Democratic and Popular Republic of Algeria
الجمهورية الجزائرية الديمقراطية الشعبية
Research Scientific and Education Higher of Ministry
وزارة التعليم العالي و البحث العلمي

جامـــعة عبـــاس لـــغرور خـنـشـلة
كليـــة العلـــوم والتكنـــولـــوجيـا
University Of ABBES LAGHROUR Khenchela
Faculty of Science and Technology

N° :

A Memoir submitted to the University of Khenchela in partial fulfilment of the requirement of the Master Degree

Option : Security and Web Technology

# Energy optimization approach in Mobile Cloud Computing


Submitted by:
Mr. ATTALAH Zakaria

Supervised by:
Dr. FELLAH Hadjer


Members of the jury:

President : Dr. LEDMI Makhlouf

Examiner : Dr. SALAMA Sofiane

Academic Year: 2022-2023

# Dedication

“

*First of all, I thank Allah for giving me the courage that allowed me to accomplish this modest work.,*

*I would like to dedicate this final project to my supervisor Mrs. Fellah Hadjer , for her invaluable gaudiness throughout this journey,*

*I dedicate this final project to my beloved [Mother and Father], whose unwavering support and encouragement in my career. Their belief in me has been a constant source of motivation, ,*

*In memory of my beloved grandfather, who has passed away, I dedicate this final work as a tribute to his unwavering support, wisdom, and enduring legacy. To all those who are dear to me, to all of you.”*

*Thank you.*

”

***- Zakaria***

# Acknowledgments

I am very much indebted to my supervisor, Dr. FELLAH Hadjer. Without her invaluable suggestions, helps, patience, insightful criticisms and continuous guidance, I might not have completed this dissertation in its present shape. Special thanks are addressed to the members of the jury, Dr. LEDMI Makhlouf and Dr. SALAMA Sofiane, for their trust, support and help which were greatly needed and deeply appreciated. Special thanks to all the teachers and members of the Math and Computer Science Department who helped me in a way or in another through my educational journey.

# Summary

The increasing demand for more processing power on mobile devices has led to a situation where not all components are capable of keeping up with the speed of development. While mobile devices are being pushed to their limits, battery life has not increased at the same pace as the power requirements, creating a need for methods that can provide the required processing power while still preserving energy.

Offloading computations from the mobile device to more powerful servers in the cloud is a well-known approach for energy conservation. However, In this manuscript, we will delve deeper into the topic of Computation offloading as a well-known approach for conserving energy, particularly by transferring Computationally intensive tasks from mobile devices to more powerful servers in the cloud.

In addition, the manuscript also emphasizes the importance of application partitioning in designing an effective offloading system. Application partitioning involves dividing a given application into local and remote parts, with the goal of minimizing the total cost and optimizing energy consumption. The study focuses on how to effectively and dynamically partition the application, taking into account factors such as computation intensity and network bandwidth. By addressing these issues, the study aims to provide a comprehensive framework for designing an efficient offloading system that can improve performance while reducing energy consumption on mobile devices.

---

**Keys words :** Mobile Cloud Computing, Computation Offloading, Application Partitioning , Energy optimization.

---

# Résumé

La demande croissante de puissance de traitement sur les appareils mobiles a conduit à une situation où tous les composants ne sont pas capables de suivre la rapidité du développement. Alors que les appareils mobiles sont poussés à leurs limites, la durée de vie de la batterie n'a pas augmenté au même rythme que les exigences de puissance, créant ainsi un besoin de méthodes pouvant fournir la puissance de traitement requise tout en préservant l'énergie.

Le déchargement des applications depuis l'appareil mobile vers des serveurs plus puissants dans le cloud est une approche bien connue pour la conservation d'énergie. Cependant, dans ce manuscrit, nous allons approfondir le sujet du déchargement des applications en tant qu'approche bien connue pour la conservation d'énergie, en particulier en transférant les tâches de calcul depuis les appareils mobiles vers des serveurs plus puissants dans le cloud.

Le manuscrit souligne l'importance du partitionnement d'application dans la conception d'un système de déchargement efficace. Le partitionnement d'application consiste à diviser une application donnée en parties locales et à distance, dans le but de minimiser les coûts totaux et d'optimiser la consommation d'énergie. L'étude se concentre sur la manière de partitionner de manière efficace et dynamique l'application, en tenant compte de facteurs tels que l'intensité de calcul, la bande passante du réseau. En résolvant ces problèmes, l'étude vise à fournir un cadre complet pour la conception d'un système de déchargement efficace qui améliore les performances tout en réduisant la consommation d'énergie sur les appareils mobiles.

---

**Mots Clés :** L'informatique Mobile en nuage, Déchargement de calculs, Partitionnement d'applications, Optimisation d'énergie.

---

# ملخص

زاد الطلب على المزيد من قوة المعالجة على الأجهزة المحمولة مما أدى إلى موقف لا يمكن لجميع المكوناتفي الوقت الحاضر، أصبحت الأجهزة المحمولة جزءاً لا يمكن الاستغناء عنه في حياتنا اليومية، فهي توفر خدمات ذكية تشبه إلى حد كبير أجهزة الكمبيوتر المحمولة وأجهزة الكمبيوتر الشخصية. ومع ذلك، لا يزال هذا التوسع في الخدمات الذكية محدوداً بقيود الأجهزة المتنقلة، سواء في سعة المعالجة والتخزين، أو في عمر البطارية، وهذا يؤدي إلى صعوبة تنفيذ التطبيقات الحسابية المكثفة

بالإضافة إلى ذلك ، يؤكد المخطوط أيضًا على أهمية تقسيم التطبيقات في تصميم نظام نقل الحمل بطريقة فعالة. ينطوي تقسيم التطبيق على تقسيم التطبيق المعطى إلى أجزاء محلية وأجزاء بعيدة ، بهدف تقليل التكلفة الإجمالية وتحسين استهلاك الطاقة. يركز الدراسة على كيفية تقسيم التطبيق بشكل فعال وديناميكي ، مع مراعاة عوامل مثل كثافة الحسابات وعرض النطاق الترددي للشبكة والتأخير. من خلال معالجة هذه المسائل ، تهدف الدراسة إلى توفير إطار شامل لتصميم نظام نقل الحمل الفعال الذي يمكن أن يحسن الأداء ويقلل استهلاك الطاقة على الأجهزة المحمولة

---

**كلمات مفتاحية :**

الحوسبة السحابية،تحميل التطبيقات الحسابية،تجزئة التطبيق،تحسين استهلاك الطاقة

---

# Contents

# List of Figures

# List of Tables

# List of algorithms

# List of Abbreviation

| | |
|---|---|
| **MCC** | *Mobile Cloud Computing* |
| **NIST** | *National Institute of Standards and Technology* |
| **QoS** | *Quality of Service* |
| **API** | *Application Programming Interface* |
| **SAO** | *Service Oriented Architecture* |
| **BTS** | *Base Transceiver Station* |
| **VM** | *Virtual Machine* |
| **BSC** | *Base Station Controller* |
| **APA** | *Application Partitioning Algorithm* |
| **MSC** | *Mobile Switching Center* |
| **CO** | *Computation Offloading* |
| **QoE** | *Quality of Experience* |
| **LP** | *Linear Programming* |
| **RCMCPP** | *Resource Constrained Multi-user Computation Partitioning Problem* |

# General introduction

## Contexte

Mobile technology is increasingly dominating the market, where mobile applications have become an essential means for creating new services for mobile users. Due to emerging trends in computer technology, various applications are running on mobile systems, ranging from simple needs to high-computing requirements. These applications demand significant storage space, bandwidth, CPU power, and, most importantly, battery usage. However, application demands outpace the evolution of battery technology, which poses one of the biggest obstacles to the future growth of smartphones.

With the advancement of mobile devices, where processors have become faster, screens sharper, and devices equipped with more sensors, the energy consumption capacity of smartphones far exceeds the battery's capacity. Unfortunately, battery technology trends indicate that these limitations are here to stay, and energy will remain the primary bottleneck for mobile devices. To mitigate the energy requirements of mobile devices, the execution of applications is offloaded to remote servers in Cloud Computing, giving rise to a new technology called "Mobile Cloud Computing."

The primary objective is to utilize Cloud Computing techniques for processing and storing data from mobile devices in the Cloud. The computation offloading technique is proposed to move resource-intensive execution tasks to remote servers, typically in the Cloud. These tasks typically consume a significant amount of resources.

## Problematic

If your Android phone battery can't hold a charge for an entire day, it indicates a problem. While Mobile Cloud Computing addresses the resource shortage issue through computa-

tion offloading, it isn't the perfect solution. We still encounter the persistent challenge of energy consumption when running applications, as it significantly diminishes battery life. Employing offloading without a well-defined strategy can result in excessive energy consumption. Therefore, how can we effectively preserve battery life and optimize energy usage?

## Objectifs

Application partitioning is a technique that involves dividing an application into distinct components while preserving the original application's semantics. The goal is to find the optimal partitioning for mobile applications, determining which parts should be executed locally on the mobile device and which should be offloaded to remote cloud servers. The aim is to improve device performance and, more specifically, extend battery life.

In this work, we propose an application partitioning approach that determines the partitions to be executed in the cloud and those that remain for local execution. The objective is minimize time execution which leads to optimize energy consumption and preserve the mobile device's battery life.

## Organization of the thesis

This thesis is organized into two parts, each part contain two chapter:

- The first part is dedicated to presenting "**The State Of The Art**". Its purpose is to provide a synthesis of the various domains that we have explored within the scope of this work. It consists of two chapters:
  - In the first chapter "**Mobile Cloud Computing and Computation Offloading**", We introduce the concept of Mobile Cloud Computing, its benefits, the application domains of MCC, and computation offloading in the end we described the application partitioning.
  - In the second chapter "**Energy Optimization**": We present energy optimization and the related works conducted in this area.
- The second part is dedicated to the "**Approach Design and Realization** " and is divided into two chapters:

- In the third chapter "**Proposed Approach**": We describe the general architecture, the objectives of the proposed approach, and then delve into detailed descriptions of the different modules and algorithms proposed.
- In the fourth chapter "**Implementation and Evaluation of the Proposed Approach**":

  We then introduce the development environment used and the platform chosen for implementing our modules. Next, we take an energy-intensive application as a use case and discuss the various results to demonstrate the effectiveness of our proposal.

# Part I

# State of the Art

# Chapter 1

# Mobile Cloud Computing and Computation Offloading

“

*Our job is to ensure that Microsoft thrives in a mobile and cloud-first world … There's a challenge in mobile computing. There's an opportunity in the cloud…gotten architectures put together that really meet the needs of our customers*

”

*- a statement echoed by Microsoft co-founder Bill Gates [1]*

## 1.1 Introduction

Mobile cloud computing is an exciting field within computer science that focuses on delivering cloud computing services specifically to mobile devices. Its primary objective is to provide users with convenient access to their data and applications at any time and from anywhere. By leveraging remote servers, mobile devices connect to the internet and interact with cloud computing services through a network connection. This connectivity enables users to access their data and applications seamlessly, as long as they have an internet connection available. Furthermore, mobile cloud computing offers additional computing resources like processing power and storage to mobile devices on-demand, catering to their specific needs.

One of the significant advantages of mobile cloud computing is the ability to offload certain tasks and processes from mobile devices to the cloud infrastructure. This offloading mechanism helps alleviate the burden on the device's hardware, resulting in improved performance. Additionally, it grants users access to more robust computing resources available in the cloud, empowering them with enhanced capabilities.

## 1.2 Mobile Cloud Computing "MCC"

### 1.2.1 Definition

Many definitions have been proposed for MCC. The National Institute of Standard and Technology (NIST) defines mobile cloud computing as an infrastructure where both the data storage and the data processing happen outside the mobile device [2].

Mobile Cloud Computing "MCC" is the inevitable result of merging three IT fields: Mobile Computing, Cloud Computing and communication networks [3]. This combination has allowed to develop all aspects of the mobile phone, hardware side (computing power, memory capacities...) and software side (operating systems, application software...) from low performance to high performance and turn it into a PC with mobility functionality.

MCC "is a rich mobile computing technology that leverages unified elastic resources of varied clouds and network technologies toward unrestricted functionality, storage, and mobility to serve a multitude of mobile devices anywhere, anytime through the channel of Ethernet or Internet regardless of heterogeneous environments and platforms based

on the pay-as-you-use principle" [4], The elastic nature of the cloud enables clients to obtain the desired level of service that aligns with their changing requirements. With the availability of high-speed Internet connections, users can seamlessly access services from cloud providers. As a result, Mobile Cloud Computing (MCC) has become a more viable option compared to Mobile Cloud.

MCC applications transfer the computational power and data storage from mobile phones to the cloud, expanding the reach of applications and mobile computing beyond just smartphone users to a broader range of mobile subscribers [5]. Put simply, MCC involves running applications on resource-rich servers remotely, with mobile devices serving as thin clients. In this scenario, data is offloaded from mobile devices to the cloud for computation or storage purposes [6].

It is essential to note that in mobile cloud computing (MCC), only applications requiring significant computational resources beyond what the mobile device can handle are executed. If a user starts an application on their mobile device and monitors the usage of resources or the status of virtual machines in the cloud, this cannot be classified as MCC. Likewise, if a mobile user uses an application like Facebook hosted on a cloud server, this does not qualify as MCC.

The integration of cloud computing and mobile devices gave rise to Mobile Cloud Computing (MCC). MCC leverages the capabilities of mobile devices, such as their mobility, context awareness, and rich user interfaces, along with the scalable resources of the cloud. This integration enables mobile devices to offload computationally intensive tasks, store data remotely, and access a wide range of cloud-based services.

The relationship between cloud computing and MCC is symbiotic [7]. Cloud computing provides the infrastructure and services on which MCC relies, while MCC extends cloud computing to the mobile domain, tailoring it to the unique requirements and constraints of mobile devices. MCC utilizes the elastic nature of the cloud to dynamically scale resources according to mobile users' needs, enabling them to access on-demand services seamlessly.

