National College of Ireland

Offloading Scheme for Speech Recognition Applications

MSc Research Project
Cloud Computing

Praise Olamide Abimbola
Student ID: 22101012

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Praise Olamide Abimbola
Student ID:	22101012
Programme:	Cloud Computing
Year:	2022-2023
Module:	MSc Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	14/08/2023
Project Title:	Offloading Scheme for Speech Recognition Applications
Word Count:	8953
Page Count:	22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	10th August 2023

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.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

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

Offloading Scheme for Speech Recognition Applications

Praise Olamide Abimbola
22101012

Abstract

The adoption of Smart Mobile Devices (SMD) is increasing rapidly. Challenges such as limited battery life, storage capacity, bandwidth, device heterogeneity, and security are hindering the use of SMDs for computation-intensive tasks. To overcome these challenges, Mobile Cloud Computing (MCC) provides a solution by offloading such tasks to the cloud. It involves transferring data processing and storage from mobile devices to the cloud infrastructure. The objective of this research is to provide a solution to the management of SMD resources by developing an offloading framework for a real-time speech recognition mobile application on a SMD.

To achieve this, a real-time speech to text mobile application has been developed. This mobile application require considerable computational resources, which can lead to performance and power consumption issues in the SMD. An offloading decision engine has been developed, the decision to offload is made based on the battery life and the network strength of the SMD. If the battery life of the SMD is below 20%, and the network is stable, the application will be offloaded to the cloud using Google Speech API. If the battery life of the SMD is above 20%, and the network is unstable, the application will run locally using flutter speech-to-text package. The results shows that with a strong network connection and low battery level, offloading resource and computation-intensive tasks to the cloud results in lower CPU and memory usage compared to local processing. However, in situations with poor network connectivity or a good battery life, local processing becomes the preferred choice.

1 Introduction

The use of Smart Mobile Devices (SMD) has grown significantly over the years. According to statista [1], the global count of active SMD reached nearly 15 billion in 2021 O'Dea (2020). This surpasses the figure of just over 14 billion in the previous year. The number of SMD is predicted to reach 18.22 billion by 2025, marking a substantial increase of 4.2 billion devices compared to the levels observed in 2020. Despite the advancements in mobile devices, developing advanced applications for SMD remains a challenge due to various resource limitations. These limitations include limited battery energy, slower CPU speeds, inadequate storage space, and restricted network bandwidth. Also, as processors become faster, screens become sharper, and devices incorporate more sensors, the

[1] https://www.statista.com/statistics/245501/multiple-mobile-device-ownership-worldwide/

energy consumption of smartphones outpaces the battery's ability to provide power. Unfortunately, current trends in battery technology suggest that these limitations are likely to persist Biswas and Whaiduzzaman (2019a).

To address these challenges, researchers have explored architectural solutions that can provide the necessary resources for SMDs to have a long lasting battery life. Cloud computing (CC) is a potential solution. Cloud computing offers easily accessible computing power, which allows users to make use of Infrastructure-as-a-Service, Platform-as-a-Service, and Software-as-a-Service services, provided by cloud providers such as Google, Amazon, and Microsoft through concepts like on-demand computing, utility computing, or pay-as-you-go computing.

The core concept of Cloud Computing involves offloading computation to remote cloud servers Chang and Hung (2011). This enables users to efficiently use unlimited cloud resources, develop mobile applications with ease and reduces power usage in the mobile device.

Mobile Cloud Computing (MCC) is the integration of CC into the mobile computing environment. MCC focuses on offloading tasks from the mobile devices to the cloud, where they are processed, and the results are transmitted back to the mobile device. By offloading tasks, smartphones benefit in terms of energy efficiency and execution time, as they can offload computational loads to the more powerful cloud resources Qi and Gani (2012). The architectural diagram of Mobile Cloud Computing is found in Figure 1.

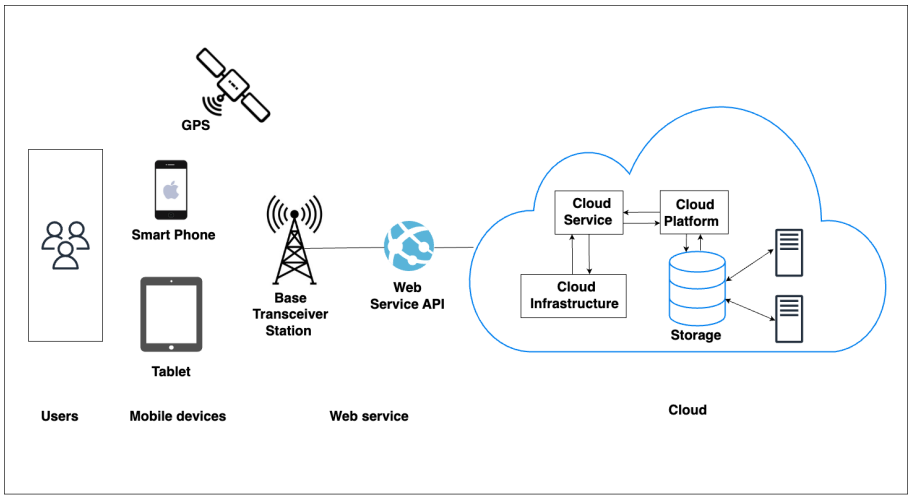

Figure 1: Architectural Diagram of Mobile Cloud Computing

With the convergence of cloud computing and mobile computing, known as Mobile Cloud Computing, numerous innovative smartphone applications and cloud services have been developed.

The objective of this research project is to address the use and management of SMD resources by creating an offloading framework to check when the device has a low battery of 20% or below and if the mobile device network is weak or strong. This affects the decision of the application to be offloaded or not. This aims to enhance the user experience while conserving the resources of the mobile devices. The results will have a positive impact on the wider research community by advancing the field of mobile cloud computing, and will lead to improved efficiency and longer battery life for mobile devices.

Section 2 discusses the review of related works. It covers what has been done, the results and the gaps. Section 3 illustrate the methodology with a case study. Section 4, 5, and 6 covers the design specifications, implementations and result evaluation. Finally, Section 7 concludes this paper and discuss the potential future work.

