frac{\text{taskLength(MI)}}{\text{cloudMips} \times \text{freqRatio}} \tag{8}$$

- Network Waiting Time: Comprises uplink (0.2s), downlink (0.2s), and cloud processing overhead (0.1s), for a total of 0.5 seconds.
- Energy Calculation: Device-side energy for cloud execution is minimal, as computation happens remotely:

$$\text{Energy\_cloud} = 0.06 \times \text{cpuReq} \times \text{execTime} \ + 0.02 \times \text{waitingTime} \tag{9}$$

  Where, 0.06 reflects the lower device-side energy due to offloading. 0.02 reflects Idle power during network wait.

- Latency Calculation: Latency is calculated as

$$\text{totalDelay} = \text{waitingTime} + \text{execTime} \tag{10}$$

5 Implementation

This section describes the final implementation of the AI driven framework for energy-efficient and latency-aware task offloading in mobile-edge-cloud environments. The discussion covers tools and languages used, development and integration of machine learning models, system integration and execution of scenario-based simulations.

5.1 Tools and Languages Used

Various tools and programming languages were used to implement this project to ensure efficient development and deployment of each component. Python (version 3.11.9) was primarily used for data handling, as well as for training and testing the machine learning models. Java (JDK 17) was used to integrate and simulate these trained models within the CloudSimPlus framework. The Eclipse IDE was used for java development, which offers the comprehensive support for debugging, and managing dependencies. For training the

LSTM model Tensorflow was used and Stable-Baselines3 which is a reinforcement learning library built on PyTorch, was used for training the the Proximal Policy Optimization (PPO) model. To achieve seamless integration between Python-trained models and the Java-based CloudSimPlus simulation environment, trained models were exported to the ONNX (Open Neural Network Exchange) format. This ONNX runtime helps in efficient integration, inference, and deployment within the CloudSimPlus environment. Finally, CloudSimPlus provided a flexible platform for simulating task scheduling, DVFS-based CPU frequency scaling and edge caching across local, edge, and cloud layers supporting accurate performance evaluation and experimentation. Resulting simulation logs were analyzed using Python (Pandas, Matplotlib).

5.2 System Initialization and Task (Cloudlet) Generation in CloudSimPlus

In cloudSimPlus three types of hosts (mobile, edge, cloud) were created, each with realistic resource configurations (CPU cores, RAM, bandwidth) as per the design specification. Each host were assigned a single virtual machine (VM) with specified MIPS, RAM, and bandwidth mentioned in **section 4.3**.

To provide realsitic and varied workload conditions, tasks (referred to as cloudlets in cloudsimplus) are generated randomly with parameters to represent real mobile application:

- CPU instructions that requires between 1 to 300 million instruction to execute. This value is later converted into an expected execution time in the PPO observation vector (Section 5.3.1).
- CPU cores: Number of CPU cores required for each task is chosen between 1 to 4.
- RAM requirement: Each task requires between 256 and 2048 MB of memory.
- Submission delay: Arrival time is set between 2 and 9998 ms
- Deadline: The maximum deadline for completing the task is between 100 and 200 milliseconds.
- Priority: Each task is assigned a priority level from 1 (lowest) to 5 (highest).
- Content ID: With a 50 percent chance, the ID is selected from the current edge cache (to simulate real-world popularity effects); otherwise, a random content ID is chosen.

These parameters are similar to the synthetic dataset used for PPO training and all the feature values are normalized to the range [0, 1] before being used as a input to the machine learning models, which follows the standard RL practice for stability and performance.

5.3 Integration of Machine Learning Models in CloudSimPlus

5.3.1 PPO based offloading decision

For each task generated in CloudSimPlus (see Cloudlet Task Generation in Section 5.2), the smartbroker class will create a 9-dimensional observation vector including Arrival time (normalized), Required CPU cores (normalized), Memory requirement (normalized), Expected execution time is computed from the cloudlet length (in million instructions) and a reference MIPS value (1000), the value is converted to milliseconds and normalized, Task deadline (normalized), Priority level (normalized), Current device battery level (continuous [0, 1]), Current available bandwidth (continuous [0, 1]), Boolean flag indicating whether the requested content is cached at the edge. This observation vector is passed to the PPO model, which will selects an offloading action: (0: Local execution, 1: Edge offloading, 2: Cloud offloading). The custom reward function was used during the training phase of the PPO model to simultaneously optimize for low latency, low energy consumption, battery preservation, and higher cache hit ratio. The trained PPO model, using this reward structure, was then integrated into the simulation to make real-time offloading decisions. The reward for each action is calculated using the below formula:

$$\begin{aligned}\text{reward} = -w_{\text{delay}} \times \frac{\text{delay}}{\text{deadline}} - w_{\text{energy}} \times \text{energy} \\ &+ \text{deadline\_bonus/penalty} + \text{cache\_hit\_bonus/cache\_miss\_penalty} \\ &+ \text{memory/cpu\_scheduling\_penalty/bonus} \\ &+ \text{battery\_aware\_penalty/bonus} \\ &+ \text{bandwidth\_aware\_penalty/bonus}\end{aligned} \tag{11}$$

- Delay penalty helps in penalizing higher task completion times, normalized by the task deadline.
- Energy penalty will penalize high energy consumption during execution.
- Deadline adherence will apply a bonus if a task meets its deadline, and a penalty if it misses.
- Cache hit bonus/penalty: Rewards the agent for using cached content at the edge, penalizes cache misses.
- Heavy task scheduling will encourage offloading of high-memory and high-CPU tasks to the cloud, and penalizes local/edge execution of such tasks.
- Battery conservation will penalize local execution when the device battery is low, and encourages offloading under such circumstances.
- Bandwidth awareness: Rewards local execution when bandwidth is insufficient, and penalizes unnecessary offloading to the cloud when bandwidth is low.

The typical weight values are $w_{\text{delay}} = 1.0$, $w_{\text{energy}} = 2.0$, cachebonus = 4.0, etc. (All settings were adjusted by trial and error during the experiments to find what worked best). The overall goal of the project is to minimize delay and energy, while encouraging or maximizing cache utilization at the edge and maintaining device battery health.

5.3.2 LSTM based Edge cache prediction

In the edge server the cache is updated actively using an LSTM model, which is provided with the past 24 hours of content access logs (each log: vector of 50 access counts). The input is formatted as a [1, 24, 50] tensor. The model will generate a vector of 50 predicted popularity scores. Based on this output top 30 content with highest predicted scores are cached at the edge for next hour which helps in maximizing cache hit rate and minimizing data transmission latency.

5.4 Dynamic Voltage and Frequency Scaling (DVFS) Implementation

Dynamic Voltage and Frequency Scaling (DVFS) is used in this system to save energy while maintaining performance. At every second of the simulation, the mobile and edge devices check how much of their CPU power is currently being used. If a device is busy, the system increases the CPU speed, but if it's less busy, the speed is reduced to save power. However, the speed will never drop below 30 percent of its maximum value to avoid slowing things down too much. This adjustment helps balance energy use and performance automatically, based on real-time workload. DVFS is only applied to mobile and edge devices; cloud servers always run at their maximum, fixed speed.