### 1.2.2 Mobile Computing

Mobile computing involves performing distributed computations on various mobile devices and hybrid networks connected by mobile communication protocols. The arrival of Mobile computing has totally changed our daily lives. In what follows the main advantages of

MC [8]:

**Flexibility:** Users are no longer confined to fixed positions and can work from anywhere they desire, providing a more flexible work environment.

**Improved Productivity:** Users can work efficiently from wherever they find it comfortable which improves their level of productivity.

**Save time:** Mobile Computing allows users to access information and resources on the go, eliminating the need for physical presence or time-consuming processes. This results in time savings as users can conveniently retrieve information, utilize online services, and perform tasks from anywhere. The mobility and accessibility provided by Mobile Computing enable users to optimize their time and accomplish tasks efficiently.

**Facilitated research:** Mobile Computing provides quick and convenient access to vast amounts of information, facilitating research activities and empowering users with instant knowledge.

Inspite of the previous advantages, the MC still suffers from some limitations in terms of:

**Data storage:** The memory of mobile devices is finite. So these devices are not capable of storing an unlimited amount of information.

**Power Consumption:** In the absence of a power outlet or portable generator, mobile devices rely on battery power to function.

**Low Bandwidth:** Wireless networks offer lower bandwidth compared to wired networks.

**Security:** Due to its nomadic nature, wireless communication is more vulnerable to security risks and issues due to the ease of breakage in wireless links. It is not easy to monitor correct use. Users may have different intentions on how to use this privilege, hacking, industrial espionage, online fraud and malicious destruction are some issues faced by mobile computing

**Disconnect:** Network failure can cause mobile devices to disconnect, affecting service quality.

### 1.2.3 Cloud Computing

Cloud computing is the core of MCC [9], it is a system for providing IT services via the internet instead of relying on a traditional in-house infrastructure. The idea of cloud computing involves utilizing a group of external servers, also known as "the cloud," to store, manage, and process data and applications. By utilizing cloud computing, individuals can access their data and applications from any device with internet access, as opposed to being restricted to a single computer or device.

Based on the NIST definition of cloud computing, the concept enables a group of adaptable computing resources (such as networks, servers, storage, applications, and services) to be shared and deployed rapidly with minimal management effort or service provider involvement. This cloud model includes three service models, four deployment models, and five essential characteristics.[10]

1. **Essential Characteristics:**

   (1) *Broad network access:* The accessibility of functionalities through the network, and their availability through shared mechanisms, promote their use across a range of platforms, whether thin or thick clients. These platforms can include devices such as workstations, laptops, tablets, and mobile phones.

   (2) *Resource pooling:* The multi-tenant cloud computing model involves pooling the provider's computing resources to serve multiple consumers, dynamically assigning and reassigning physical and virtual resources based on consumer demand. Although customers may be able to specify location at a higher level of abstraction, they generally need more control or knowledge over the exact location of the provided resources, resulting in location independence.

   (3) *Rapid elasticity:* In order to quickly increase or decrease in size in response to the degree of demand, capabilities can be dynamically assigned and deallocated. The capabilities that are accessible for allocation appear limitless to the user and are always available in whatever amount.

   (4) *Measured service:* Cloud systems optimize resource usage through automatic control mechanisms, utilizing metering capabilities appropriate to the service provided (such as storage, processing, bandwidth, and active user accounts). This allows for monitoring, control, and reporting resource usage, promoting transparency for the provider and the service consumer.

(5) *On-demand self-service:* If needed, a user can autonomously allocate computing resources, such as server time and network storage, without communicating with each service provider.

2. **Service Models:**

(1) *Software as a Service (SaaS):* Individuals or organizations who depend on cloud computing for common office or productivity services. (e.g., accounting tasks)

(2) *Platform as a Service (PaaS):* Individuals who depend on cloud computing to meet their business intelligence requirements (e.g., application integration and database management).

(3) *Infrastructure as a Service (IaaS):* consumers who rely on cloud computing for Information Technology needs.

3. **Deployment Models:**

(1) *Private cloud:* The cloud infrastructure and resources are either hosted locally by the service provider or remotely by a third party, and are exclusively accessible to a single customer.

(2) *Community cloud:* Cloud infrastructure and resources are generally available to customers within the same sector or with similar security concerns or requirements.

(3) *Public cloud:* Diverse users, including the general public and multiple subscribers, can utilize most cloud infrastructure and resources.

(4) *Hybrid cloud:* There are several packaged distribution models available for accessing cloud infrastructure and resources.

This Figure 3.1 present a general description of the Cloud Computing following NIST:

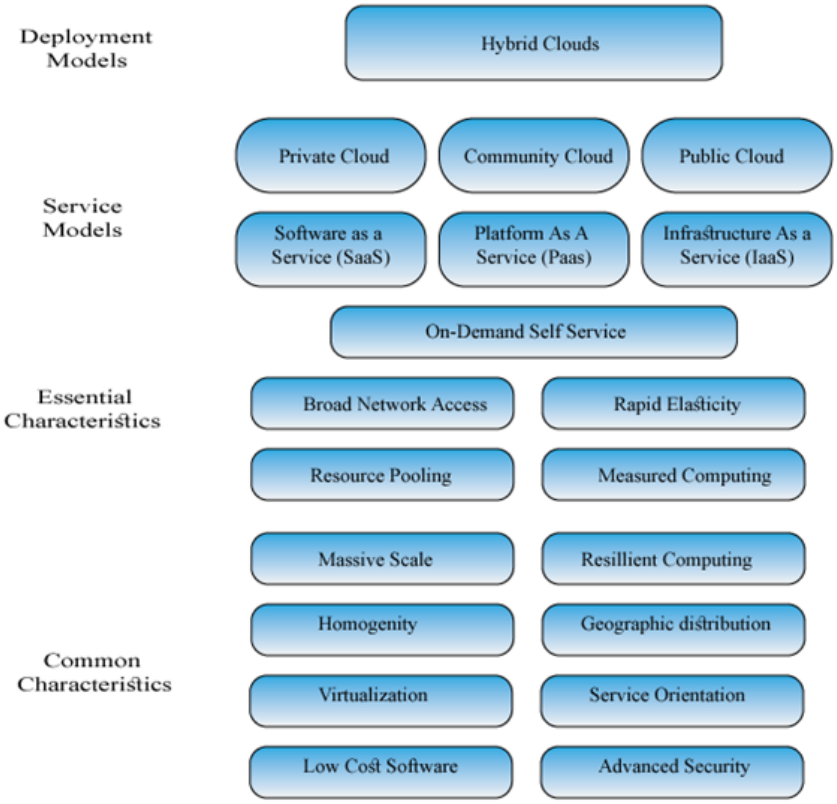


Figure 1.1: Description of Cloud Computing according to NIST. [11]

The table 1.1 below provides several examples of service-oriented cloud computing architectures:

Table 1.1: Examples of Service-Oriented Cloud Computing Architectures

| Architecture | Examples |
|---|---|
| Infrastructure as a Service (IaaS) | Amazon Web Services (AWS) EC2, Microsoft Azure Virtual Machines, Google Compute Engine |
| Platform as a Service (PaaS) | Microsoft Azure App Service, Google App Engine, Heroku |
| Software as a Service (SaaS) | Salesforce, Google Workspace, Microsoft Office 365 |

Therefore, the network is a means of linking the communication between Mobile Computing and Cloud Computing to transfer and consume the resources, data and information. This communication has evolved over time from the first generation (1G) to the fifth generation (5G).

Table 1.2: Evolution of wireless technologies 1G to 5G in mobile communication [12]

| | 1G | 2G | 3G | 4G | 5G |
|---|---|---|---|---|---|
| Year of appearance | 1985 | 1992 | 2002 | 2010-2012 | Starts from 2020 |
| Radio frequency(Hz) | 400M-800M | 800-900M | 2G | 3G-4G | Plus than 4.9G |
| Bandwidth(bps) | 2.4K-3.0K | 9.6K-14.4K | 2M-5M | 10M-20M | Plus than 1G |
| Cellular Coverage | Large area | Medium area | Small area | Mini-Area | |
| Core networks | Telecom networks | Telecom networks | Telecom networks | IP Networks | IP Networks |
| Service type | Voice | Voice, SMS | Voice, Data, multimedia | Multimedia | |
| Driven technique | Analog signal processing | Digital signal | Smart signal processing | Intelligent Signal auto configuration | Packet switching |

After giving the definition of the main fields evolved in Mobile Cloud Computing, the latter can be resumed in the figure 1.2:

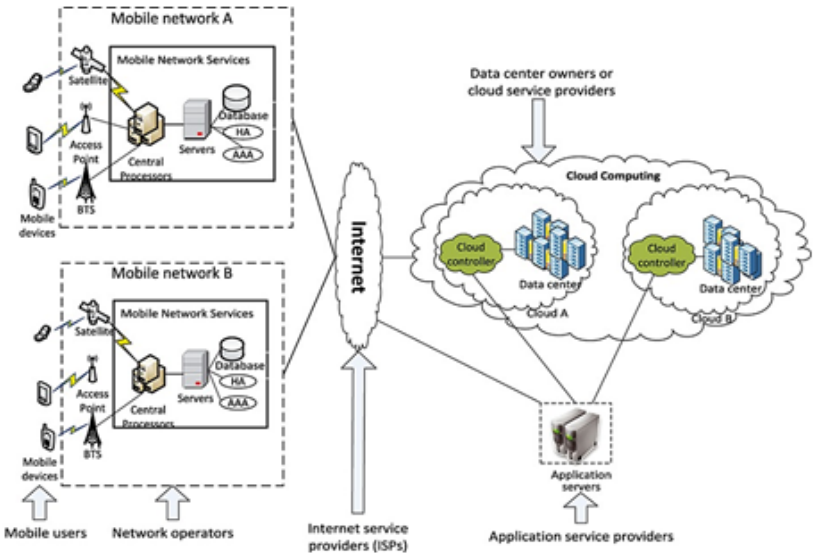


Figure 1.2: Mobile Cloud Computing Architecture. *[13]*

### 1.2.4 Advantages of mobile cloud computing

Cloud computing is a feasible substitute for mobile computing due to several factors, including portability, communication, and mobility. The benefits of mobile cloud computing (MCC) are emphasized in the following explanation of how the cloud can help overcome the limitations of mobile computing.

1. **Extending battery lifetime:** The need for enhanced mobile device performance clashes with the need for extended battery life. A potential solution to this issue is to transfer computational tasks to powerful servers, which can lessen energy usage and boost performance for mobile applications. This approach has become feasible due to the widespread availability of high-speed wireless connections such as 4G and Wi-Fi, which can help conserve a mobile device's battery life. [14]

2. **Accessibility:** Mobile cloud computing enables users to access their data and applications from any location and device, as long as there is an internet connection.

3. **Cost-effectiveness:** By using cloud-based services, users can avoid the high costs associated with maintaining and upgrading on-premise hardware and software.

4. **Increased security:** With mobile cloud computing, data is stored in highly secure data centers, protected by multiple layers of security. This reduces the risk of data loss or theft compared to traditional on-premise solutions.

5. **Improved mobility:** Mobile cloud computing enables users to work from anywhere and on any device, leading to increased productivity and flexibility.

6. **Green computing:** By reducing the need for on-premise hardware and software, mobile cloud computing can lead to reduced carbon emissions and a more sustainable computing environment [4].


Figure 1.3: Advantages of mobile cloud computing .
*[15]*

## 1.3 Computation Offloading

### 1.3.1 Definition

Computation offloading refers to a mechanism of executing the whole application (or part of it), that exceeds the mobile device's capabilities, on resourceful servers in the Cloud instead of mobile device. Consequently, the battery life of the device is increased as well as the difficulties of storage and resource limitations are removed.

Furthermore, computation offloading eliminates the challenges posed by resource limitations on mobile devices. Mobile devices typically have limited processing power, memory capacity, and other resources. By leveraging the computational capabilities of the Cloud servers, mobile devices can tap into the vast resources available, enabling them to execute demanding applications and services that would otherwise be impractical or impossible to run on the device alone.

### 1.3.2 Evolution of Computation Offloading

Researchers have been exploring computation offloading, mainly focusing on deciding when and developing the necessary infrastructure to enable offloading. Before 2000, the primary emphasis of researchers was on making computation offloading a viable option[16] [17]. During the early 2000s, the focus shifted towards developing algorithms that could determine when offloading was necessary. Advancements have influenced the determination of whether offloading would be advantageous for mobile users in virtualization technology, network bandwidths, and cloud computing infrastructures, which have also altered the course of offloading. [18] The progress in technology has made it more feasible to implement computation offloading (Fig 1-4) [19].

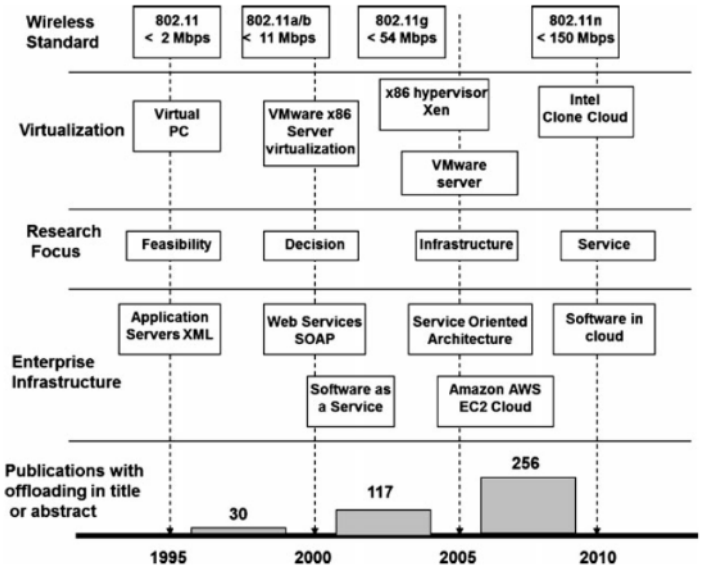


Figure 1.4: Enabling technologies for computation offloading. *[14]*

### 1.3.3 Computation offloading approaches

Over the past 20 years, numerous efforts have been made to minimize the energy usage of mobile devices by utilizing computation offloading. Generally, these efforts can be categorized into one of two approaches [19]:

The first approach requires programmers to specify how to partition a program, what computations should be offloaded, and how to adjust the offloading strategy based on

changes in network conditions. This approach is considered fine-grained, as it allows applications only to offload specific sub-parts that would benefit from remote execution, resulting in significant energy savings. For instance, a video player typically has an energy-intensive decoder; offloading this decoder can significantly reduce the energy consumption of the mobile device while playing videos.