2 Related Work

In a paper by Cuervo et al. (2010), the paper presents a popular approach to decrease the energy consumption of mobile devices is through remote execution, where applications leverage resource-rich infrastructure by offloading code execution to remote servers. This paper introduces MAUI, an architecture designed to maximize energy savings by employing fine-grained code offloading while minimizing the impact on applications. MAUI utilizes specific features of managed code environments, such as the Microsoft .NET Common Language Runtime (CLR). Firstly, MAUI leverages code portability to create two versions of a smartphone application: one running locally on the device and the other executing remotely in the infrastructure. Secondly, MAUI makes use of programming reflection and type safety to automatically identify methods that is fit to be executed remotely and use only the necessary program state for those methods. Thirdly, MAUI profiles each method, determining its network shipping costs through serialization. It combines the network and CPU costs with wireless connectivity measurements such as bandwidth and latency, using a linear programming formulation to determine how to partition the application at runtime for maximum energy savings based on the current networking conditions. The paper presents MAUI's program partitioning, profiling methodology, and the formulation and solution of program partitioning as a 0-1 integer linear programming problem. Throughout the presentation, the paper also highlights several low-level challenges encountered during the implementation. For instance, the use of power-save mode (PSM) during state transfer can actually increase energy consumption when latency to the server is low. The results demonstrate impressive energy savings and performance improvements achieved by MAUI, with some applications experiencing up to an order of magnitude enhancement.

Biswas and Whaiduzzaman (2019b), discusses the current research on energy-efficient execution offloading techniques and presents a mobile cloud-based application model. Various offloading schemes are implemented, and the paper evaluates their combination under suitable conditions to achieve optimal outcomes in MCC. The successful implementation of offloading and effective use of Cloud Computing (CC) in the mobile environment enable the development of improved applications and services, including mobile voice and keyword search, picture search, mobile games, healthcare, e-commerce, and more. The integration of MCC has the potential to improve our daily lives by addressing various real-life problems.

2.1 Conventional Approach

In Chang and Hung (2011), the authors discuss the kernel-offload paradigm, which guides the design of collaborative mobile cloud applications. They focus on system architecture, application partitioning principles, computation offloading methods, and data access control policies. The paper first presents the design of a cloud-assisted speech recognition (CSR) service and explains its implementation within the collaborative application paradigm. To assess performance, the authors analyze latency, power consumption, and

privacy concerns associated with the CSR service. However, configuring the filtering mechanism for users can be cumbersome, especially when mobile applications and cloud services undergo frequent updates. Additionally, developers face the task of defining and creating an interface for offloading kernel functions to the cloud, followed by implementing and deploying the offload service. Cloud computing technologies, like Google App Engine (GAE), simplify deployment and management but raise privacy issues as developers gain access to users' personal information. The paper introduces an architecture and paradigm for collaborative applications, making minor modifications to existing mobile and cloud computing infrastructures. It includes a performance/power evaluation model to assess the benefits of offloading kernel functions. The authors are still working on developing the software infrastructure to support this paradigm. The speech recognition case study demonstrates promising results by efficiently offloading the most time-consuming function to a cloud server, reducing response time and smartphone power consumption.

The main contribution of the paper by Zhang et al. (2012) is a survey on recent mobile cloud application models, assessing their strengths, weaknesses, and areas that require further attention. They conducted a survey and discussed the distinctions between cloud and mobile cloud computing, mobile cloud architecture, and the key entities influencing computation offloading decisions. Parameters impacting mobile cloud application models were highlighted, and a classification of application models was presented. The paper analyzed and compared these models, addressing their issues and proposing future research directions. Mobile cloud application models based on augmented execution of smartphone clones in the cloud require synchronization between the smartphone and the clone. This necessitates synchronization policies that ensure timely synchronization, considering factors like accuracy, execution delay, and bandwidth utilization. Furthermore, smartphone clones store user data and licensed applications, making them vulnerable to security attacks and piracy. A security mechanism is needed to protect clones from unauthorized access and safeguard smartphone users from malicious virtual machines (VMs) executing in the cloud. There is also a need for a piracy control framework to prevent the illegal installation of smartphone clones on devices of the same model by unauthorized individuals.

The objective of this paper Guo et al. (2016), is to develop an energy-efficient dynamic offloading and resource scheduling (eDors) policy that minimizes the energy expended by mobile devices when executing applications. This paper introduces several contributions compared to previous research. Firstly, it considers the influence of task precedence on computation completion time and characterizes the energy expended as the energy consumption in local computing and the computation completion time in cloud computing. Secondly, the eDors problem is formulated as an energy consumption minimization problem with constraints on application completion time and task precedence requirements. The formulation enforces a maximum completion time to accommodate different application types, such as those sensitive to delays or tolerant of delays. Thirdly, to solve the optimization problem, a distributed eDors algorithm is proposed for the selection of computation offloading, clock frequency control, and transmission power allocation. Importantly, it is discovered that the computation offloading decision depends not only on the computing workload of a task but also on the maximum completion time of its immediate predecessors, as well as the clock frequency and transmission power of the mobile device. Lastly, the eDors policy is implemented on a testbed consisting of 20 Android smartphones and a cloud server, and experimental results demonstrate that it effectively reduces energy consumption and application completion time compared to existing

policies. This work is the first to address dynamic offloading and resource scheduling with the objective of minimizing energy consumption and application completion time while considering completion time deadlines and task precedence requirements. It also incorporates CPU clock frequency control in local computing and transmission power allocation in cloud computing.

Gu et al. (2018) proposes a comprehensive taxonomy of current partitioning and offloading schemes, taking into account various parameters such as the offload infrastructure, augmentation model, communication model, partitioning model, programming language, application partitioning, application profiler, optimization model, allocation decision, and granularity. By analyzing well-known partitioning and offloading frameworks, the performance of these frameworks is evaluated based on the parameters defined in the taxonomy. The study also investigates critical decisions related to partitioning and offloading that significantly impact performance. Considering the resource constraints, applications running on devices should be divided into multiple components. While previous research has focused on partitioning methods and strategy constraints, future studies should implement lightweight approaches, adaptability, and context-awareness in partitioning and offloading.

The increasing demands of computation-intensive mobile applications, such as speech recognition, natural language processing, computer vision, machine learning, augmented reality, and decision making, require more than just powerful SMDs. MCC necessitates significant changes in cloud computing, including programming models for seamless remote execution, a low-latency middle tier, optimized cloud infrastructure for mobile applications, and essential mobile cloud services. In Bahl et al. (2012), the authors purposes that these advancements will allow mobile users to effortlessly harness the benefits of cloud resources without experiencing delays, interruptions, or energy concerns. By empowering mobile users in this way, mobile cloud computing can overcome existing limitations.