5.5 Simulation Workflow

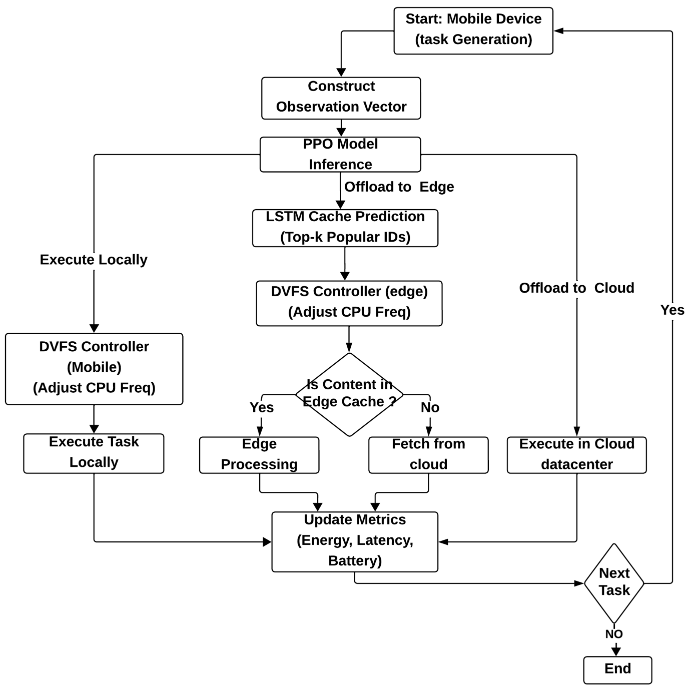
All hosts (mobile, edge, and cloud) and their corresponding virtual machines (VMs) are instantiated with realistic resource specifications. The SmartBroker is a custom broker class designed to manage task scheduling, machine learning model integration (PPO and LSTM), and real-time resource management (such as DVFS logic). It has been initialized with the scenario flags (enabling or disabling PPO, LSTM, and DVFS) and loads the relevant machine learning models via the ONNX Runtime. A set of cloudlets (tasks) are programmatically generated in Java in randomized manner and all tasks are submitted to the broker for scheduling. For each task arrival, the SmartBroker constructs an observation vector describing the task and current device state. If PPO is enabled for task scheduling , PPO model is queried for an offloading decision (local, edge, or cloud). If LSTM is enabled for caching, the LSTM model predicts content popularity for edge caching. DVFS logic, if enabled, adjusts CPU frequencies at each simulation tick based on utilization. Tasks are executed according to the offloading decision. After the task executes on the chosen target, the simulator computes the actual latency, energy consumption, and battery update using the formulas and coefficients defined in **section 4.3**, and records these metrics. To evaluate the effectiveness of each policy, multiple simulation runs are performed under different configurations. Each run enables or disables PPO, LSTM caching, and DVFS as appropriate, producing a comprehensive dataset for analysis. After simulation, all logged metrics are exported to CSV files for further analysis and visualization using Python (Pandas, Matplotlib). This includes per-task and per-scenario data. The complete workflow is shown in figure: 2.

5.6 Outputs Produced

Throughout the simulation after each run, all relevant metrics such as latency in milliseconds, energy consumption in Joules and battery levels are systematically recorded and all results are exported to CSV for analysis. The CSV files have per task and per scenario metrics which is later imported into python (Pandas and matplotlib) for post-simulation analysis.

6 Evaluation

The main purpose of this section is to provide a comprehensive evaluation of the proposed AI-driven task offloading framework in mobile-edge-cloud environments. The evaluation



Start: Mobile Device (task Generation)
Construct Observation Vector
PPO Model Inference
Offload to Edge
LSTM Cache Prediction (Top-k Popular IDs)
Execute Locally
DVFS Controller (edge) (Adjust CPU Freq)
Offload to Cloud
Yes
DVFS Controller (Mobile) (Adjust CPU Freq)
Is Content in Edge Cache ?
Yes
No
Execute Task Locally
Edge Processing
Fetch from cloud
Execute in Cloud datacenter
Update Metrics (Energy, Latency, Battery)
Next Task
NO
End

Figure 2: FlowChart

focuses on three core metrics: average energy consumption, task latency, and device battery level. To analyze the contribution of individual components and their combinations, eight simulation scenarios were implemented:

- Baseline (No Optimization) in which all Tasks executed locally without caching or energy management.
- PPO only: Task offloading decisions made by the PPO agent, without caching or DVFS.
- LSTM only means that the tasks will be executed in the edge server with Edge caching via LSTM predictions, no PPO or DVFS.
- PPO+LSTM means combined PPO offloading and LSTM edge caching.
- DVFS only means dynamic voltage and frequency scaling on the local device, no PPO or LSTM.
- PPO + DVFS means PPO-based offloading with DVFS enabled.
- LSTM + DVFS means Caching and energy management without PPO.
- Integrated Model (PPO + LSTM + DVFS): Full integration of all three methods.

Each scenario was implemented by selectively enabling or disabling components within the simulation. This was managed through simulation flags that toggled the SmartBroker logic at run-time. For example, PPO could be turned off while keeping LSTM predictions active, and DVFS could be disabled for baseline experiments. For each scenario, a workload of 100 tasks was simulated. The following metrics were recorded for every completed task and aggregated for each scenario: Average Energy Used (Joules), Average Latency (milliseconds), Average Battery Level (normalized, 0–1).