The second strategy involves transferring the entire process or virtual machine (VM) to the infrastructure. For instance, in Android, each application operates in a process that functions in a particular VM. This approach can reduce the workload on programmers, as they do not need to modify applications. Instead, their code and program state is automatically moved to the remote infrastructure. However, this method has the drawback of incurring high migration costs, as all program states must be transferred. Most computation offloading systems aim to accomplish two primary objectives:

- Reduce energy consumption for mobile devices.
- Enhance application performance.

### 1.3.4 Making Offloading Decisions

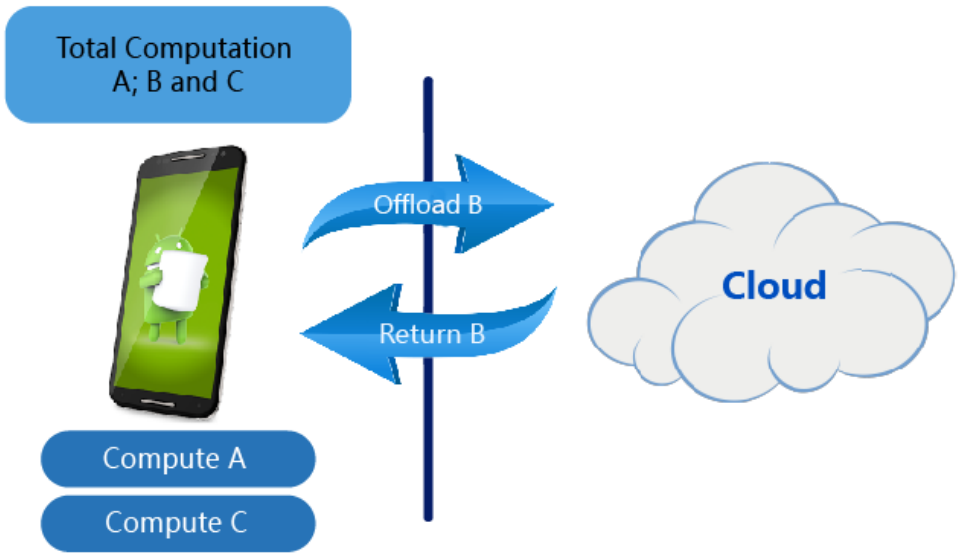


Figure 1.5: Computation process performed using offloading to cloud. [4]

To minimize the execution time and energy consumption, computation offloading from a mobile device to a server machine is performed by applying specific criteria to ensure that the offloading will be advantageous. [20] [21] [22] The necessary criteria considers

various parameters, as explained below. To reduce the execution time, the overhead of runtime activities, which includes the time for data transfer and offloading code, is denoted as $O_r$.

$$O_r = T_d + T_o \tag{1.1}$$

Where, $T_o$ time of data transfer and $T_o$ time for offload the code, and $T_s$ The time taken to execute code on the cloud, $T_s$ The time taken to execute code on the mobile device.Computation offloading is deemed effective in minimizing execution time if the following conditions are met:

$$T_s + O_r < T_m \tag{1.2}$$

Likewise, to reduce energy consumption, $E_d$ denotes the energy required for data transfer, while $E_o$ represents the energy required for offloading. $E_m$ represents the energy required for the entire application's execution on the mobile device, and $E_r$ denotes the energy required for runtime activities. Computation offloading is effective in reducing energy requirements if the following conditions are met:

$$E_r < E_m \tag{1.3}$$

While $E_r$:

$$E_r = E_d + E_o \tag{1.4}$$

### 1.3.5 The Determinants Impacting the Offloading

The potential of mobile offloading depends mainly on mobile network technologies such as cellular and Wi-Fi networks. Unlike Wi-Fi, data transmission using the cellular network requires a considerable amount of energy from the mobile device, which offers high-bandwidth connections. Therefore, computations offloading should only be done when local execution consumes more time and energy than remote execution.
Many factors can influence the offloading decision and could impact the offloading process,

the Table shows these factors:

Table 1.3: Factors affecting the offloading decision

| Factor | Value |
|---|---|
| Application Type | delay-tolerant, Delay-sensitive |
| Mobility | Distance between server and MD |
| Mobile Device Specification | CPU Speed, RAM, Battery... |
| Server Specification | CPU, RAM, laod... |
| Wireless Environments | 4G, 5G, WIFI ... |

*[23]*

### 1.3.6 Offloading Architectures and Approaches

We can classify computation offloading methods into two categories, static and dynamic, based on when the decision to offload is made.

#### 1. Static Offloading

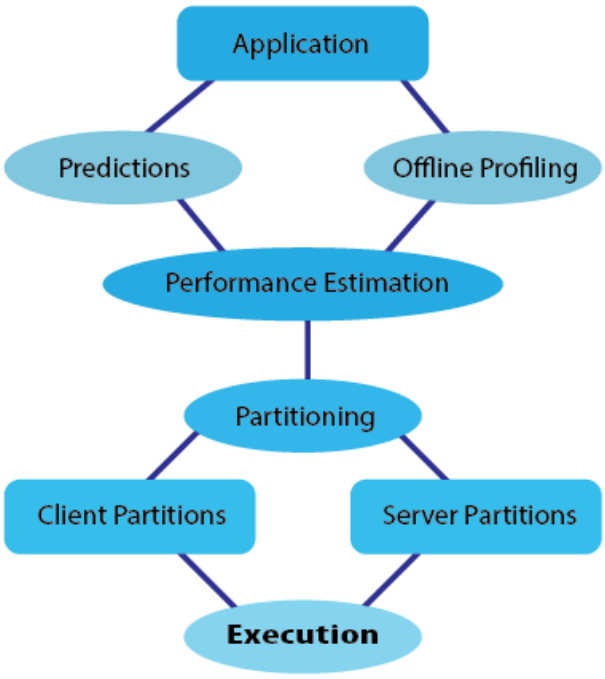


Figure 1.6: Static Offloading Mechanism.
*[24]*

The use of performance prediction models or offline profiling is employed in the static offloading approach, as demonstrated in Figure 1.6, to estimate the performance. This

then results in the application being partitioned into client and server partitions that can be executed at a later time [20] [21] [25].

### 2. Dynamic Offloading

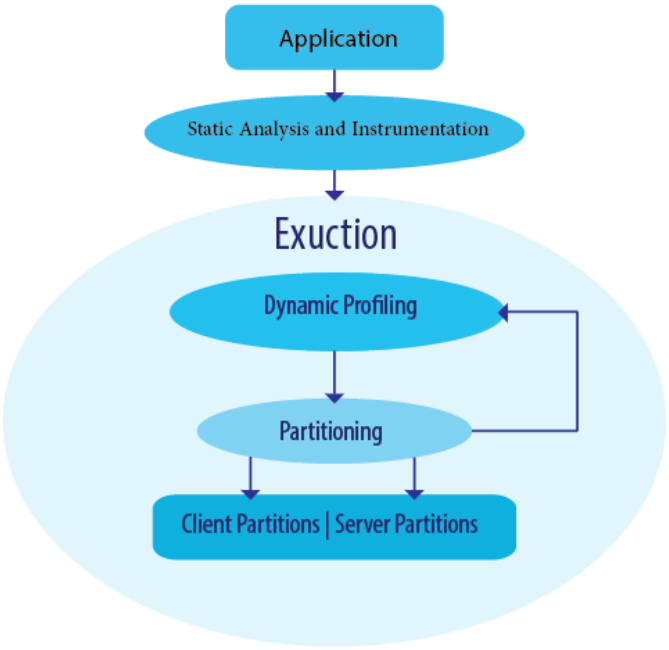


Figure 1.7: Dynamic Offloading Mechanism .
*[24]*

The dynamic offloading strategies depicted in Figure 1.8 start with a static analysis of the code and instrumentation, enabling them to conduct dynamic profiling during the execution phase. The application is subsequently divided into client and server partitions based on the data obtained from the dynamic profiling. Following this, the execution resumes with the updated configuration [26][27].

### Computation Offloading Process

The process of Computation Offloading consists of three main steps, as follows [28]:

- **Application partitioning:** Seeks to divide the mobile application into components, with some being moved to the cloud server for remote execution and others remaining on the mobile device for local execution.
- **Preparation:** The preparation stage accomplishes all the steps necessary for the dischargeable components to be used in mobile applications. This involves choosing

a remote server, transferring and installing code, and so on.

- **Offload Decision:** The final step before remote execution of the offloadable components begins is to make an offload decision.

## 1.4 Application Partitioning

### 1.4.1 Definition

This is a technique that allows the application to be divided into several tasks. These tasks are classified into two parts: a first part will be executed on the mobile terminal and a second part that will be executed on Cloud servers [29]:

- **Unoffloadable Tasks:** Some tasks must be executed locally unconditionally on the mobile device, either because transferring relevant information takes too much time and energy or because these tasks need to access local components or sensitive and confidential tasks.
- **Offloadable Tasks** Some components of the application are flexible tasks that can be processed either locally on the mobile device's processor or remotely in a cloud infrastructure. Many tasks fall into this category.

When it comes to components that cannot be offloaded, there is no need to make offloading decisions. However, when it comes to components that can be offloaded, it's important to consider which tasks should be executed locally on the mobile device and which ones should be offloaded to the remote cloud for execution, taking into account factors such as available networks, response time, and energy consumption. The mobile device will need to make an offloading decision based on the outcome of a dynamic optimization problem.

### 1.4.2 Type of partitioning

In addition to the types of partitioning based on the nature of the application, two broad categories of partitioning techniques in computation offloading are [19][24]:

- **Static partitioning:** In this technique, the partitioning of tasks is done before the application is executed, and the same partitioning is used throughout the execution.

It is a fixed partitioning that does not adapt to changes in the application or network conditions during runtime.

- **Dynamic partitioning:** As the name suggests, in this technique, the partitioning of tasks is done dynamically during runtime based on the current state of the application and network conditions. The partitioning can be adapted to changes in the application or network, which makes it more flexible and efficient than static partitioning.

The decision between static and dynamic partitioning techniques in computation offloading depends on various factors such as the application's characteristics, the available resources, and network conditions. While dynamic partitioning is more flexible and adaptable, it comes with additional processing overhead and may lead to latency issues due to the dynamic nature of the partitioning. Static partitioning, on the other hand, is less flexible but requires less processing overhead and may result in lower latency.

### 1.4.3 Taxonomy of application partitioning algorithms for MCC

Figure 1.8 depicts the thematic taxonomy that is utilized for the classification and comparison of APAs, and it involves several classification parameters.

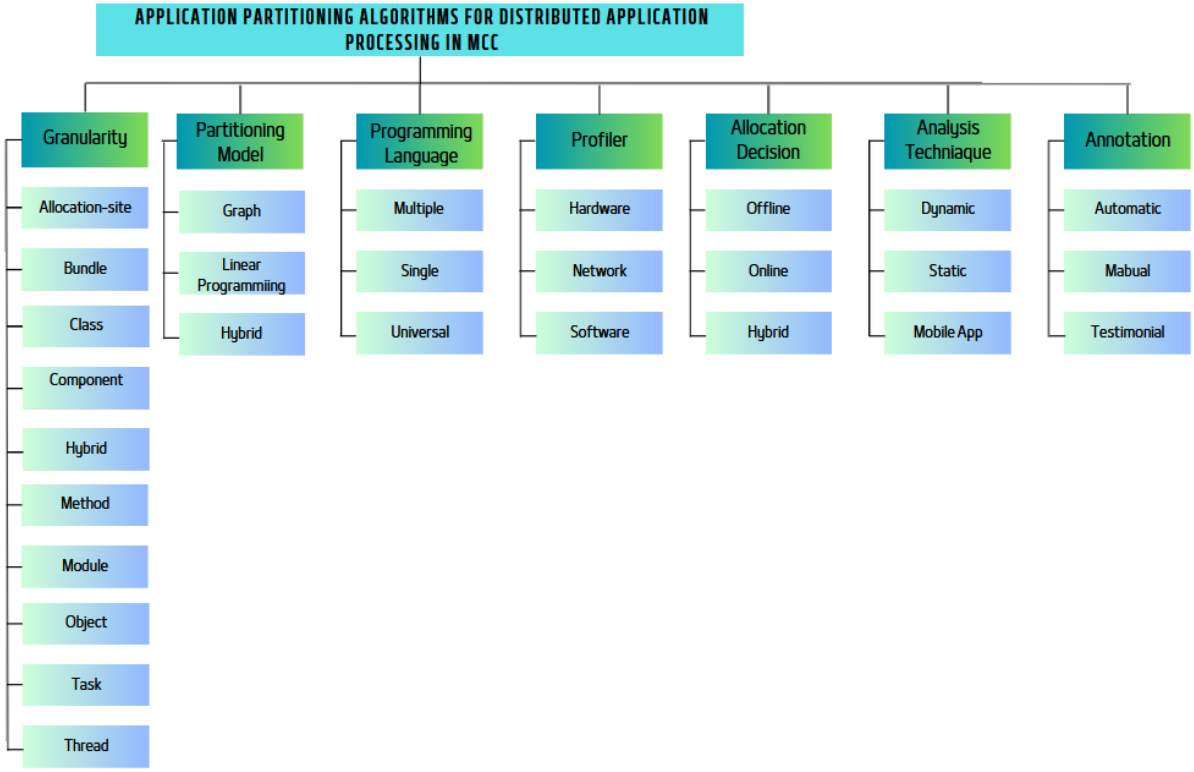


Figure 1.8: The thematic taxonomy pertains to the algorithms used for partitioning applications in distributed processing within MCC.
*[30]*

- **Granularity:**"Indicate the level of granularity for partitioning a mobile application. Current APAs (Application Partitioning Algorithm) implement the following levels of granularity for partitioning an application: Module, Method, Object, Thread, Class, Task, Component, Bundle, Allocation-site."