In addition, there are various frameworks that facilitate the remote processing of data-intensive tasks on cloud servers. One notable example is the ASM computation offloading framework, which demonstrated significant benefits in terms of energy consumption and application turnaround time. Specifically, it reduced the energy consumption cost of mobile devices by 33% and improved application turnaround time by 45%. This paper contributes to the field in two key ways. Firstly, it classifies existing computation offloading frameworks, analyzing their approaches and identifying crucial issues. Secondly, it presents open issues and challenges in computation offloading for mobile cloud computing (MCC) that require further investigation and elaboration. The paper delves into the concepts of cloud computing, mobile cloud computing, and computation offloading. It provides an overview of existing frameworks for computation offloading, highlighting the various techniques used to enhance smartphone capabilities through the utilization of cloud resources. The paper Khadija Akherfi (2018), studies the challenges and issues faced by current offloading frameworks in MCC. It also explores different approaches employed by these frameworks, including static and dynamic offloading, all aiming to improve smartphone capabilities by saving energy, reducing response time, and minimizing execution costs. However, the authors acknowledge that current offloading frameworks encounter challenges, such as the lack of standardized architectures, which complicates the development and management of proposed frameworks. In conclusion, the paper emphasizes the need for a lightweight paradigm or model that can overcome difficulties and minimize efforts in developing, deploying, and managing offloading frameworks. The

authors propose exploring alternative solutions, such as a middleware-based architecture with an optimizing offloading algorithm, to enhance existing frameworks and provide more efficient and flexible solutions for MCC users.

The primary focus of this paper Khan et al. (2014) is to conduct a comprehensive survey of the latest mobile cloud application models developed between 2008 and 2012. The authors aim to identify the strengths, weaknesses, and unresolved issues in these models, highlighting areas that require further attention. Additionally, the paper explores the differences between cloud computing and mobile cloud computing, providing insights into mobile cloud architecture and the key factors influencing computation offloading decisions. Furthermore, the authors examine the parameters that impact mobile cloud application models and present a classification of these models. They critically compare and analyze the different application models, addressing their significant unresolved challenges, and propose potential directions for future research in the field.

Hwang (2015) proposed a cloud offloading approach specifically designed for web applications, aiming to optimize resource utilization on devices based on web standards. The authors justify their research by highlighting the increasing popularity of smart devices and the significant presence of web applications among smart mobile device users. By developing an advanced method for offloading web applications to leverage multiple devices, they aim to provide mobile users with more flexible access to resources. Through their experiments, the authors observe the potential for resource sharing among devices, where the workload of web applications on one device can be executed on another device's browser. However, they also identify a performance limitation due to the network overhead associated with the current WebRTC channel used for communication between devices. They suggest that this limitation could be overcome by optimizing the channel's settings. The authors discusses various benefits from cloud offloading, including improved performance, reduced processing time, and lower battery consumption. They introduce WWF-D as an extension of WWF, emphasizing the importance of developing a unified version that integrates resource utilization on both servers and devices. They outline future research directions, which involve enhancing WebRTC latency, exploring intelligent policy selection based on device conditions, and establishing a framework for incorporating different policies into WWF. Furthermore, they intend to research additional benefits associated with WWF-D, such as improved battery life.

The current state code offloading for mobile devices is examined in this research paper Jiao et al. (2013), along with the major challenges encountered when developing a framework for cloud-based offloading that is more effective. It is also investigated how current technologies may help with the implementation of such a framework. The paper discuses MAUI technique, which performs computation offloading, treats an application as a call graph represented by a directed graph G=(V,E), where a call graph is a directed graph. Each vertex in G represents a method, and invocations are represented by edges. The offloading is converted into a graph partitioning problem, with one partition running locally and the other on the cloud. In order to maximise energy savings while taking into account the overall execution time, an integer linear programme is used. Another method examined in the paper is CloneCloud. CloneCloud also enables method-level code for offloading. It makes use of profile trees built by the CloneCloud Profiler to represent execution on mobile devices and execution in the cloud. The nodes and edges of the tree indicate method calls and invocations, respectively. The call graph of MAUI and CloneCloud are very similar. Without taking into account the resources used at the offloading destination, MAUI and CloneCloud both place a higher priority on reducing

energy consumption and improving application performance on the mobile device. This paper reviews provides a comparison of both MAUI and CloneCloud. It draws attention to the difficulties that prevent the development of an offloading framework that is more effective and advises using current technology to aid in implementation.

In Kosta et al. (2012), the authors introduced ThinkAir, a framework built to make it easier to migrate mobile applications to the cloud. ThinkAir enables method-level compute offloading and makes use of smartphone virtualization in the cloud. The methodology builds on earlier work by emphasising the elasticity and scalability of the cloud and amplifying the power of mobile cloud computing by parallelizing method execution over many virtual machine (VM) images. They install ThinkAir and run a range of benchmark tests, from simple micro-benchmarks to more complex applications. The evaluation result show a considerable reduction in execution time and energy use. For instance, the face recognition and virus scan applications both achieve one-order reductions in execution time and energy usage. However, the N-queens puzzle application experiences a two-order reduction. It also shows how parallelizable applications execute on several virtual machines (VMs) in the cloud, substantially reducing execution time and energy usage. ThinkAir addresses CloneCloud's limitations on applications, inputs, and environmental conditions while also addressing MAUI's scaling problems by developing a whole smartphone systems as virtual machines in the cloud. For a straightforward integration with ThinkAir, they offer a toolchain and source code changes for the application. The advantages of ThinkAir for profiling, code offloading, and adjusting to changing computational needs are demonstrated in this studies with micro benchmarks and computation-intensive applications.

Wei et al. (2017) introduces the MVR architecture, a method for offloading computation in mobile edge computing. The MVR architecture seeks to close the gap between resource-rich Edge Cloud and computation intensive applications. There are a number of different challenges that may arise in the future due to the emergence of technologies like 5G. For instance, in a multi-dynamic environment, it is necessary to address the optimisation problem of leasing cost, runtime, energy usage, delay time, and others. The ability to create and sustain a reliable "Mobile Federation" is a crucial issue to research. There is also need to evaluate how effective these approaches works in environments with numerous clouds and service providers. The implementation of privacy services in the MEC environment is expected to make computation offloading more complex due to the quick and frequent changes in security levels experienced by mobile users across various locations. The future research focuses on finding more ways to address the difficulties associated with compute offloading.

Lee and Shin (2013), offers a mobility model that takes the normal patterns in individual user mobility. The authors create a computation offloading decision-making method based on user mobility by building on this model. They run trace-based simulations using actual log data traces from 14 Android users to evaluate the efficacy of our method. The evaluation's findings show that, when exhibit high mobility, the performance of mobile devices is improved in terms of respect time and energy usage. This paper proposes a method for offloading mobile compute, highlighting the significance of user mobility-aware decision-making. Their method can predict near-future network conditions and make wise offloading decisions by using user mobility models.