Scenario	Avg Energy Used (J)	Avg Latency (ms)	Avg Battery Level
No Optimization	0.08870	147.8600	0.44259
PPO	0.03816	385.8151	0.89397
LSTM	0.03903	517.9300	0.86745
PPO+LSTM	0.03815	352.0534	0.87562
DVFS	0.04442	295.7200	0.71104
PPO+DVFS	0.02935	421.8651	0.93936
LSTM+DVFS	0.02834	591.8600	0.90448
PPO+LSTM+DVFS	0.02773	395.3101	0.93143

Table 1: Comparison of Average Energy Consumption, Task Latency, and Battery Level

6.1 Average Energy Used per Scenario

Figure: 3 and Table: 1 shows average energy across scenerios, the scenario with No Optimization (baseline) shows the highest average energy usage (0.08870 J), meaning the device drains its energy most rapidly when all tasks are executed locally, without any intelligent offloading, caching, or energy management. The DVFS-only scenario moderately reduces energy usage (0.04442 J) by dynamically adjusting CPU frequency based on workload, but without offloading or caching, the savings are limited. Scenarios involving PPO, LSTM, or both (with or without DVFS) show a substantial reduction in energy consumption. The lowest average energy usage is observed in the PPO+LSTM+DVFS scenario (0.02773 J), followed closely by LSTM+DVFS (0.02834 J) and PPO+DVFS (0.02935 J). PPO-based offloading smartly chooses whether to execute tasks locally, at the edge, or in the cloud, that helps in prioritizing options which helps in reducing device-side computation and as well as energy usage. LSTM-based caching at the edge increases the chances of cache hits. This helps in reducing the number of tasks that need to fetch data from the remote cloud, which reduces the latency due to longer network transmissions. DVFS uses CPU frequency and voltage based on workload demands, lowering the energy consumed during less intensive tasks dynamically. When combining all three techniques PPO+LSTM+DVFS it achieves the best results: task scheduling is highly efficient, cache usage are maximized, and CPU runs at energy-efficient speeds, resulting in the lowest possible energy consumption.