- **Modele:** Shows the type of partitioning model used to model the components of a mobile application. Current APAs (Application Partitioning Algorithm) implement the following models:

  - *Graph:* APAs model the entire partitioning process as a graph.
  - *Linear Programming:* APAs formulate a linear equation to represent the partitioning of the application.

  - *Hybride:* APAs combine the graph-based model and the LP model for implementing the partitioning application.

- **Programming languages:** This shows the programming language type that is compatible with the partitioning approach.

  - *Single Programming Language:* Programming in a unique language.
  - *Multi-Languages:* Supports multiple programming languages.
  - *Universal Programming Language:* Supports the majority of programming languages.

- **Profiler:** The profiler indicates the type of profiling used, current APAs use the following profilers:

  - *Hardware Profiler:* CPU, RAM, BATTERY
  - *Software Profiler:* It collects application information such as size of accessible data, interdependence between modules, application behavior, performance cost (execution time, throughput, and latency), and code size.
  - *Network Profiler:* WiFi, 3G, 4G...

- **Allocation Decision:** This attribute indicates the decision-making policy of the APA in allocating components locally or to a remote server. The represented decision can be:

  - *Offline:* the decision is made before execution.
  - *Online:* The decision is made at execution.
  - *Hybrid:* A portion of the decision is made by a specification defined by the programmer or a static analysis tool, and a portion of the decision is made during the execution of the application.

- **Analysis Techniques:** The analysis technique of the APA is the technique used for identifying dependency relationships among the components of a mobile application. It is classified as (Static and Dynamic)

- **Annotation:** an annotation is a syntactic metadata that is added to the source code of the application to provide additional information about its components or behavior. These annotations are used by the partitioning algorithm to make

decisions about how to partition the application. They can include information about the size and dependencies of components, as well as other performance-related metrics, annotations can be classified into three main types based on how they are created:

- *Manual annotations:* are annotations that developers create and add to the code themselves. While this type of annotation requires more work, it also offers more precise and comprehensive information about the application.
- *Automatic annotations:* refer to annotations that are generated through automated tools or algorithms that analyze the application code. This type of annotation may be less precise than manual annotations, but it requires less effort from the developer.
- *Testimonial annotations:* are added to the code by users or testers of the application, who provide feedback based on their experience. While this type of annotation can be beneficial in providing insight into the application's performance and behavior, it may not be as objective or dependable as manual or automatic annotations.

## 1.5 Conclusion

In this chapter, we have explored the concept of Mobile Cloud Computing (MCC) and its potential benefits, particularly in the realm of Computation Offloading (CO). We began by defining MCC and highlighting some of its key advantages. MCC is a rapidly growing field that leverages the power and resources of cloud servers to enhance the performance of mobile devices. By offloading computation tasks to the cloud, We then introduced the concept of CO, discussing its architecture and how it can be used to optimize the performance of mobile devices.

We also discussed the decision-making process involved in CO. Furthermore, we explored the different types of application partitioning, including static and dynamic partitioning, and their associated benefits and drawbacks. Finally, we provided a taxonomy of application partitioning algorithms for MCC.

In the next chapter, we will delve deeper into this topic and discuss the different algorithms and techniques that can be used for computation offloading. We will also explore the various factors that impact the performance of these algorithms, and provide

insights into the challenges involved in their implementation.

By understanding the fundamentals of Mobile Cloud Computing and computation offloading, we are better equipped to design energy-efficient and high-performance mobile applications that can provide a seamless user experience while preserving battery life.

# Chapter 2

# Energy Optimization

## 2.1 Introduction

Optimizing energy consumption is a crucial aspect of mobile computing, as it allows for longer battery life and improved device performance. Computation Offloading (CO) has emerged as a promising technique for reducing energy consumption, with cloud servers providing a powerful resource for mobile devices to utilize.

In this chapter, we will explore different approaches for optimizing energy usage, focusing specifically on the Application Partitioning Algorithms (APA) that can be used in CO. We will examine different types of APA algorithms, including graph models, LP models, and hybrid approaches.

By exploring the potential of APA algorithms and other optimization techniques to reduce energy consumption and enhance the performance of mobile devices, we can develop new strategies and approaches that can transform the way we interact with technology. As mobile devices continue to play an increasingly important role in our daily lives, the need for energy-efficient computing will become ever more important, making the exploration of optimization techniques a critical area of research and development.

## 2.2 Energy Optimization

The primary goal of offloading systems is to reduce energy consumption on mobile devices, which can also lead to better application performance by using more powerful cloud machines. However, many factors can affect energy consumption, such as execution time, latency, network conditions, bandwidth, and packet drops.

The study's authors [31] conducted a formative analysis to examine how mobile computation affects performance and energy conservation. They investigated the impact of input size, bandwidth, and network conditions on offloading performance. The findings indicate that offloading yields better performance and energy conservation on fast networks (like WiFi and LTE). However, the benefits are not as straightforward on slower networks, as they depend on the complexity of the task being offloaded.

Recently, the use of mobile devices, particularly smartphones, has increased rapidly, increasing the demand for better Quality of Service (QoS) and Quality of Experience (QoE). Although mobile devices effectively satisfy user demands, they require methodologies to improve their efficiency and battery life. Mobile Cloud Computing (MCC)

allows for offloading data storage and application execution from mobile devices to cloud servers. Offloading effectively solves the battery problem, but it is not free and incurs overhead costs, such as the cost and energy required for communication between mobile devices and the cloud. Some message transfers are necessary between the cloud and the mobile device to establish communication with the cloud. These message transfers incur additional costs and energy consumption, so it is necessary to study the overhead costs incurred for data storage and application execution [4]. Therefore, to avoid another energy optimization problem when offloading to the cloud, it is necessary to study energy optimization algorithms to manipulate offloading better and use it correctly with desirable results, It is necessary to study energy optimization algorithms to manipulate offloading better and use it correctly with desirable results.

## 2.3 Energy Optimization Algorithms

In this part, an overview of the existing APAs is provided, and their implications and other significant factors are discussed based on the various application partitioning models presented in the taxonomy. The present APAs can be categorized into three models, including graph-based, LP-based, and hybrid-based models.

### 2.3.1 Graph-based application partitioning algorithms

Typically, the components of a graph - vertices and edges - are utilized to depict the context or parameters of an application. This involves available resources, data size, communication overhead, computation, and memory costs. APAs employ the graph model to model execution states, cost models, internal dependency, data flow, and control flow.

- **Offloading Inference Engine (OLIE):**

  Is a Java program designed for mobile devices, which is used to distribute Java classes without native states . The algorithm functions by dividing Java classes into groups using customized min-cut heuristic techniques, with the aim of minimizing component interactions among the partitions. The algorithm partitions a class graph into potential partition plans based on edge-weight, and selects the optimal plan by comparing the combined metrics. However, the min-cut heuristic technique may miss superior partitioning solutions in its candidate partition plans, as most

of the resource constraints are related to vertex-weights or memory consumption. The decision criterion is solely based on memory and does not take other factors into account. This approach also does not consider partitioning constraints like CloneCloud, resulting in a coarse granularity of partitioning at the class level and a focus on static partitioning.

- **Hybrid Granularity Graph (HGG):**

  To minimize network overhead, a Hybrid Granularity Graph (HGG) has been proposed. In HGG, a vertex is linked to a configurable subset of objects of a given class instead of being linked to a single runtime component (object or class). A profiler collects information on performance cost, memory utilization, and runtime coupling patterns via bytecode injection, which is then used to create runtime graphs. Compared to an object-level graph, HGG provides a finer level of adaptation and more effective object topologies without incurring computational overheads. Furthermore, HGG offers finer granularity than a class graph, providing greater flexibility. HGG is computationally feasible because it is smaller than an object graph.

- **Distributed Abstract Class Graph:**

  Ermyas Abebe and Caspar Ryan [32] proposed an algorithm based on an abstract class graph. In this algorithm, the devices maintain graph vertices for components within their memory space, and cloud-vertices, which are abstraction vertices, for components in remote devices. The algorithm uses a multi-level graph partitioning heuristic to reduce network, power supply, memory utilization, and performance cost of adaptive offloading. The algorithm also allows the partitioning result on the cloud to be loaded back to the client device for utility purposes. The partitions generated by this algorithm are more effective, and the remote object coupling and migration costs are reduced as well.

### 2.3.2 Linear programming-based application partitioning algorithms

Linear programming (LP) is a mathematical approach that aims to find the best possible outcome given a set of constraints expressed as linear equations in a mathematical model. It is a method to optimize a linear objective function, subject to linear equality and linear inequality constraints, and is commonly used for solving challenging problems. The main

benefit of LP is that it always produces optimal results for a specific objective function. Nevertheless, solving LP problems can take time due to a large amount of computation [33].

- **Mobile Augmentation Cloud Services (MACS):**

  Dejan Kovachev and Ralf Klamma[34] proposed a middleware called Mobile Augmentation Cloud Services (MACS) to reduce local execution time and battery power consumption. MACS allows Android applications to be extended into the cloud adaptively. It is responsible for application partitioning, resource monitoring, and computation offloading. The Elastic applications function like regular mobile applications but can use remote computing resources. MACS can offload multiple Android services within a single application by determining the allocation from the resource monitor. MACS uses a dynamic partitioning scheme and lightweight profiling during runtime for adaptive partitioning decisions [33].

- **RCMCPP:**

  Lei Yang, Jiannong Cao, Hui Cheng, and Yusheng Ji [35] conducted a study on the problem of application partitioning to ensure the benefits of service providers by evaluating constraints on cloud resources. The researchers first considered a problem of partitioning computation for a single user, known as the single-user computation partitioning problem (SCPP). Then they derived the resource-constrained multi-user computation partitioning problem (RCMCPP). To minimize the total execution time, the RCMCPP was formulated as a mixed-integer linear programming (MILP) problem. To make a tradeoff between application performance and the cost of cloud resources, application providers used the performance-resource-load (PRL) model to design a resource provisioning mechanism to improve the utilization of cloud resources [33].

### 2.3.3 Hybrid application partitioning algorithms

The hybrid application partitioning algorithms (APAs) integrate the characteristics of both graph-based and LP-based techniques. They aim to extract the essential features of both methods to enhance the performance of APAs.

- **Mobile Assistance Using Infrastructure (MAUI):**

  Eduardo Cuervo et al [36] proposed a method for solving graph optimization in application partitioning by representing the application as a call graph and using 0-1LP. This approach allows for fine-grained code offloading to optimize energy usage with minimal involvement from the programmer. Conversely, MAUI enables programmers to produce an initial partitioning of the application by annotating its local and remote components. A key feature of MAUI is its use of managed codes to ease the burden on programmers when dealing with program partitioning while maximizing energy savings during computation offloading [33].

- **CloneCloud:**

  CloneCloud employs a control flow graph to model the application, which is optimized using ILP. It conducts an offline static analysis of various running scenarios of the process binary on both a smartphone and the cloud. The analysis results are utilized to create a database containing pre-computed partitions of the binary, determining which sections should be offloaded to the cloud. CloneCloud is an effective tool for dynamically partitioning Java code based on static analysis. However, it is limited to the input and environmental conditions that are used during the offline pre-processing. To use it with a new application, the analysis must be repeated. Additionally, ordinary users may need help to customize the application's functionality as it requires the support of programmers to annotate the components [33].

- **Wishbone:**

  Wishbone is an APA that adopts a profile-based strategy to partition applications. It models the application as a data flow graph of operators to facilitate execution on multiple and diverse devices in a sensor network. It prioritizes high-rate data processing applications and uses a static approach to minimize network bandwidth and CPU load during compilation by solving an ILP problem. On the other hand,

Wishbone utilizes WaveScript to define the data processing pipeline, allowing for adaptable responses to modifications in the operating environment.

## 2.4 Exceptional application partitioning algorithms

Exceptional APAs differ from other types of application partitioning algorithms in that they do not model the application as a graph or LP equation. Instead, these algorithms prioritize either manual or automatic annotation before carrying out application partitioning. They require a relatively higher level of input effort from application developers to produce effective partitioning solutions. Despite this, exceptional APAs can still generate optimal results similar to other APA types.

- **Cyber foraging solution:**

  The Cyber foraging solution introduces an adaptive runtime system that utilizes Vivendi [37], a little language, to provide application partitioning. Programmers analyze the application's source code and create a tactics file that outlines the function prototype of each procedure that can be executed remotely. This file also specifies how these procedures are combined to generate the partition result. However, it is essential to note that the application developers must create a tactics file for each application, which can be time-consuming and error-prone [33].

- **J-Orchestra:**

  J-Orchestra is a system that automates the partitioning of Java programs, dividing them into distributed applications that run on separate Java virtual machines. By utilizing bytecode rewriting, J-Orchestra replaces method calls with remote method calls and converts direct object references into proxy references. Programmers need only specify the network location of hardware and software resources, along with the corresponding application classes, requiring minimal effort. JOrchestra recognizes the potential impact of annotation errors on the efficiency and accuracy of distributed applications. To address this concern, J-Orchestra supplies two tools—a profiler and a classifier—to ensure precise and efficient partitioning. When compared to previous approaches, J-Orchestra offers significant benefits in terms of adaptability, flexibility, and automation [33].