This research Bajaj et al. (2022), examines context-based offloading in ensuring that IoT-enabled services achieve their performance requirements. In order to assess their performance and novelty, a number of current frameworks including EMCO, MobiCOP-

IoT, Autonomic Management Framework, CSOS, Fog Computing Framework, and others are compared to MAUI, AnyRun Computing (ARC), AutoScaler, Edge Computing, and Context-Sensitive Model for Offloading System (CoSMOS) frameworks. The report outlines potential future possibilities for offloading based on the conclusions and limits of these frameworks. In order to understand the relevance of context in data offloading, the paper gives a thorough examination and comparison of several data offloading systems. On the basis of their creative methods and results, certain current frameworks are implemented and assessed. The analysis shows that offloading is essential for ensuring that IoT-enabled services satisfy their performance requirements. Making judgements about when and where to undertake offloading requires prior knowledge of the data environment. Despite the fact that various learning approaches were used in some implementations, they were primarily restricted to mobile-based scenarios. In order to offload compute, a smart middleware architecture that includes hybrid learning techniques is still possible to construct. Due to the paucity of research in these domains, the report also indicates possible future work in edge structures and edge-based cloud architectures for offloading frameworks. In order to improve performance, fixed scheduling approaches can also be used more effectively.

Jade is introduced in Huerta-Canepa and Lee (2008). A solution created to improve Android applications with features for energy-aware compute offloading. Jade automatically decides the best location for code execution based on the condition of the device and the application. Based on changes in the workload, communication costs, and device status, it dynamically alters the offloading approach. The creation of Jade was influenced by earlier research on code offloading, remote execution, and programme partitioning. In their MAUI proposal, Cuervo et al. focused on the energy-aware offloading of mobile code to infrastructure. By annotating methods or classes as remotely accessible, MAUI enables developers to first divide their applications. The MAUI solver chooses which remotable methods should run locally and which should be offloaded during runtime. Jade, as opposed to MAUI, has a complex programming paradigm with extensive APIs, giving developers full control over how the application is divided, where code is offloaded, and how remotable code interacts with local code. Jade gets rid of dependencies between remotable tasks, which spares the profiler and optimizer from having to perform a thorough programme analysis. As a result, compared to MAUI, the energy cost associated with programme profiling and cost model computation is reduced.

In Ning et al. (2019), the paper presents a three-layer offloading framework for intelligent Internet of Vehicles (IoV) systems is proposed. The objective is to reduce total energy consumption while meeting user delay constraints. The authors break down the issue into two components: flow redirection and offloading decision, to address its high computational complexity. To solve the problem, they provide a deep reinforcement learning-based approach. The performance assessments, which are based on actual taxi traces in Shanghai, China, show how successful their techniques are. Their method delivers an average energy consumption decrease of about 60% when compared to the baseline algorithm.

2.2 Machine Learning Approach

In this paper Cao and Cai (2018), the researchers evaluates the development of an efficient distributed algorithm for multi-user computation offloading in a cloudlet-based mobile cloud computing (MCC) system. In this system, each SMD independently de-

cides between local computing and cloud computing based on energy consumption and time cost. However, in a multi-channel environment, efficient coordination of wireless access among multiple devices is crucial. Without proper coordination, simultaneous offloading on the same channel can lead to collisions and significantly reduced transmission rates, resulting in low energy efficiency and long transmission times. The researchers identify two key challenges for achieving efficient computation offloading in a distributed cloudlet-based MCC system: individual decision-making between local and cloud computing for each mobile device, and channel selection without information exchange with other devices to achieve high transmission rates. To address these challenges, the researchers formulate the multi-user computation offloading decision-making problem as a non-cooperative game. Mobile devices act in their own interests and make local offloading decisions based on strategic interactions with other devices to reach a mutually satisfactory solution. By analyzing the game's structure, they prove that it is an exact potential game, ensuring the existence of at least one pure-strategy Nash Equilibrium Point (NEP). To achieve NEPs in a fully distributed manner, they introduce machine learning technology and propose a fully distributed computation offloading (FDCO) algorithm. This algorithm enables each SMD to learn from individual information and independently and automatically adjust its behavior towards the NEPs. Simulation results are presented to demonstrate the convergence behavior and advantages of the proposed FDCO algorithm compared to other methods.

Due to its safe, open, and decentralised nature, blockchain technology has recently grown in mobile applications. However, due to their constrained computing and storage capacities, mobile devices make it difficult to carry out complex operations like mining blockchains and running computation-intensive data applications. This paper Nguyen et al. (2020), presents a blockchain network based on mobile edge computing (MEC) that allows multiple mobile users (MUs) to participate as miners in order to solve this problem. They use wireless channels to outsource their data processing and mining operations to a nearby MEC server. With our strategy, workload offloading, user privacy protection, and mining profit all function as a single optimisation issue. In order to minimise long-term system offloading utility and increase privacy levels for all blockchain users, they model it as a Markov decision process. They initially suggested an offloading technique based on reinforcement learning (RL). With the help of this system, MUs may choose the best offloading options depending on the characteristics of wireless channels, power hash states, and blockchain transaction states. They created a deep RL method employing a deep Q-network to enhance offloading performance for larger-scale blockchain scenarios. Without any prior knowledge of the system dynamics, this approach effectively solves a huge state space. In comparison to benchmark offloading methods, their RL-based offloading techniques dramatically improve user privacy, lower energy use and compute delay, and minimise offloading costs.

The authors of Ali Shakarami (2020), presented a thorough analysis of computation offloading strategies based on machine learning (ML) in the context of mobile edge computing (MEC). The goal is to identify and classify the current mechanisms according to a traditional taxonomy and to draw attention to unresolved issues. The proposed taxonomy divides the three primary categories ML-based offloading mechanisms into three main fields: reinforcement learning-based, supervised learning-based, and unsupervised learning-based mechanisms. The classes are compared based on including performance measures such as, case studies, methodologies, and evaluation tools. They also talk about each class's benefits and drawbacks. In the MEC context, granularity and segmentation

are essential for increasing the effectiveness of time or resource-constrained applications. Granularity is divided into two groups: coarse-grained and fine-grained. They also discuss on the four partitioning kinds of virtual machine (VM), application, task, and method.