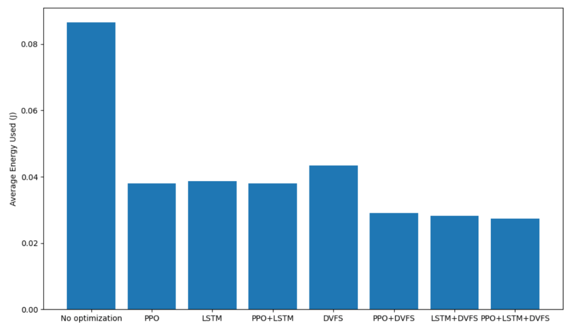


Figure 3: Average Energy Across Scenarios

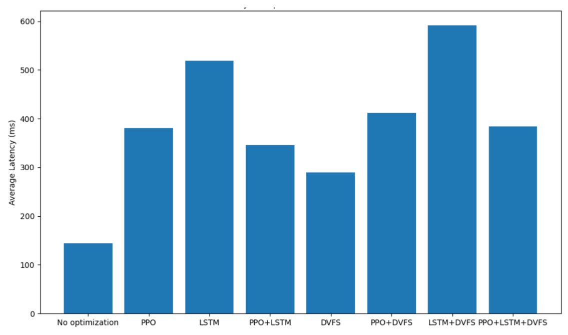


Figure 4: Average Latency Across Scenarios

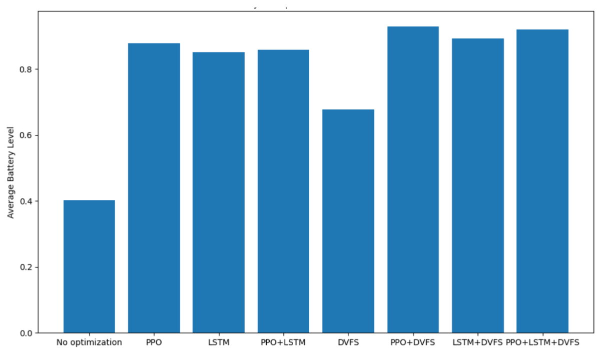


Figure 5: Average Battery Level Across Scenarios

6.2 Average Task Latency per Scenario

Latency results are plotted in Figure: 4 and shown in Table: 1.The lowest latency is observed in the No Optimization (147.86 ms) scenario, since all tasks are processed locally without any offloading or network delays. However, this comes at a very high energy cost and battery drain. Scenarios that uses offloading (PPO, LSTM, and their combinations) experience an increase in average latency (between 350 and 590 ms). This is expected since offloading introduces network transmission and possibly queuing delays. Notably, the PPO+LSTM+DVFS scenario achieves lower latency (395.31 ms) than LSTM+DVFS (591.86 ms) and is better than most other offloading-based approaches, showing that optimal task scheduling (PPO) combined with proactive edge caching (LSTM) and energy efficiency (DVFS) can keep delays within an acceptable range. In general, offloading the task increases latency due to communication overhead, with the help of intelligent scheduling (PPO) minimizes unnecessary offloads and LSTM caching ensures popular content is readily available at the edge, reducing unnecessary cloud offloads. DVFS may slightly increase latency in some scenarios due to reduced CPU speed, but its energy benefits overcome this for battery-limited devices. The best trade-off between reduced energy consumption and acceptable latency is achieved in PPO+LSTM+DVFS, where latency remains reasonable despite aggressive energy savings.

6.3 Average Battery Level per Scenario

Figure: 5 and Table: 1 summarizes the average post-task battery level, in which the battery level is lowest for No Optimization (0.44), indicating rapid battery drain. By enabling DVFS alone improves the battery level by 0.71, but does not approach the levels achieved by the ML-based scenarios. All scenarios with PPO,LSTM,or their combinations result in much higher final battery levels (0.87–0.94), with PPO+LSTM+DVFS (0.93) among the best. By reducing energy consumption through offloading and dynamic frequency scaling, more charge remains in the battery after processing all tasks. The scenario PPO+LSTM+DVFS, which integrates intelligent offloading, proactive caching, and energy scaling, preserves the most battery, making it the best choice for mobile devices.

6.4 Discussion

The results of this research prove that integrating Proximal Policy Optimization (PPO) based intelligent task offloading, LSTM based predictive edge caching, and Dynamic

Voltage and Frequency Scaling (DVFS) into a unified mobile-edge-cloud (MEC) framework shows improvements in both energy efficiency and device battery lifetime, while keeping task latency within an acceptable range. The eight different experiments were set up to see exactly how each part of the system and their combinations affect performance. The results not only showed the expected benefits of using AI for scheduling and caching but also revealed some trade-offs and a few limitations. The existing literature on reinforcement learning methods such as PPO has been shown by (Zhang et al., 2024; Chandrasiri Meedeniya, 2025) to enable dynamic and adaptive task allocation, but they did not use predictive caching or DVFS. Predictive caching studies (Demirci Korcak, 2024; Zong et al., 2023) focus on improving content delivery but did not focus on scheduling or power consumption. Similarly, DVFS-based works (Li et al., 2023; Liu et al., 2025) achieve energy savings, some integrate with scheduling/offloading but do not integrate caching modules. The results confirm that the integrated approach provides better improvements.

Novelty: This system design is especially useful now as mobile applications become increasingly resource-intensive and latency-sensitive. Unlike earlier methods that focus on optimizing a single aspect, the PPO-based scheduler in this work dynamically adjusts task allocation based on real-time device status, workload demands, and changing conditions. The LSTM-based caching module proactively predicts user content requests, reducing reliance on remote cloud servers, and lowering network latency. The DVFS module works closely with task offloading and caching, helping both mobile devices and edge servers find the best balance between saving energy and getting good performance. By directly addressing the integration gap identified by Zhou et al. (2024), this research not only advances theoretical understanding, but also offers a practical implementation for future MEC deployments in domains such as IoT, autonomous vehicles, and smart cities, where low latency and energy efficiency are critical.