## 2.5 Comparison of application partitioning algorithms by using thematic taxonomy

Table 3.1 displayed a comparison among various APAs that we went through for MCC using the parameters outlined in Figure 1.8. One of these parameters is the Partitioning Granularity (PG) attribute, which denotes the level of granularity for partitioning computational-intensive mobile applications.

Table 2.1: Comparison of application partitioning algorithms.

*[33]*

| APAs | PG | PM | PLS | PR | AD | AT | AN |
|---|---|---|---|---|---|---|---|
| OLIE [38] | Class | Graph | Single | software, Network | Online | Dynamic | Manual |
| HGG [39] | Hybrid | Graph | Single | Software, Hardware | Online | Static | Automatic |
| Distributed Abstract Class Graph [32] | Class | Graph | Single | N/A | Offline | Dynamic | N/A |
| MACS [34] | Bundle | LP | Single | Software, hardware | Online | Dynamic | Automatic |
| RCMCPP [40] | Module | LP | N/A | Network, Hardware | Online | Dynamic | Automatic |
| MAUI [41] | Method | Hybrid | Single | All | Online | Dynamic | Manual |
| CloneCloud [42] | Thread | Hybrid | Multiple | Network, Hardware | Online | Static | Automatic |
| Wishbone [43] | Thread | Hybrid | Multiple | Network, Hardware | Offline | Static | Manual |
| Simplifying Cyber Foraging [37] | Module | Exceptional | Universal | N/A | Offline | Static | Manual |
| J-Orchestra [44] | Object | Exceptional | Single | Software | Offline | Dynamic | Manual |

Note: **PG:** Partitioning Granularity, **PM:** Partitioning Model, **PLS:** Programming Language Support, **PR:** Profiler, **AD:** Allocation Decision, **AT:** Analysis Technique, **AN:** Annotation

## 2.6 Conclusion

In conclusion, Chapter 2 has highlighted the significance of energy optimization in mobile cloud computing and its impact on improving the performance of MCC systems. The review of various application partitioning algorithms and their effectiveness in reducing energy consumption has provided a comprehensive understanding of the current state of research in this area. Overall, the insights and analysis presented in this chapter have significant implications for the development of MCC systems that are energy-efficient, high-performing, and sustainable.

# Part II

# Approach Design and Realization

# Chapter 3

# Proposed Approach

## 3.1 Introduction

Mobile Cloud Computing leverages the power of cloud servers to process data and complex computational tasks, eliminating the need for mobile devices to have high-end configurations like fast processors and large memory capacities. By offloading these tasks to the cloud, the performance of mobile devices is enhanced, and energy consumption is optimized.

To achieve optimal energy consumption while offloading mobile application tasks, it is crucial to accurately determine which tasks should be offloaded. This presents a significant challenge.

In this chapter, we present a dynamic partitioning mechanism as a solution. This mechanism aids in making informed decisions regarding offloading computational tasks, resulting in efficient energy management for energy-intensive applications. By dynamically partitioning tasks between mobile devices and cloud servers, our approach ensures the best utilization of resources and maximizes energy efficiency.

## 3.2 General description of the proposed approach

Our approach takes advantage of a clever combination of LP and graph models, resulting in a hybrid algorithm that offers several benefits. By incorporating an intelligent selection mechanism, we enhance the likelihood of generating a correct solution that can effectively partition and offload tasks to the Cloud. This selection mechanism allows us to explore various possibilities and find feasible solutions based on informed criteria or optimized algorithms.

The primary goal of our approach is to reduce the execution time of different tasks within a mobile application. By doing so, we optimize the energy consumption of the mobile device. This is achieved through strategic task offloading to the Cloud, which provides access to powerful and energy-efficient computational resources, surpassing the limitations of mobile devices.

To achieve optimal and dynamic results, we consider multiple factors, including local costs on the mobile device, remote costs in the Cloud, and communication costs between the device and the Cloud. These costs play a vital role in determining the most efficient approach to task partitioning and offloading.

To facilitate the decision-making process, our approach incorporates various modules such as the profiler, analyzer, and solver. The profiler module gathers essential information about the tasks and their specific resource requirements. The analyzer module then evaluates this information, taking into account the aforementioned costs. Finally, the solver module utilizes the analyses performed by the previous modules to determine the best possible partitioning result.

In summary, our proposed approach closely follows the task offloading steps outlined in the previous chapter.

- **Application Decomposition:**

  Decompose the Android application into individual methods using a specified algorithm. This step involves determining which methods are invoked by each method. In parallel, establish a weighting scale for each method based on its dependencies with other methods within the application.

- **Performance Profiling:**

  Conduct a performance profiling of the application, focusing on three major resources: CPU, battery, and network. The aim is to derive the execution time for both mobile and cloud environments.

- **Offloading Decision:**

  Based on the information acquired from the previous steps, make a strategic decision regarding method offloading. Determine which methods are to be offloaded for remote execution and which ones are to remain for local execution.

The titles are intended to provide a quick and clear understanding of each step's purpose and activities.

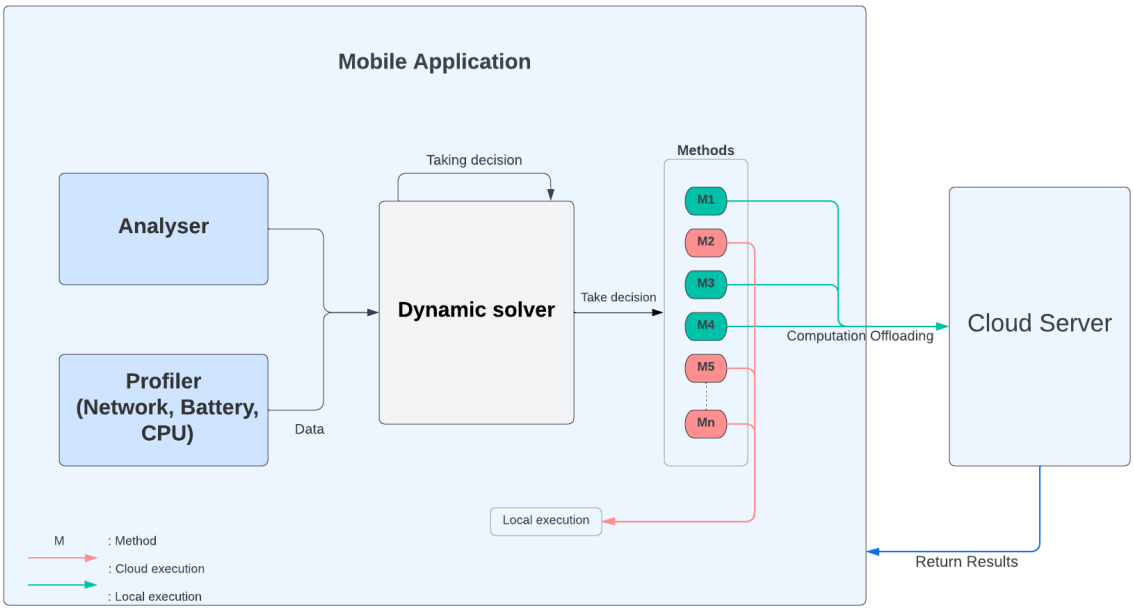


Figure 3.1: Schematic Representation of Application Decomposition and Offloading

## 3.3 Objectives of the proposed approach

As with the majority of studies conducted in the realm of computational offloading, our proposed approach is centered on optimizing energy consumption by reducing the runtime of the mobile application.

- **Minimizing Execution Time:** The primary objective of this approach is to reduce the time it takes for the mobile application to run. By decomposing the application into individual methods and offloading certain methods for remote execution, the overall processing load on the mobile device is lessened. This decrease in load can lead to faster execution times as the mobile device can process local tasks more swiftly and remote tasks can run concurrently. The decision to offload a method is influenced by several factors, including the method's dependency on other methods, execution time in different environments, and the available resources (like CPU, battery, and network).

- **Optimizing Energy Consumption:** Energy optimization is a primary objective of our approach, achieved through intelligent task offloading. By considering the energy consumption of both local execution and remote computation, we aim to strike a balance that results in net energy savings. The offloading decisions take into account the energy costs associated with data transfer and remote computation.

Our approach targets scenarios where the mobile device's resources (CPU, RAM, battery) may be insufficient for the application's computational demands. In such cases, computationally intensive tasks are dynamically offloaded to the cloud during runtime. This offloading strategy alleviates the strain on the mobile device's battery, contributing to energy conservation.

Through profiling and understanding the energy footprint of different methods, we optimize the offloading process to minimize energy consumption. By intelligently distributing tasks between the mobile device and the cloud, we achieve energy savings while ensuring efficient execution of the application.

In reference to the thematic taxonomy of algorithms in Figure 1.8, as presented in Chapter Two, my research contribution can be visualized and compared using the following table. This table will illustrate the unique attributes of my work, providing a distinct representation within the established taxonomy:

Table 3.1: Characteristics of the Proposed Approach in MCC Partitioning

| APA | Granularity | Model | Language | Profiler | Allocation Decision | Analysis Technique | Annotation |
|---|---|---|---|---|---|---|---|
| APA Proposed | Method | Hybrid | Single | Hardware, Network, Software | Online | Static | Automatic |

- **Granularity:** Partitioning by method is highly suitable for an object-oriented programming language like Java. This is due to the inherent structure of object-oriented languages, where code is organized into methods that represent actions an object can take. By partitioning at the method level, you can effectively manage and offload individual tasks, enabling more flexibility and potentially finer control over performance optimization.

- **Supported Language:** Java was chosen due to its wide adoption in mobile application development, particularly for Android. Java's object-oriented nature, robustness, and established libraries make it a logical choice for tackling complex tasks like application partitioning.

- **Partitioning Model:** The hybrid model combines a general representation through a method call graph with problem formulation in LP (Linear Programming). This approach provides a visual, structured understanding of method interactions, while LP offers a mathematical way to optimize the offloading decisions considering multiple factors and constraints.

- **Profile:** Profiling is crucial as it provides the necessary data about the hardware, software, and network state. It helps in making informed decisions about which methods to offload, based on the current state and capabilities of both the mobile device and the cloud server.

- **Allocation Decision:** Making allocation decisions online, in real time, allows for greater responsiveness to changes in the application context. This could include changes in device state, network conditions, or user behavior, allowing for more dynamic and efficient method offloading.

- **Analysis and Annotation Techniques:** Automating this process increases the efficiency and accuracy of identifying suitable methods for offloading. By analyzing method dependencies and selecting the most appropriate ones based on conditions and constraints, this approach ensures that only those methods that stand to benefit most from offloading are selected, maximizing the effectiveness of the offloading strategy.

All of these elements contribute to the overall goals of minimizing application execution time and optimizing energy consumption, making the strategy responsive, flexible, and effective.

## 3.4 Detailed description of the approach

We propose an architecture that aims to minimize execution time and, consequently, optimize energy consumption. The architecture is based on three modules that determine which methods should remain on the mobile device and which ones should be offloaded:

### 3.4.1 Profiling module

The module actively monitors hardware, software, and network components, employing a specific profiling technique to gather information for monitoring purposes. It ensures comprehensive tracking of system performance by focusing on the underlying hardware infrastructure, software processes, and network activity as follow:

**Hardware Profiling:**

The objective of this module is to gather comprehensive information about the physical hardware resources (mobile and cloud devices), such as RAM, processor, and battery capacity. It focuses on collecting data related to the hardware components to better understand their capabilities and optimize resource allocation.

**Software Profiling:**

This module is responsible for gathering information about the mobile application, such as code size, memory usage, and other software-related metrics. Its role is to analyze the characteristics and resource requirements of the application to inform decision-making processes, such as method allocation and offloading. By profiling the software aspects, it helps optimize resource utilization and enhance overall performance.

**Network Profiling:**

The network profiling module gathers data on the network environment, such as bandwidth, 3G, 4G, and 5G connectivity, as well as Wi-Fi connectivity. Its primary objective is to obtain information about network conditions and capabilities, enabling informed decision-making regarding method allocation and offloading. By analyzing network parameters.

### 3.4.2 Analyzer module

The"Analyzer" is designed to analyze source code of an application and provide valuable insights about its methods, including local execution time $LE_t$, remote execution time $RE_t$, and communication cost $CC$. By examining the code line by line, first we build a matrix that captures crucial information about each method. This matrix serves as a comprehensive representation of the application's functionality and performance characteristics, the algorithm 1 provided gives a general description about how it works:

**Algorithm 1:** Analyzer

**Input:** Source_code_of_the_application

**Output:** Matrix_of_methods, local_execution_time_for_each_method, remote_execution_time_for_each_method ,communication_cost

```
1  if the code is compiled then
2      Recompile the code
3  else
4      Read the code
5  for (each code line) do
6      Matrix ← Matrix + get_Method
7  for (Matrix) do
8      Calculate local execution time
9      Matrix ← Matrix + local_execution_time
10     Calculate remote execution time
11     Matrix ← Matrix + remote_execution_time
12     Calculate communication cost
13     Matrix ← Matrix + communication_cost
14 return Matrix
```

The formula for calculating execution time based on the information taken from the previous module can be reformulated as follows:

$$\text{Execution Time} = \frac{\text{Number of Instructions}}{\text{Instructions per Cycle}} \text{Cycle Time} \tag{3.1}$$

In this reformulated formula, we first divide the number of instructions by the instructions per cycle to determine the number of cycles needed to execute all the instructions. Then, we multiply this by the cycle time to obtain the total execution time.

$$\text{Transfer Time} = \frac{\text{Data Size}}{\text{Bandwidth}} \tag{3.2}$$

In this formula, we divide the data size by the available bandwidth to determine the time it would take to transfer the given amount of data.

After extracting all the necessary data using the previous algorithm, the next step is to construct a graph that we will use for the next module. The graph will capture the methods, their dependencies, and the associated execution and communication costs.