Computation offloading, which involves moving time-sensitive and computationally heavy mobile application operations to distant cloud-based data centres, has recently come to light as a feasible method for overcoming the limits of mobile devices (MDs). Offloading to edge points can have substantial advantages in the setting of cyber-physical-social systems (CPSS), such as traffic infraction tracking cameras in smart cities. The authors take into account the existence of mobile edge computing networks (MECNs) in diverse geographies. These networks include different access points, numerous edge servers, and N MDs, each of which has M separate real-time massive tasks. Through mobile networks or access points, MDs can connect to MECNs. Each task may be handled locally by the MD or offloaded remotely. Alfakih et al. (2020) presents the state-action-reward-state-action (SARSA) algorithm, a reinforcement learning-based algorithm, to address the resource management issue in the edge server and make the best decisions for offloading with the goal of minimising system cost, which includes energy consumption and computing time delay. Reinforcement learning is used in this method, known as OD-SARSA (offloading decision-based SARSA), to identify the best offloading technique and reduce system costs in terms of energy use and processing latency. As an optimisation challenge, they create a MEC system model that takes into account both compute time delay and power usage. They propose the OD-SARSA technique in particular, which uses reinforcement learning to decide on offloading choices optimally and reduces system cost in terms of energy usage and computation latency.

Due to the paucity of computing capacity on front-end devices, deep learning is only used in existing AR applications, limiting its potential to improve Augmented Reality (AR) devices. To solve this problem, the authors provide a distributed system that links front-end hardware with more potent back-end "helpers" to allow deep learning tasks to be executed locally or offloaded remotely. With the help of intelligently utilising network conditions, back-end server loads, and application needs, this framework strives to choose the best decision. Ran et al. (2017) presents the initial exploration into putting such a framework into practise, with smartphones as the presumptive front-end devices. One of their contributions is the creation of an Android app that can identify objects in real-time, either locally on the smartphone or remotely on a server. Additionally, by taking into account system variables like video quality, deep learning model size, and offloading selections, they analyse the trade-offs between object identification accuracy, latency, and battery usage.

Huang et al. (2020) focuses on a Mobile Edge Computing (MEC) network powered by wireless technology that employs a binary offloading strategy, where compute tasks from wireless devices (WDs) are either fully offloaded to a MEC server or done locally. The goal of the research is to provide an online algorithm that can optimise wireless resource allocations and task offloading choices based on dynamically changing wireless channel circumstances. The DROO framework makes use of a deep neural network as a scalable method to learn the binary offloading choices based on past experiences, obviating the requirement to solve combinatorial optimisation issues and drastically lowering computing cost, especially in large networks. They offer an adaptive method that dynamically modifies the DROO algorithm's parameters in real-time to further increase efficiency. In comparison to existing optimisation techniques, numerical results show that the proposed approach achieves near-optimal performance and drastically decreases computing time.

For instance, DROO's CPU execution latency is less than 0.1 seconds in a network of 30 users, enabling real-time and ideal offloading even in rapid fading situations.

Offloading computation tasks in mobile Fog environments is difficult because various mobile devices have different resource needs in terms of time and space. In order to give low-latency service to mobile service users, the authors of Alam et al. (2016), present a code offloading method based on reinforcement learning. The method makes use of a distributed reinforcement learning technique to offload basic code blocks in a decentralized manner, allowing the deployment of mobile codes around mobile Fog nodes that are spread out geographically. They ran simulations with OMNeT++ to assess the suggested prototype while taking into consideration the fluctuating service needs of mobile users as well as the dynamic resource availability of mobile Fog environment. The result show that the proposed approach lowers execution time and latency for using mobile services, as well as guaranteeing that mobile devices use less energy. They also acknowledge that addressing mobility and context-aware offloading are difficult problems that must be solved for mobile Fog computing to succeed. In the context of mobile Fog computing, the continuing research focuses on topics like virtualization, mobility management, privacy issues, and resolving end-user concerns.

The paper Ran et al. (2017), present a distributed algorithm for offloading in a mobile edge computing (MEC) system that is based on model-free deep reinforcement learning. In this method, each device can decide which tasks to offload without reference to task models or decisions made by other devices. They use methods like long short-term memory (LSTM), duelling deep Q-network (DQN), and double-DQN to enhance the estimation of long-term costs. They show through simulations that their approach efficiently makes use of the processing power of edge nodes, lowering the average latency and dropped job ratio in comparison to other techniques. Their goal in this work is to address the job offloading problem inside a MEC system's unknown dynamics of load levels at edge nodes. Compared to previous works done, the method takes a more realistic MEC scenario into account. They analyse non-divisible tasks with queuing systems, in contrast to prior studies that either assume divisible tasks or ignore queuing systems, taking into consideration the real-world scenarios where task processing and transmission might take place over a number of time slots. This poses difficulties since the introduction of new tasks may cause those from earlier time slots to be delayed. Furthermore, they concentrate on delay-sensitive activities with processing deadlines, in contrast to prior works that mostly deal with delay-tolerant tasks. This increases complexity since the deadlines affect the load levels at edge nodes and, as a result, the delay of tasks that have been offloaded. The MEC system's interconnected activities make it difficult to apply traditional methods like game theory and online optimisation. They suggest using deep Q-learning, a model-free deep reinforcement learning approach, to address these issues. They can efficiently handle the MEC system's complexity by using deep Q-learning, which also offers a reliable task offloading method.

In Table 1 presents a table of comparisons between various code offloading frameworks and approaches.

3 Methodology

This section discusses the steps taken to achieve the object of this research. As stated earlier in the paper, the aim of this research is to provide a solution to the management

Table 1: Comparison of code offloading frameworks

Offloading Framework	**Offloading Mechanism**	**Pros**	**Cons**
MAUI	Mobile Assistance Using Infrastructure	Supports dynamic profiling and migration	Not as widely used as other frameworks
Think Air	Context-aware offloading approach	Dynamic resource allocation and parallel execution	More complex to implement
Clone Cloud	Code execution offloading to cloud servers	Automatic code offloading	Requires a customized Android branch

of SMD resources by creating an offloading framework for a real-time speech recognition mobile application. This is achieved by developing a real-time speech recognition as the case study. An offload decision engine that uses the current device context to determine whether or not a computing intensive task should be offloaded is developed.

3.1 Application Tools used

To carry out this research, an Android mobile speech recognition application is developed. This application is a real-time speech to text a application. The decision to offload is made based on the battery life of the SMD and the network strength. If the decision made is to execute locally, flutter speech-to-text package is called and used. If the decision made is to offload to the cloud, Google Speech API is called and used. To develop this application, Dart programming language and Flutter SDK is used. Android Studio IDEs are also used for developing and testing on Android and iOS devices.