Despite the results of the proposed integrated PPO, LSTM caching, and DVFS framework, several limitations remain. First, the effectiveness of the approach is highly dependent on the quality of the training data. Performance may be affected if the actual user behavior or workload patterns differ from those used during training. While energy efficiency is maximized, the latency increases in offloading scenarios cannot be ignored. Although the integrated approach (PPO+LSTM+DVFS) keeps this trade-off within reasonable bounds, in real-time applications with strict QoS requirements, further tuning or multiobjective optimization may be required. The results obtained from the simulated environment are customized to reflect realistic scenarios, and further real-world validation is necessary to fully establish confidence in the approach. The research supports a shift toward feedback-driven, multi-layer optimization in future mobile computing ecosystems. By enabling real-time integration between scheduling, caching, and energy management, the framework helps create smarter, more flexible, and more efficient ways to manage resources for different types of applications.

7 Conclusion and Future Work

This research addresses the challenges of optimizing energy efficiency and latency in mobile-edge-cloud (MEC) systems and answers the research question. The simulation results clearly demonstrate that integration AI-based task scheduling, predictive edge caching, and adaptive DVFS have provided significant improvements in energy efficiency, device battery lifetime, and overall task latency compared to traditional static or single or dual method baselines. The combined approach reduced average energy usage and maintained higher battery levels while keeping latency within acceptable ranges across diverse workload scenarios. This work is important for building better mobile systems in the future. The research provides practical evidence that cross-layer, AI-driven integration leads to superior resource utilization, which is especially critical for modern, resource-constrained mobile devices running latency-sensitive applications. It is built in parts, which is flexible and can be extended in real-world deployments. However, research also has limitations. The simulations, while realistic, are based on synthetic workloads and controlled network conditions. In addition, the current approach assumes accurate model inference and does not yet account for the overhead of model updates or potential privacy concerns.

Future work includes extending the framework to support more heterogeneous environments, incorporating various device types, and user mobility patterns. This could involve training the models with broader, real-world datasets. Real-world validation, by deploying the framework on real devices and networks to see how well it works when users are moving around and devices are different.Updating the system so it can keep learning and adjusting on each device as users needs and tasks change, without sending all the data to a central server. Add security features to protect data and tasks that are sent or stored, especially when using shared or public edge and cloud systems.

References

Azeem, M. R., Muzammal, S. M., Zaman, N. and Khan, M. A. (2022). Edge caching for mobile devices, *2022 14th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS)*, Karachi, Pakistan, pp. 1–6. doi:10.1109/MACS56771.2022.10022729.

Bambagini, M., Marinoni, M., Aydin, H. and Buttazzo, G. (2016). Energy-aware scheduling for real-time systems: A survey, *ACM Transactions on Embedded Computing Systems (TECS)* **15**(1): 1–34. doi:10.1145/2808231.

Breslau, L., Cao, P., Fan, L., Phillips, G. and Shenker, S. (1999). Web caching and zipf-like distributions: Evidence and implications, *Proceedings of IEEE INFOCOM'99. Conference on Computer Communications. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, IEEE, New York, NY, USA, pp. 126–134. doi:10.1109/INFCOM.1999.749260.

Chandrasiri, S. and Meedeniya, D. (2025). Energy-efficient dynamic workflow scheduling in cloud environments using deep learning, *Sensors* **25**(5): 1428. doi:10.3390/s25051428.

Demirci, I. and Korcak, O. (2024). Proactive edge caching with lstm-based popularity prediction, *Proceedings of the 2024 7th International Balkan Conference on Communications and Networking (BalkanCom)*, Ljubljana, Slovenia, pp. 218–223. doi:10.1109/BalkanCom61808.2024.10557165.

Florence, A. P., Shanthi, V. and Simon, C. B. S. (2016). Energy conservation using dynamic voltage frequency scaling for computational cloud, *The Scientific World Journal* **2016**: 1–10. doi:10.1155/2016/9328070.

Fu, S., Zhai, X., Yi, C., Pang, L. and Tan, C. W. (2024). Energy efficient and low latency computation offloading via truly ppo in mobile edge computing networks with multi-uav, *2024 IEEE International Conferences on Internet of Things (iThings), Green Computing & Communications (GreenCom), Cyber, Physical & Social Computing (CPSCom), Smart Data (SmartData) and Congress on Cybermatics*, Copenhagen, Denmark, pp. 98–104. doi:10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics62450.2024.00038.