The algorithm begins by initializing an empty graph object. It then creates two special nodes: a source node (representing the mobile device) and a sink node (representing the cloud). These nodes serve as the starting and ending points of the graph.

Next, the algorithm iterates over each method, representing a method that can be offloaded to the cloud. For each method, a corresponding node is created and added to the graph. The algorithm sets the weights of two edges connected to the method node.

The first edge connects the source node to the method node, representing the local execution of the task on the mobile device. The weight of this edge is set to the local execution time of the method.

The second edge connects the method node to the sink node, representing the remote execution of the task in the cloud, including the communication cost. The weight of this edge is set to the sum of the remote execution time and the communication cost associated with the method.

Additionally, the algorithm considers the data dependencies between methods. For each method, it iterates over its dependencies and creates edges between the method node and the nodes representing its dependent methods. These edges capture the data transfer time between the tasks.

Once all the nodes and edges have been created and weighted according to the extracted data, the algorithm returns the constructed graph, along with the source and sink nodes.

The resulting graph can then be used for further analysis and energy optimization , such as applying a mincut algorithm to determine the optimal partitioning of methods between local and remote execution, minimizing the overall execution time while considering the communication costs in mobile cloud computing.

In this given algorithm 2, we will proceed with the previous steps to construct the graph:

**Algorithm 2:** Creating Graph

**Input:** methods, local_execution_time, remote_execution_time, communication_cost, data_dependencies

**Output:** Graph

```
Define the graph
graph ← new Graph()
Create new node: node_source
Create new node: node_sink
graph.addNode(node_source)
graph.addNode(node_sink)
for each method ∈ methods do
    method_node ← new Node(method)
    graph.addNode(method_node)
Set Edge Weights
add edge between node_source and method_node with weight =
 local_execution_time[method]
add edge between method_node and node_sink with weight =
 (remote_execution_time[method] + communication_cost[method])
for each method ∈ methods do
    for each dependency in data_dependencies[method] do
        dependent_method ← findNode(dependency)
        graph.addEdge(method_node, dependent_method, weight =
         data_transfer_time[dependency])
return Graph, source, sink
```

In summary, the provided figure3.4 illustrates a simulated representation of the final graph:

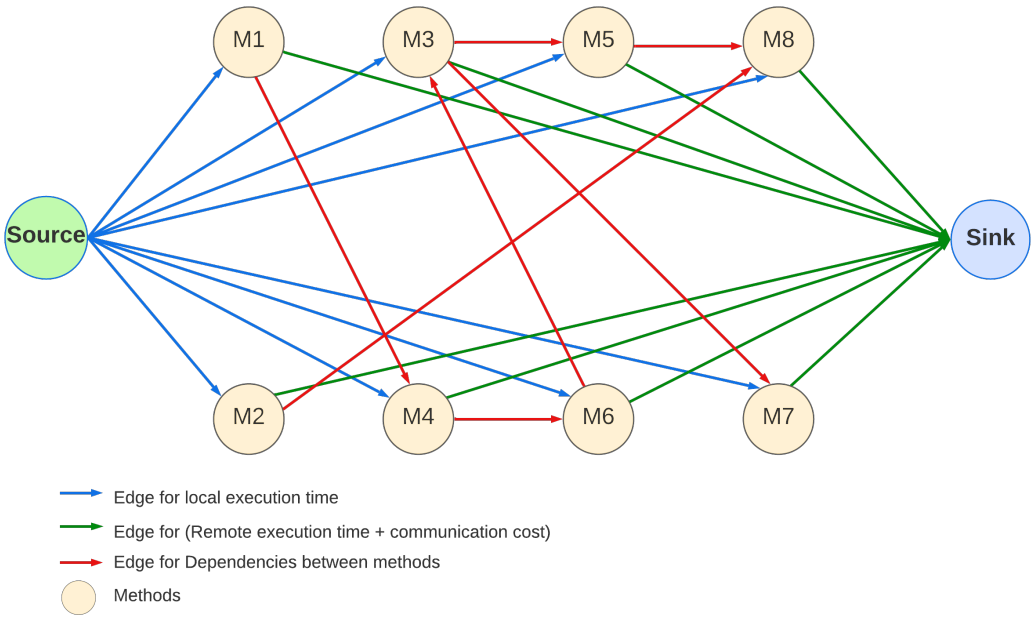


Figure 3.2: Weighted Graph Visualization

### 3.4.3 Solver

The main module ”Solver” gathers information from different modules: profiler and analyzer, and uses them as inputs for a global optimization of the solution.

In our approach we chose the MinCut Algorithm, also known as the Minimum Cut Algorithm, is a graph-based algorithm used to find the minimum cut in a graph. A cut in a graph divides the vertices into two disjoint sets, and the weight of the cut is the sum of the weights of the edges crossing the cut. The MinCut Algorithm aims to find the cut with the minimum weight, which represents the least costly way to separate the graph into two components.

The chosen model for the MinCut Algorithm is the Edmonds-Karp Algorithm[1] . The Edmonds-Karp Algorithm is a specific implementation of the Ford-Fulkerson[2] method for finding the maximum flow in a network. It efficiently computes the minimum cut

[1]The Edmonds-Karp algorithm is a realization of the Ford-Fulkerson method used to calculate the maximum flow in a flow network.[45]

[2]The Ford-Fulkerson algorithm is a solution for the max-flow min-cut problem, which deals with finding the maximum flow in a network consisting of vertices and weighted edges connecting them.[46]

by repeatedly finding augmenting paths in the residual graph until no more augmenting paths can be found.

By using the Edmonds-Karp Algorithm as the underlying model for the MinCut Algorithm, we can effectively determine the minimum cut that minimizes energy consumption. This algorithm guarantees an efficient and reliable partitioning solution that optimizes energy usage.

The Edmonds-Karp Algorithm has been chosen due to its favorable time complexity compared to other algorithms, as highlighted in the study conducted in the TPcoder website [47]. The study presented in the following table that compared the time complexities $O(n^2m)$ of different algorithms:

Table 3.2: Comparison of all the algorithms for maximum flow *[47]*

| Number of Vertices | Ford-Fulkerson-Karp | Edmond-Karp | Goldberg-Tarjan |
|---|---|---|---|
| 100 | 12.4008 | 3.10001 | 11.8008 |
| 550 | 1476.08 | 360.021 | 1378.78 |
| 1000 | 9118.72 | 2062.02 | 9568.05 |

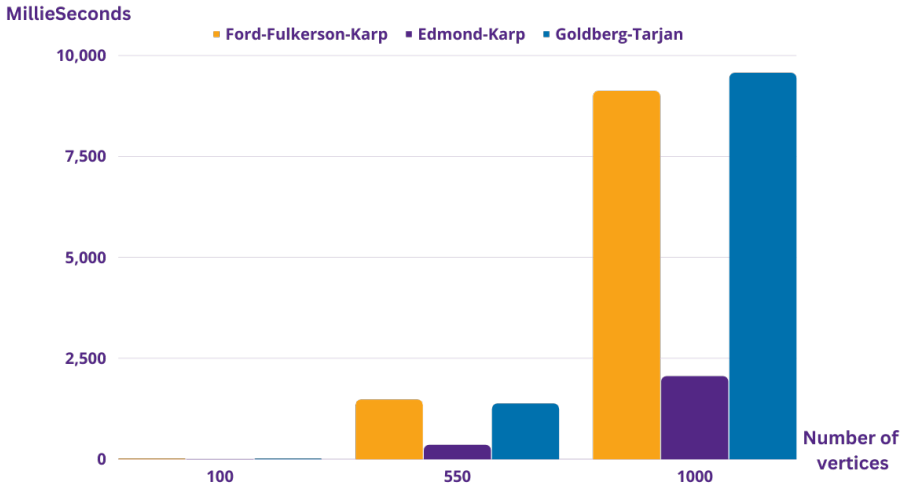


Figure 3.3: Graph of comparison of all the algorithms for maximum flow *[47]*

As mentioned earlier, the Edmonds-Karp Algorithm is primarily used for finding the maximum flow in a network. However, after the maximum flow is calculated, the algorithm

then iterates through all methods (graph edges from the source and to the sink). For each method, it calculates the flow on the device $F_d$ and the cloud $F_c$. The aim is to balance the total execution time on the device and the cloud, so the target flow $T_d$ on the device is set to half of the total flow on the cloud.

Next, the algorithm iterates through all the methods again. If adding the flow of a method to the current flow on the device $cF_d$ doesn't exceed the target flow, this method is selected for execution on the device $D$, and the flow of this method $f(s, m)$ is added to the current flow on the device $cF_d$. Otherwise, the method is offloaded to the cloud $C$.

The provided algorithm serves as a general framework for the Solver module:

**Algorithm 3:** Solver

**Input:** Graph

**Output:** Methods_to_offload, Methods_for_local_execution

$F \leftarrow$ Apply Edmonds_Karp_Algorithm(G, source, sink);
$F_d \leftarrow 0$
**for** *each method* $\in$ *methods* **do**
    $F_d \leftarrow F_d + f(s, m)$;
$F_c \leftarrow 0$
**for** *each method* $\in$ *methods* **do**
    $F_c \leftarrow F_c + f(m, t)$;
$T_d \leftarrow \frac{F_c}{2}$
$F_c \leftarrow 0$
$D \leftarrow []$
$C \leftarrow []$
$cF_d \leftarrow 0$
**for** *each method* $\in$ *methods* **do**
    **if** $cF_d + f(s, m) \leq T_d$ **then**
        $D \leftarrow D + m$
        $cF_d \leftarrow cF_d + f(s, m)$
    **else**
        $C \leftarrow C + m$;
**return** D, C;

## 3.5 Scenario of interaction between modules

Here is the sequence diagram that represents the interactions between the different modules of the proposed approach.

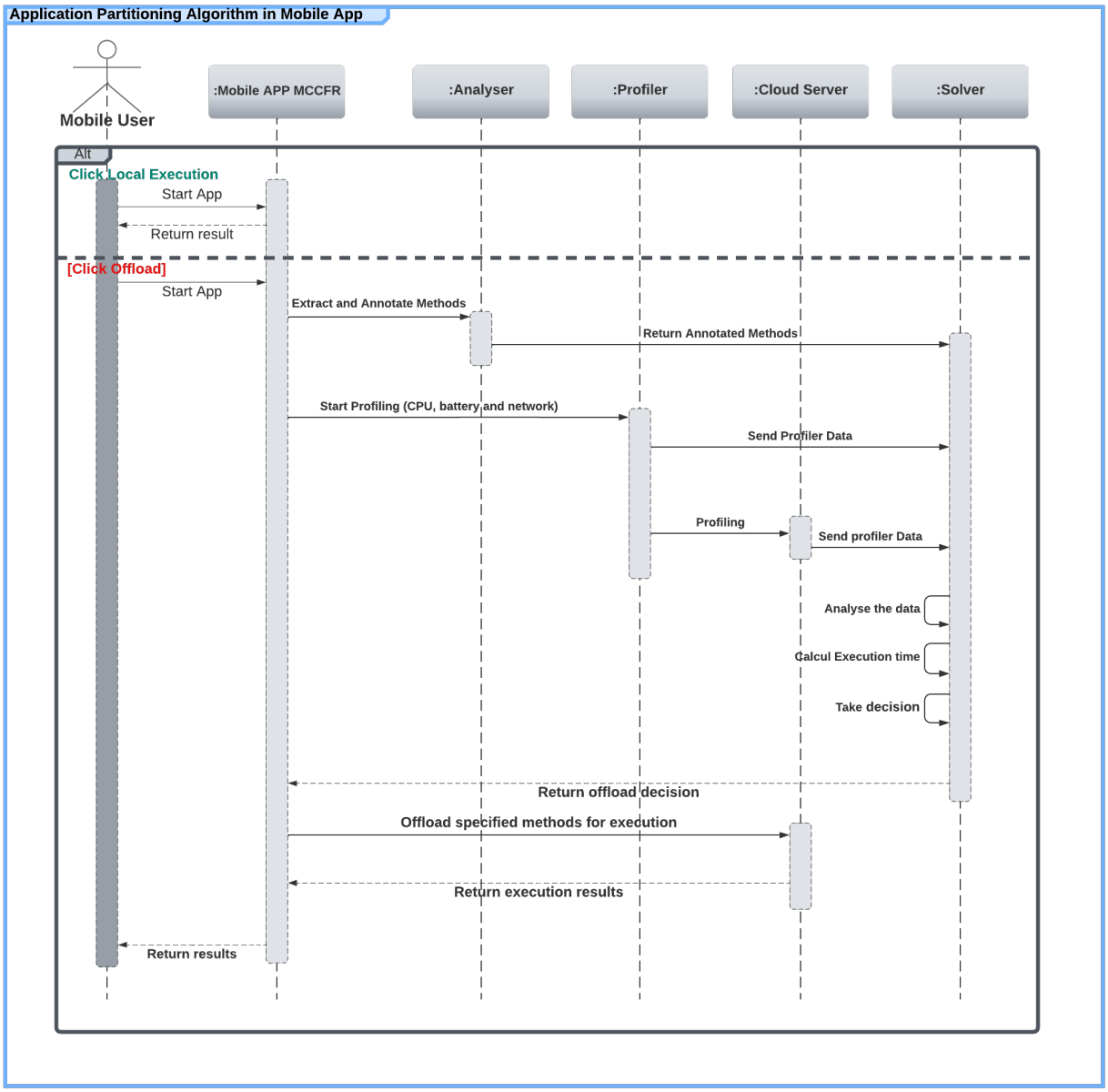


Figure 3.4: Sequence diagram of the proposed approach

## 3.6 Conclusion

In this chapter, we provided a general overview of our proposed approach, followed by a detailed architecture diagram illustrating the flow of actions and the various modules that constitute it. We further depicted their interactions through a comprehensive sequence diagram.