3.2 Offloading Decision Engine

The application partitioning used to carry out this research is dynamic which means that the decision to offload is made during the runtime of the application. The conditions or requirements to carry out this decision is based on the SMD'S battery life and the available Network. The application checks the battery percentage of the SMD. If the battery is 20% or below, the compute intensive part of the application is offloaded to the cloud and the application switches to a cloud-based speech recognition to offload the computation to the Google Speech API Anggraini et al. (2018). Also, if the network strength of the SMD is strong, then the application will be offloaded as well. Otherwise, the speech recognition application will run locally using flutter speech-to-text package to process the speech input and provide the text output. These requirements are monitored using Android Studio profiler. The profiler is used to check the resources being used when the application is running.

3.3 Real-time Monitoring

As stated earlier, the application partition is dynamic. This allows the offloading decision to be made during runtime. The SMD application developed is a real-time speech to text

application. The application listens to the user's voice through the SMD's microphone and converts the speech into text in real-time. The application is continuously mentioned with Android Studio profiler, once the battery life drops below the threshold, based on the offloading decision condition, the application offloads and switches to the a cloud-based application. If the battery percentage increases above 20%, it switches back to a local application.

3.4 Case Study – Speech to text mobile application

The speech to text application runs on android devices. It is a real-time application with a simple user interface that has a record button. When the user presses the record button, the application starts to listen to the user's speech and the recognized text is displayed on the screen in real-time. The package used to perform this task locally is flutter speech-to-text package Eze (2022) which is a well-known and used plugin for speech recognition in flutter applications. This package provides real-time speech to text conversion locally on the SMD and it supports configurations such as language and punctuation. The API used to perform this task on the cloud is google speech API which is a cloud-based speech recognition service provided by Google Cloud. This API provides accurate speech recognition, supports various languages but in this case study, English language is used. It also provides automatic punctuation, and streaming recognition for real-time processing. To evaluate this, android studio profiler is used to compare the accuracy, speed, and battery consumption of local and cloud-based speech recognition under different scenarios.

4 Design Specification

The flow chat in Figure 2 shows the workflow of the application and offloading process. Firstly, there is the SMD which has the speech to text application running on it. Secondly, the decision engine decides whether to offload the task to the cloud or run it locally based on the battery life and network of the device. Thirdly, the result of the application is displayed on the screen in real-time. For the application to successfully run locally, flutter speech-to-text package is implemented. The SMD requires access to the device's microphone to capture the users audio input for the speech-to-text plugin to process. On the other hand, for the application to successfully run on google cloud, google speech. API is used to implement this. The application needs strong active internet connection for easy communication with the google speech API, send audio data to the API and receive recognized words to be displayed back on the users screen. The application constantly monitors the network availability and battery life of the SMD to make offloading decision.

The application starts with a splash screen, the user is then required to login if an account has been created or sign up if an account has not been created. The users authentication is stored in a database on firebase. The user logs into the application and is presented with the main interface. When the user presses the record button, the application request for the user permission to access the device's microphone. The application begins local speech to text process if the battery is above the threshold as started in the offloading decision engine. While the application is running, the battery life and network availability is monitored and if there is a drop in the battery life which is below the threshold then the application offloads to the cloud. The speech recognition result remains the same. It is displayed in real-time on the mobile device.

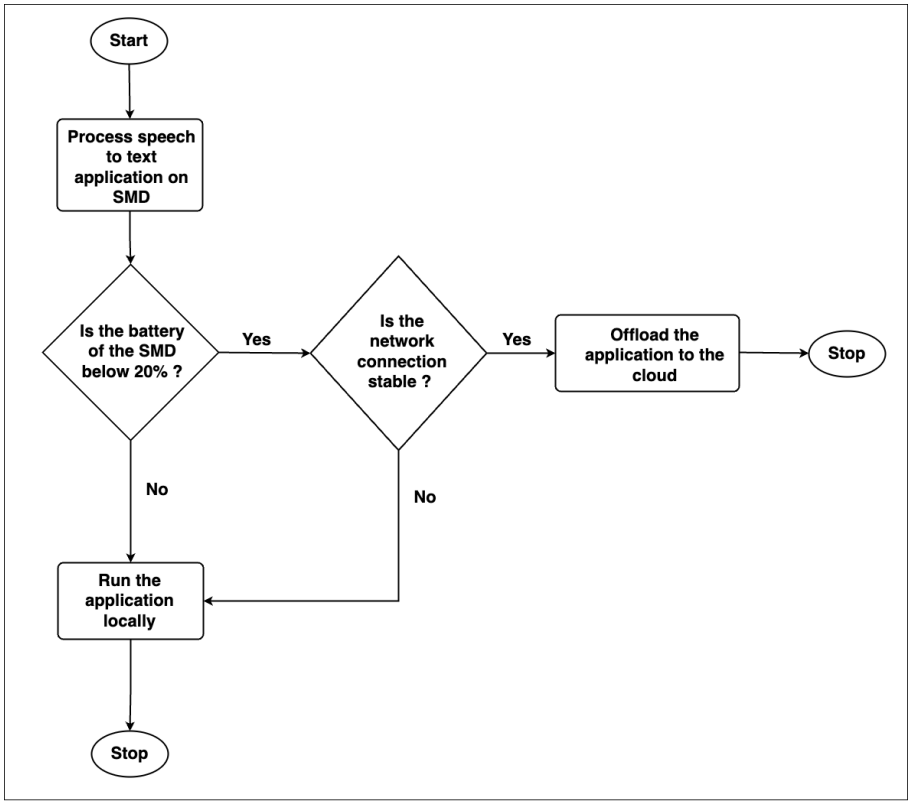

Figure 2: Flow chat of workflow of the application and offloading process

5 Implementation

The real-time speech-to-text mobile application is primarily written in Dart programming language. The application is built for Android platform using Android Studio IDE. The application combines both local and cloud-based speech recognition packages. For the offloading, Google Speech API is called. The application integrates flutter frameworks and plugins such as 'flutter-sound-lite' for audio recording, 'permission-handler' for managing device permissions, and 'connectivity' for monitoring network availability. For the testing, two SMDs are used. The details are given in the configurations manual.