HajiKhodaverdian, M., Rastaghi, H., Saadat, M. and Shah-Mansouri, H. (2023). Reinforcement learning-based task scheduling using dvfs techniques in mobile devices, *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Toronto, Canada, pp. 1–6. doi:10.1109/PIMRC56721.2023.10293955.

Ismail, L. and Fardoun, A. A. (2017). Towards energy-aware task scheduling (eats) framework for divisible-load applications in cloud computing infrastructure, *2017 Annual IEEE International Systems Conference (SysCon)*, Montreal, QC, Canada, pp. 1–6. doi:10.1109/SYSCON.2017.7934791.

Li, B., Chen, W., Jin, S. and Shi, Y. (2023). Dvfs-based energy-saving workflow offloading strategy in mobile edge computing environments, *2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, Rio de Janeiro, Brazil, pp. 125–130. doi:10.1109/CSCWD57460.2023.10152756.

Liang, J., Xing, H., Wang, F. and Lau, V. K. N. (2023). Joint task offloading and cache placement for energy-efficient mobile edge computing systems, *IEEE Wireless Communications Letters* **12**(4): 694–698. doi:10.1109/LWC.2023.3240476.

Lin, X., Wang, Y., Xie, Q. and Pedram, M. (2014). Energy and performance-aware task scheduling in a mobile cloud computing environment, *2014 IEEE 7th International Conference on Cloud Computing (CLOUD)*, Anchorage, AK, USA, pp. 192–199. doi:10.1109/CLOUD.2014.35.

Lin, X., Wang, Y., Xie, Q. and Pedram, M. (2015). Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment, *IEEE Transactions on Services Computing* **8**(2): 175–186. doi:10.1109/TSC.2014.2381227.

Liu, Y., Qu, H., Chen, S. and Feng, X. (2025). Energy efficient task scheduling for heterogeneous multicore processors in edge computing, *Scientific Reports* **15**(1): 1–25. doi:10.1038/s41598-025-92604-6.

Ma, Y. and Tian, J. (2025). Task offloading scheme based on proximal policy optimization algorithm, *Applied Sciences* **15**(9): 4761. doi:10.3390/app15094761.

Mao, Y., You, C., Zhang, J., Huang, K. and Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective, *IEEE Communications Surveys & Tutorials* **19**(4): 2322–2358. doi:10.1109/COMST.2017.2745201.

Panda, S. K., Lin, M. and Zhou, T. (2023). Energy-efficient computation offloading with dvfs using deep reinforcement learning for time-critical iot applications in edge computing, *IEEE Internet of Things Journal* **10**(8): 6611–6621. doi:10.1109/JIOT.2022.3153399.

Wu, H., Fan, Y., Wang, Y., Ma, H. and Xing, L. (2021). A comprehensive review on edge caching from the perspective of total process: Placement, policy and delivery, *Sensors* **21**(15): 5033. doi:10.3390/s21155033.

Xiong, A., Chen, M., Guo, S., Li, Y., Zhao, Y., Ou, Q., Liu, C., Xu, S. and Liu, X. (2022). An energy aware algorithm for edge task offloading, *Intelligent Automation and Soft Computing* **31**(3): 1641–1654. doi:10.32604/IASC.2022.018881.

Zhang, C., Wu, C., Lin, M., Lin, Y. and Liu, W. (2024). Proximal policy optimization for efficient d2d-assisted computation offloading and resource allocation in multi-access edge computing, *Future Internet* **16**(1): 19. doi:10.3390/fi16010019.

Zhang, Q., Lin, X., Hao, Y. and Cao, J. (2020). Energy-aware scheduling in edge computing based on energy internet, *IEEE Access* **8**: 229052–229065. doi:10.1109/ACCESS.2020.3044932.

Zhou, G., Tian, W., Buyya, R., Xue, R. and Song, L. (2024). Deep reinforcement learning-based methods for resource scheduling in cloud computing: a review and future directions, *Artificial Intelligence Review* . doi:10.1007/s10462-024-10756-9.

Zong, T., Li, C., Lei, Y., Li, G., Cao, H. and Liu, Y. (2023). Cocktail edge caching: Ride dynamic trends of content popularity with ensemble learning, *IEEE/ACM Transactions on Networking* **31**(1): 208–219. doi:10.1109/TNET.2022.3193680.