The next chapter will delve into the implementation and evaluation of the proposed approach, providing a thorough analysis and examination of its practical application. By exploring the implementation details, we aim to validate the effectiveness and efficiency of our approach in addressing the identified challenges.

# Chapter 4

# Implementation and Evaluation of the Proposed Approach

## 4.1 Introduction

In this chapter, we implemented our proposed approach to optimize energy consumption and execution time in mobile computing. Specifically, we focused on a facial recognition application developed for the Android platform.

We will discuss the chosen domain, the tools and languages used for development, and present the execution results obtained. This analysis will provide insights into the practical applicability and effectiveness of our method partitioning strategy.

The results obtained from the application's execution, including metrics such as energy consumption, execution time, and overall performance, will be presented and analyzed. These findings will allow us to evaluate the efficiency and effectiveness of our approach in achieving its objectives.

Overall, this chapter serves as a crucial step in assessing the real-world feasibility and performance of our proposed approach in the context of mobile computing.

## 4.2 The use case

In our project, our primary focus is on optimizing energy consumption in the context of mobile cloud computing through data offloading. To illustrate and evaluate our approach, we have chosen to develop a facial recognition Android application.

Facial recognition applications are widely used and offer potential for studying energy optimization in mobile cloud computing.

There are several reasons why we specifically chose a facial recognition application for our project, shown in this follow table 4.1

Table 4.1: Reasons for Facial Recognition

| Reasons for Selection |
|---|
| 1. Facial recognition involves complex algorithms and computational tasks, making it suitable for energy optimization in mobile cloud computing. |
| 2. Facial recognition applications are widely used and relevant in various industries. |
| 3. Evaluating energy optimization in a practical and tangible use case. |

Figure 4.1 refers to the application logo that is symbolizes three key aspects: the

phone positioned behind the cloud, representing the cloud's role in supporting mobile devices; and the face enclosed within the logo, representing our specific use case of facial recognition. This logo design illustrates the concept of the cloud taking care of mobile devices while highlighting our focus on facial recognition technology. By incorporating these elements, the logo visually communicates the central theme of the thesis in a concise and meaningful manner.


Figure 4.1: MCC Facial Recognition Android application logo

## 4.3 Technical Tools and Frameworks:

### 4.3.1 Development Environment

For configuring the development environment, I opted to utilize a personal laptop to set up the cloud infrastructure, this table4.2 summarizing the development environment for the client-side (Pixel 6 Pro) and server-side (HP laptop Elite-Book):

Table 4.2: Development environment tools

| Client-side | Server-side |
|---|---|
| **Device**<br>Pixel 6 Pro | **Laptop**<br>HP EliteBook |
| **CPU**<br>Google Tensor G1 | **CPU**<br>Ryzen 5 PRO 3500U |
| **RAM**<br>8GB | **RAM**<br>16GB 2666mhz |
| **Storage**<br>128GB | **Storage**<br>250GB SSD NVMe |

### 4.3.2 Programming Languages

#### Java

Java is a popular programming language known for its"write once, run anywhere" principle. It is object-oriented, platform-independent, and has a vast ecosystem of libraries and frameworks. It is widely used for mobile development. [48]

#### Why Java:

Java is the predominant language used for creating Android applications. Given the widespread usage of Android devices in the mobile market, opting for Java simplifies the integration of your mobile cloud computing solution with the Android platform. By utilizing established Android development practices, libraries, and APIs, you can develop resilient and scalable applications.


Figure 4.2: Java Logo
*[49]*

#### Python

Python is an interpreted, high-level programming language developed by Guido van Rossum and introduced in 1991. It is renowned for its emphasis on code readability and simplicity, making it widely favored by both novice and experienced programmers. Python's design philosophy revolves around providing a clear and concise syntax, enabling developers to write code that is easily comprehensible and maintainable. [50]

**Why Python:**

Python is highly regarded for its simplicity and rapid development capabilities, partly due to the extensive range of third-party libraries and frameworks available. Python boasts popular frameworks such as Django and Flask (the one we used), which come equipped with pre-built components and abstractions that expedite server-side development. These frameworks streamline tasks like managing HTTP requests, handling databases, and implementing authentication and security functionalities.


Figure 4.3: Python Logo
*[51]*

### 4.3.3 Frameworks and libraries

In this folowing table 4.3 frameworks, and libraries that were used to develop MCCFR. This comprises the technology stack for both the client and server sides:

Table 4.3: Frameworks and Libraries

| Library | Definition |
|---|---|
|   | Flask is a popular Python web framework that provides tools and functionalities for building web applications. It simplifies tasks such as routing, handling HTTP requests, and managing sessions. [52] |
|   | TensorFlow Lite is a lightweight version of the TensorFlow deep learning framework designed for mobile and embedded devices. It enables efficient execution of machine learning models on resource-constrained platforms. [53] |
|   | JGraphT is a Java library for graph data structures and algorithms. It provides a wide range of graph-related functionalities, including graph creation, manipulation, traversal, and path finding. [54] |

### 4.3.4 Development Tools

The tools we used to develop the system are briefly described here:

#### Android Studio

Android Studio is a specialized integrated development environment (IDE) specifically crafted for creating Android applications. It is constructed upon the foundation of IntelliJ IDEA and offers an extensive range of tools and functionalities that streamline the entire app development process. There are several compelling reasons to opt for Android Studio. Firstly, it serves as the official IDE for Android app development, ensuring compatibility and optimal support. Additionally, it equips developers with all the essential resources, documentation, and tools required for constructing Android applications. The versatility of Android Studio allows developers to utilize either Java or Kotlin to develop native Android apps. Moreover, Android Studio provides robust debugging and profiling capabilities to facilitate the identification and resolution of issues within your application. It encompasses features like breakpoints, step-by-step debugging, and advanced profiling tools that aid in analyzing performance bottlenecks, memory consumption, and CPU

usage.[55]


Figure 4.4: Android studio Logo
*[55]*

### PyCharm

PyCharm is an efficient integrated development environment (IDE) meticulously crafted for Python programming. It is a powerful software developed by JetBrains that facilitates the writing of Python code with greater ease. Specifically designed for Python programming, it encompasses a range of features that expedite coding and reduce the errors. PyCharm provides helpful tools like code suggestions, highlighting important syntax, formatting code neatly, and catching mistakes.

Moreover, PyCharm has a large and active community of Python developers who share knowledge and resources, making it a great choice for beginners and experienced programmers alike. With PyCharm, Python development becomes more efficient and enjoyable [56].


Figure 4.5: PyCharm Logo
*[57]*

## 4.4 The architecture of the project

For the implementation of our approach, we followed the architecture proposed in Chapter 3. We started by developing the first two modules, profiler and analyzer, in order to use their results in the implementation of the final module, solver.

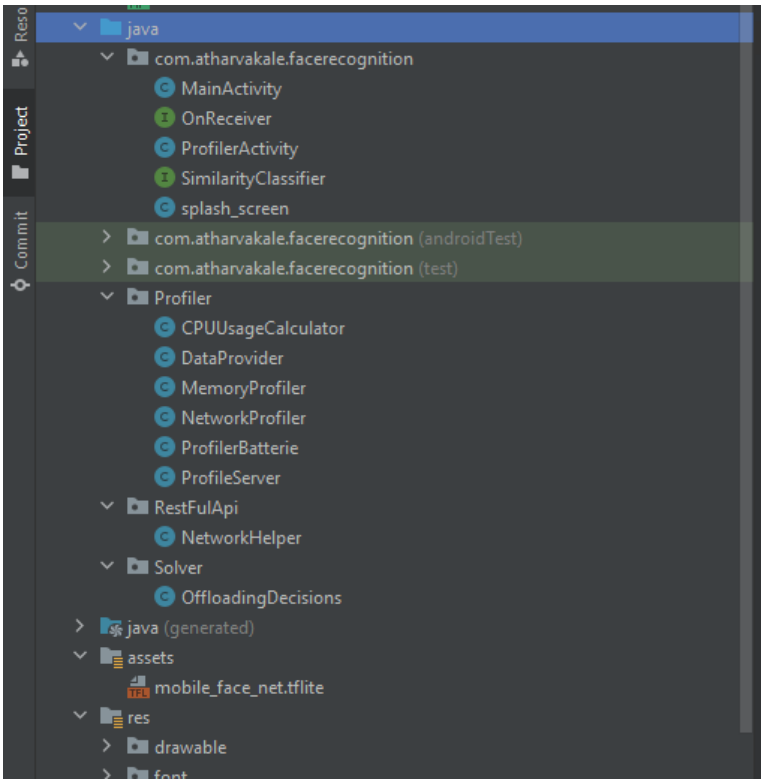


Figure 4.6: The architecture of the project in Android Studio

### 4.4.1 Profileur H/N/S

As presented in Chapter 3, the profiler consists of three sub-modules (in Android, these are services that execute their methods and store the results): the Hardware Profiler, Network Profiler, and Software Profiler. In this section, we present code snippets from each module along with the necessary explanations.

#### Hardware Profiler

Code to retrieve the current battery percentage (using a BroadcastReceiver[1]).

```
3 usages
public static class BatteryInfoReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        int level = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, defaultValue: 0);
        Log.d( tag: "Battery Level", msg: "Battery level: " + level);
    }
```

Figure 4.7: Code to get the battery level of the mobile device

[1]BroadcastReceiver : In order to receive intents, Android allows the creation of a class that implements BroadcastReceiver. These objects are designed to receive intents and perform specific behaviors. The BroadcastReceiver interface has only one method, onReceive().

```
2 usages
public float calculateCPUUsage() {
    long startTime = System.currentTimeMillis();
    long startCpuTime = Debug.threadCpuTimeNanos();

    try {
        Thread.sleep( millis: 1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    long endTime = System.currentTimeMillis();
    long endCpuTime = Debug.threadCpuTimeNanos();

    long elapsedTime = endTime - startTime;
    long cpuTime = endCpuTime - startCpuTime;

    float usage = (cpuTime / (float) (elapsedTime )) ;
    Log.d( tag: "CPU Usage",  msg: "Usage: " + usage + "%");

    return usage;
}
```

Figure 4.8: Code to retrieve the CPU capacity

```
1 usage
private final Context context;
2 usages
private final ActivityManager activityManager;
2 usages
private final TextView textView;
1 usage
public MemoryProfiler(Context context, TextView textView) {
    this.context = context;
    this.activityManager = (ActivityManager) context.getSystemService(Context.ACTIVITY_SERVICE);
    this.textView = textView;
}
1 usage
public void displayMemoryUsage() {
    long memoryUsage = getMemoryUsage();
    textView.setText( memoryUsage/1024 + " KB");
}
1 usage
private long getMemoryUsage() {
    ActivityManager.MemoryInfo memoryInfo = new ActivityManager.MemoryInfo();
    activityManager.getMemoryInfo(memoryInfo);
    return memoryInfo.totalMem - memoryInfo.availMem;
}
```

Figure 4.9: Code to retrieve the available RAM

The following code runs on the server side of a mobile application. It retrieves CPU capacity information from the cloud using HTTP methods through the OkHttp[2] library. The specific method used is ”GET” to fetch the CPU capacity from the cloud. We have two main purposes for obtaining this information.

Firstly, we want to display the CPU usage result on the mobile app’s User Interface. To achieve this, we have a button illustrated in Figure 4.21 of the User Interface section. When the user clicks this button, a toast message will be shown on the mobile app,

[2]OkHttp, developed by Square, is a Java and Android HTTP client that aims to enhance resource loading speed and reduce bandwidth consumption. It enjoys extensive usage in open-source projects and serves as a fundamental component in libraries like Retrofit, Picasso, and numerous others.[58]

presenting the CPU usage of the cloud.

Additionally, the CPU capacity information is utilized to calculate the execution time on the cloud. This calculation is important for measuring performance and optimizing the application's functionality in the cloud environment.

```
1 usage
public static void getCpuUsage(Context context, CpuUsageCallback callback) {
    OkHttpClient client = new OkHttpClient();
    Request request = new Request.Builder()
            .url("http://192.168.1.102:5000/cpu_usage")
            .build();

    client.newCall(request).enqueue(new Callback() {
        @Override
        public void onResponse(Call call, Response response) throws IOException {
            String result = response.body().string();

            // Parse the JSON response and extract the CPU usage value
            try {
                JSONObject json = new JSONObject(result);
                double cpuUsage = json.getDouble( name: "cpu_usage");

                // Pass the CPU usage value to the callback
                callback.onCpuUsageReceived(cpuUsage);
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
```

Figure 4.10: Code to retrieve the CPU capacity from the Cloud

### Network Profiler

Below, I will present the collection of code snippets that are relevant to this module:

```
2 usages
public static boolean isNetworkAvailable(Context context) {
    ConnectivityManager connectivityManager = (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
    if (connectivityManager == null) {
        return false;
    } else {
        NetworkInfo[] networkInfoArray = connectivityManager.getAllNetworkInfo();
        if (networkInfoArray != null) {
            for (NetworkInfo networkInfo : networkInfoArray) {
                if (networkInfo.getState() == NetworkInfo.State.CONNECTED) {
                    return true;
```

Figure 4.11: Code to check the connectivity

```
public static String getNetworkType(Context context) {
    final ConnectivityManager connMgr = (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
    final NetworkInfo wifi = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
    final NetworkInfo mobile = connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);

    if (wifi.isAvailable() && wifi.getDetailedState() == NetworkInfo.DetailedState.CONNECTED) {
        return "WIFI";
    } else if (mobile.isAvailable() && mobile.getDetailedState() == NetworkInfo.DetailedState.CONNECTED) {
        return "4G";
    }

    return null;
}
```

Figure 4.12: Get the network type "4G or WIFI"