5.1 Offloading Decision Engine

As mentioned in section 3.2, and shown in algorithm 1, the offloading decision engine is used to check the SMD's for the conditions stated. The performSpeechToText() function has a Future void which represents a potential value that is expected in the future and the function marked as 'async', means it can use 'await' to wait for other asynchronous operations to complete. The bool isNetworkConnected = await checkNetworkConnectivity() is used to place a hold on the application until the network is checked if the device has an stable connection and it returns a boolean result. The result is stored in the variable 'isNetworkConnected', which will hold a 'true' value if the network is connected, and 'false' otherwise. To check the battery level, the 'bool isBatteryLow = await checkBatteryLevel();' similar to the previous line, uses 'await' to wait for the 'checkBatteryLevel()' function to complete and return a result. The result of 'checkBatteryLevel()' is stored in the variable 'isBatteryLow', which will hold a 'true' value if the battery life

is low, and 'false' otherwise. 'if (isNetworkConnected && !isBatteryLow) ... else ... '; is the if...else condition statement. If both conditions are true, the device has an active network connection and the battery life is not low, the code block inside the 'if' statement will be executed. Otherwise, if any of the conditions is false, the code block inside the 'else' statement will be executed. The 'performCloudSpeechToText(); and performLocalSpeechToText();' functions also have a Future void. The await performCloudSpeechToText(); is called if the condition is true, otherwise, the performLocalSpeechToText(); is called.

Algorithm 1 Speech Recognition Algorithm

```
function performSpeechToText
  isNetworkConnected ← checkNetworkConnectivity()
  isBatteryLow ← checkBatteryLevel()
  if isNetworkConnected && isBatteryLow then
    await performCloudSpeechToText
  else
    await performLocalSpeechToText
  end if
end function
```

This is a successful implementation of a real-time speech-to-text mobile application that uses both local and cloud-based speech recognition. The developed model and decision engine addresses the limitations of SMDs.

6 Evaluation

This section evaluates the result gotten from the implementation of the project. To do this, two SMDs are used. An android emulator called Pixel 2 API 33 is used to run the application of a battery life of 100% while a physical android device called CLT L09 is used to test the application at a battery percent of 18%. Android studio profiler measures the CPU, memory and Energy level of the resources begin used on both devices.

6.1 Experiment 1 / Offloading Locally

The image in Figure 3 illustrate the resource usage (CPU, memory, and energy level) when running the application locally on Pixel 2 API 33.

Figure 3: Android Studio Profiler - CPU, Memory and Energy usage

6.2 Experiment 2 / Offloading on the cloud

The image Figure 4 illustrates the resource usage (CPU, memory, and energy level) when running the application locally on CLT L09.

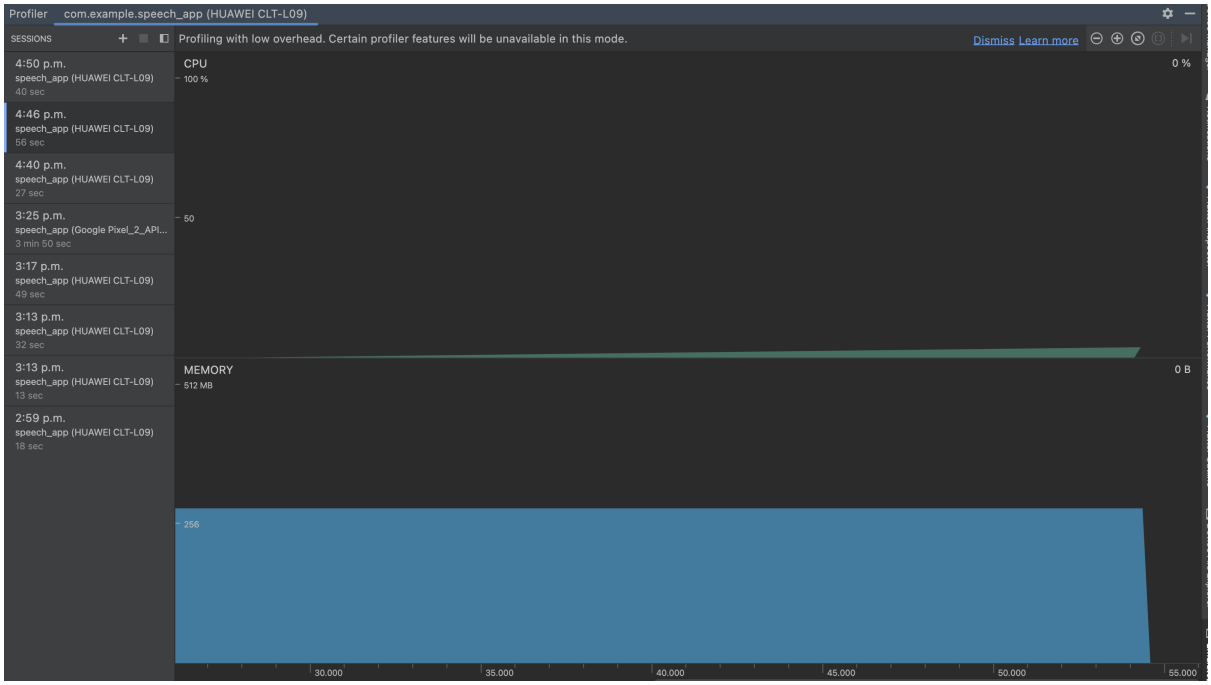

Figure 4: Android Studio Profiler - CPU, Memory and Energy usage

Table 2: Offloading Decision Engine Test

Scenario	SMD Context	Offloading Decision
1	Battery – 100%, Network – Strong	Local
2	Battery – 18%, Network – Strong	Cloud
3	Battery – 20%, Network – weak	Local

6.3 Discussion

This evaluation shows that the case study with a strong network connection and low battery level, offloading resource and computation-intensive tasks to the cloud results in lower CPU and memory usage compared to local processing. In situations with poor network connectivity, local processing becomes the preferred choice to ensure the real-time application is responsive. The evaluation of the research results provides valuable insights into SMD resource usage analysis and the results of offloading tasks to the cloud versus local processing. It also highlights the significance of network connectivity and battery life in determining the most optimal processing approach. This research contribute to the existing literature on resource optimization in mobile application development, and offer guidance to making design decisions for enhancing application performance and user experience when using SMDs.

7 Conclusion and Future Work

In this research, a framework for offloading the computing-intensive part of a real-time speech recognition android application on a SMD was developed. In this framework, the decision to offload is based on the battery life of the SMD and the network connectivity. To achieve this, an offloading decision engine which made use of the SMDs current context at runtime to decide if the application should run locally or on the cloud has been developed. To evaluate this, android studio profiler is used to monitor the resources used while the application was running. The obtained results indicate that with a strong stable network connectivity and low battery level, offloading tasks to the cloud resulted in reduced CPU and memory usage, leading to optimized battery consumption. Conversely, in situations with poor network connectivity and high battery levels, local processing was more preferable to ensure app responsiveness and user satisfaction. This research study also explored the trade-offs between cloud-based processing and local execution based on network connectivity and battery life on SMDs and contributes to the broader goal of improving the battery life of SMDs. Overall, the research successfully addressed the research question and achieved the aim of the research. In comparing the results with existing research, it is found that the research is consistent with them. The research contributes to the body of knowledge on SMD resource optimization and provides practical guidance for developers to make offloading decisions.

While this research addresses a limitation of SMDs, there are many other aspects of SMDs and offloading that need to be explored. One of which includes researching the integration of edge computing techniques to offload resource-intensive tasks to edge servers when available and analysing the effect of edge computing on mobile applications.

References

Alam, M. G. R., Tun, Y. K. and Hong, C. S. (2016). Multi-agent and reinforcement learning based code offloading in mobile fog, *2016 International Conference on Information Networking (ICOIN)* pp. 285–290.

Alfakih, T., Hassan, M. M., Gumaei, A., Savaglio, C. and Fortino, G. (2020). Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa, *IEEE Access* **8**: 54074–54084.

Ali Shakarami, Mostafa Ghobaei-Arani, A. S. (2020). A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective, *Computer Networks* **182**: 107496.

Anggraini, N., Kurniawan, A., Wardhani, L. K. and Hakiem, N. (2018). Speech recognition application for the speech impaired using the android-based google cloud speech api, *TELKOMNIKA (Telecommunication Computing Electronics and Control)* **16**(6): 2733–2739.

Bahl, P., Han, R. Y., Li, L. E. and Satyanarayanan, M. (2012). Advancing the state of mobile cloud computing, *Association for Computing Machinery* (8): 21–28.

Bajaj, K., Sharma, B. and Singh, R. (2022). Implementation analysis of iot-based offloading frameworks on cloud/edge computing for sensor generated big data, *Complex Intelligent Systems* **8**(5): 2198–6053.

Biswas, M. and Whaiduzzaman, M. (2019a). Efficient mobile cloud computing through computation offloading, *International Journal of Advancements in Technology* **1**(10).

Biswas, M. and Whaiduzzaman, M. (2019b). Efficient mobile cloud computing through computation offloading, *International Journal ofAdvancements in Technology* .

Cao, H. and Cai, J. (2018). Distributed multiuser computation offloading for cloudlet-based mobile cloud computing: A game-theoretic machine learning approach, *IEEE Transactions on Vehicular Technology* **67**(1): 752–764.

Chang, Y.-S. and Hung, S.-H. (2011). Developing collaborative applications with mobile cloud - a case study of speech recognition, *Journal of Internet Services and Information Security* **1**: 18–36.

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R. and Bahl, P. (2010). Maui: Making smartphones last longer with code offload, *Association for Computing Machinery* p. 49–62.

Eze, T. (2022). Effective detection of local languages for tourists based on surrounding features.

Gu, F., Niu, J., Qi, Z. and Atiquzzaman, M. (2018). Partitioning and offloading in smart mobile devices for mobile cloud computing: State of the art and future directions, *Journal of Network and Computer Applications* **119**: 83–96.

Guo, S., Xiao, B., Yang, Y. and Yang, Y. (2016). Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing, *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications* pp. 1–9.

Huang, L., Bi, S. and Zhang, Y.-J. A. (2020). Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks, *IEEE Transactions on Mobile Computing* **19**(11): 2581–2593.

Huerta-Canepa, G. and Lee, D. (2008). An adaptable application offloading scheme based on application behavior, *22nd International Conference on Advanced Information Networking and Applications - Workshops (aina workshops 2008)* pp. 387–392.

Hwang, I. (2015). Design and implementation of cloud offloading framework among devices for web applications, *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)* pp. 41–46.

Jiao, L., Friedman, R., Fu, X., Secci, S., Smoreda, Z. and Tschofenig, H. (2013). Cloud-based computation offloading for mobile devices: State of the art, challenges and opportunities, *2013 Future Network Mobile Summit* pp. 1–11.

Khadija Akherfi, Micheal Gerndt, H. H. (2018). Mobile cloud computing for computation offloading: Issues and challenges, *Applied Computing and Informatics* **14**(1).

Khan, A. u. R., Othman, M., Madani, S. A. and Khan, S. U. (2014). A survey of mobile cloud computing application models, *IEEE Communications Surveys Tutorials* **16**(1): 393–413.

Kosta, S., Aucinas, A., Hui, P., Mortier, R. and Zhang, X. (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, *2012 Proceedings IEEE INFOCOM* pp. 945–953.

Lee, K. and Shin, I. (2013). User mobility-aware decision making for mobile computation offloading, *2013 IEEE 1st International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)* pp. 116–119.

Nguyen, D. C., Pathirana, P. N., Ding, M. and Seneviratne, A. (2020). Privacy-preserved task offloading in mobile blockchain with deep reinforcement learning, *IEEE Transactions on Network and Service Management* **17**(4): 2536–2549.

Ning, Z., Dong, P., Wang, X., Guo, L., Rodrigues, J. J. P. C., Kong, X., Huang, J. and Kwok, R. Y. K. (2019). Deep reinforcement learning for intelligent internet of vehicles: An energy-efficient computational offloading scheme, *IEEE Transactions on Cognitive Communications and Networking* **5**(4): 1060–1072.

O'Dea, S. (2020). Forecast number of mobile devices worldwide from 2020 to 2025 (in billions), *Statista* .

Qi, H. and Gani, A. (2012). Research on mobile cloud computing: Review, trend and perspectives, *2012 second international conference on digital information and communication technology and it's applications (DICTAP)* pp. 195–202.

Ran, X., Chen, H., Liu, Z. and Chen, J. (2017). Delivering deep learning to mobile devices via offloading, *Association for Computing Machinery* .

Wei, X., Wang, S., Zhou, A., Xu, J., Su, S., Kumar, S. and Yang, F. (2017). Mvr: An architecture for computation offloading in mobile edge computing, *2017 IEEE International Conference on Edge Computing (EDGE)* pp. 232–235.

Zhang, Y., Liu, H., Jiao, L. and Fu, X. (2012). To offload or not to offload: An efficient code partition algorithm for mobile cloud computing, *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)* pp. 80–86.