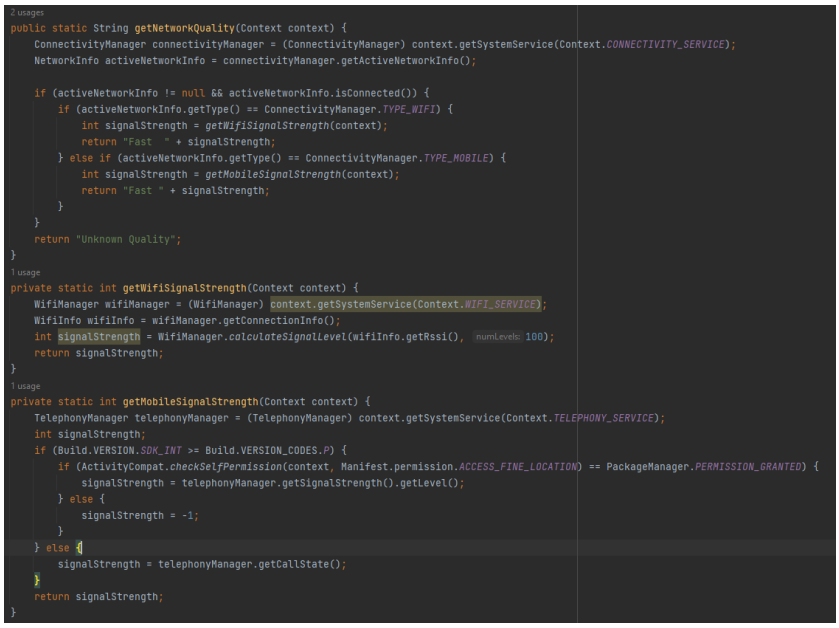


Figure 4.13: Code to Get the SignalStrengh

### Software Profiler

Code to determine the existing dependencies between methods: (This code needs to be dynamically injected into all methods using invoke by adding this"logCallerInfo()")


Figure 4.14: Code to determine the existing dependencies between methods

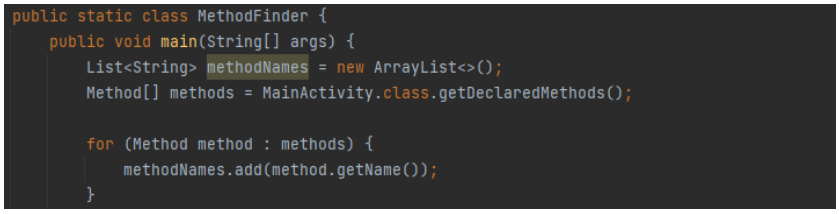


Figure 4.15: Code to get all the class methods

## 4.4.2 Analyzer

In this module, we constructed the graph using the JGraphT library in Java within the Android Studio environment, as illustrated in Chapter 3, I set it in the OffloadingDecisions class as it shown in the code provided in this figure 4.16:

```
1 usage
public OffloadingDecisions() {
    DataProvider dataProvider = new DataProvider();
    this.methods = dataProvider.getMethods();
    this.execution_time_on_device = dataProvider.getExecutionTimeOnDevice();
    this.execution_time_on_cloud = dataProvider.getExecutionTimeOnCloud();
    this.data_transfer_time = dataProvider.getDataTransferTime();
    this.data_dependencies = dataProvider.getDataDependencies();
}

1 usage
public Pair<Set<String>, Set<String>> makeOffloadingDecisions() {
    SimpleDirectedWeightedGraph<String, DefaultWeightedEdge> graph = new SimpleDirectedWeightedGraph<>(DefaultWeightedEdge.class);

    String source = "source";
    String sink = "sink";

    graph.addVertex(source);
    graph.addVertex(sink);

    for (int i = 0; i < methods.length; i++) {
        String method = methods[i];
        graph.addVertex(method);
        graph.setEdgeWeight(graph.addEdge(source, method), weight: 1.0 / execution_time_on_device[i]);
        graph.setEdgeWeight(graph.addEdge(method, sink), weight: 1.0 / (execution_time_on_cloud[i] + data_transfer_time[i]));

        for (String dependency : data_dependencies[i]) {
            graph.addVertex(dependency);
            graph.setEdgeWeight(graph.addEdge(method, dependency), weight: 1.0 / data_transfer_time[i]);
        }
    }
```

Figure 4.16: Code to construct the weighted graph

### 4.4.3 Solver

In the solver module, we have chosen to use the JGraphT library, which provides the Edmonds-Karp algorithm for calculating the maximum flow. The specific implementation of the Edmonds-Karp algorithm in JGraphT is represented by the EdmondsKarpMFImpl [3] class and then we finish the process of the Algorithm 3 of the third chapter .

```
network_security_config.xml | mobile_face_net.tflite | OffloadingDecisions.java | ProfilerBatterie.java | ProfileServer.java | MemoryProfiler.java | Netw

        EdmondsKarpMFImpl<String, DefaultWeightedEdge> ek = new EdmondsKarpMFImpl<>(graph);
        ek.calculateMaximumFlow(source, sink);

        Set<String> execute_on_device = new HashSet<>();
        Set<String> offload_to_cloud = new HashSet<>();

        double[] flowOnDevice = new double[methods.length];
        double[] flowOnCloud = new double[methods.length];
        double totalFlowOnDevice = 0;
        double totalFlowOnCloud = 0;

        Map<DefaultWeightedEdge, Double> flowMap = ek.getFlowMap();

        for (DefaultWeightedEdge edge : graph.outgoingEdgesOf(source)) {
            String method = graph.getEdgeTarget(edge);
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
                flowOnDevice[getIndex(method)] = flowMap.getOrDefault(edge, defaultValue: 0.0);
            }
            totalFlowOnDevice += flowOnDevice[getIndex(method)];
        }
```

Figure 4.17: Code to make the offloading decision

[3] EdmondsKarpMFImpl is a JGraphT class that implements the Edmonds-Karp algorithm to solve the maximum flow problem in a flow network. The network is represented by a weighted directed or undirected graph $G(V, E)$. The goal is to find the maximum feasible flow from a source vertex $s$ to a sink vertex $t$ while respecting capacity and flow conservation constraints. The algorithm efficiently computes the maximum flow in the given network. [59]

```
for (DefaultWeightedEdge edge : graph.incomingEdgesOf(sink)) {
    String method = graph.getEdgeSource(edge);
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
        flowOnCloud[getIndex(method)] = flowMap.getOrDefault(edge,  defaultValue: 0.0);
    }
    totalFlowOnCloud += flowOnCloud[getIndex(method)];
}

double targetFlowOnDevice = totalFlowOnCloud / 2;
double currentFlowOnDevice = 0;

for (int i = 0; i < methods.length; i++) {
    if (currentFlowOnDevice + flowOnDevice[i] <= targetFlowOnDevice) {
        execute_on_device.add(methods[i]);
        currentFlowOnDevice += flowOnDevice[i];
    } else {
        offload_to_cloud.add(methods[i]);
    }
}

return new Pair<>(execute_on_device, offload_to_cloud);
```

Figure 4.18: The rest of Code of Solver module

## 4.5 User Interface

Figure 4.19 illustrates the MCCFR interface, showcasing the face recognition process and providing a simplified overview of the application's functionality.

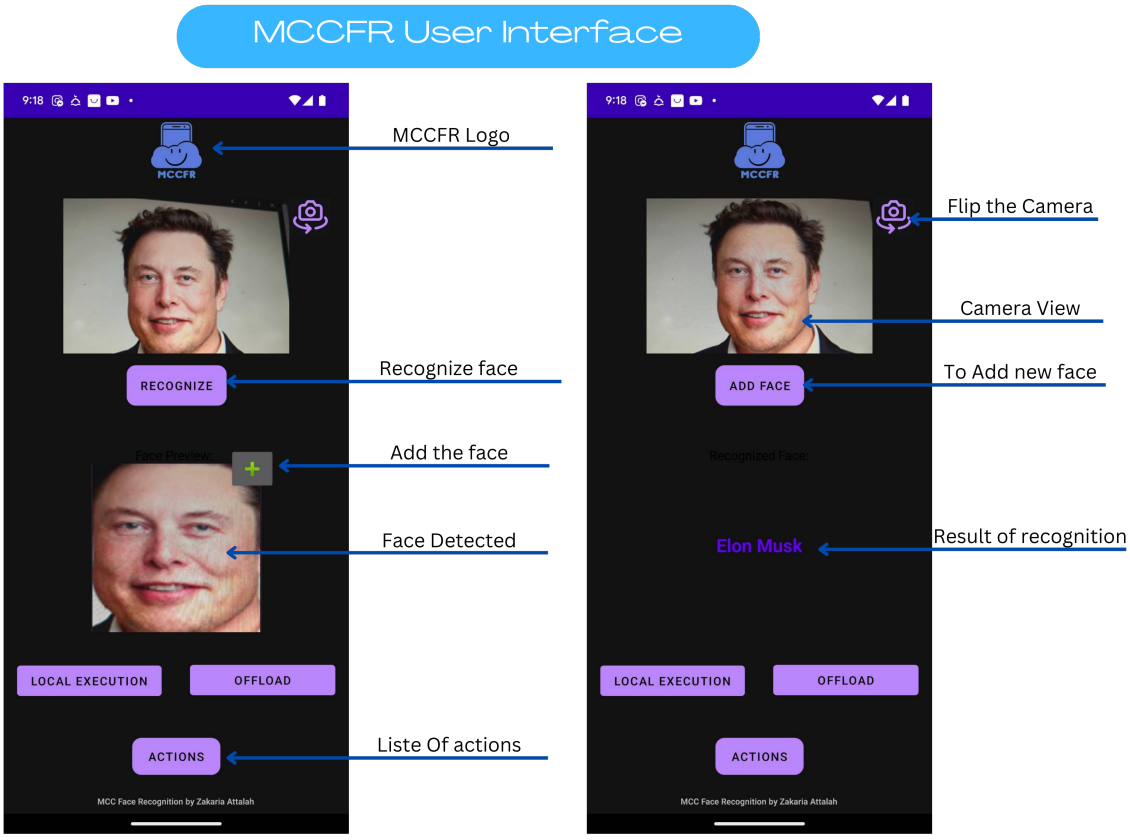


Figure 4.19: MCCFR User Interface

Figure 4.20 shows two scenarios for offloading decisions: the user choosing between Offloading Computation Process mode or remaining in Local Execution mode. Initially, the application operates in Local Execution mode. However, if the user selects Offload-

ing, the availability of the server is checked. If the server is accessible, the application initiates the offloading process and button of OFFLOAD becomes green. If the server is unavailable, an alert dialogue prompts the user to establish a connection, the OFFLOAD button becomes red and the application continues with local execution until the server becomes available.

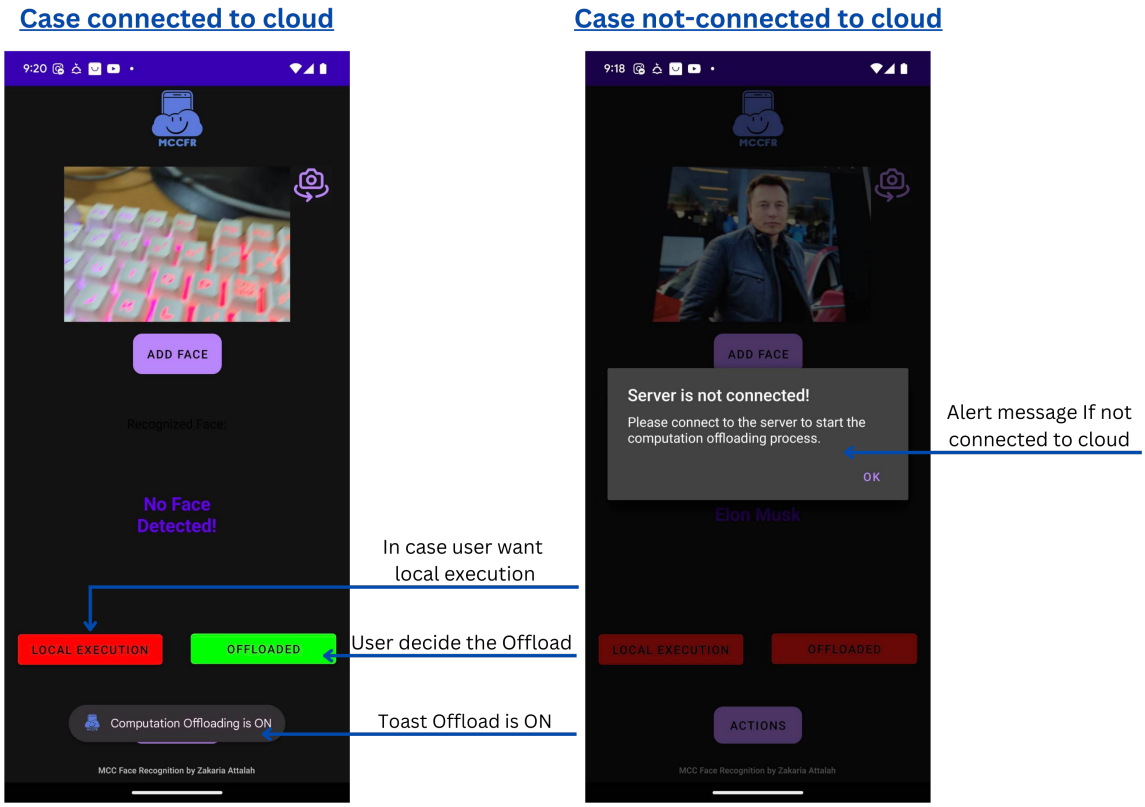


Figure 4.20: Cases of Offoloading decision by the user

Figure 4.21 displays two key components. Firstly, it presents a list of actions primarily associated with the facial recognition process. One notable action is the connectivity check with the server, without clicking the 'OFFLOAD' button. If the server connection is established, the user receives a 'Request received' toast message. In case of a failed connection, a 'Request failed' message is displayed. Additionally, the figure showcases the 'Get Profiler Information' action, which leads to another layout. This layout exhibits profiling data obtained from the initial Profiler modal. Furthermore, the figure includes a 'Cloud CPU Usage' button, which, when clicked, shows the CPU usage of the cloud through a toast message.

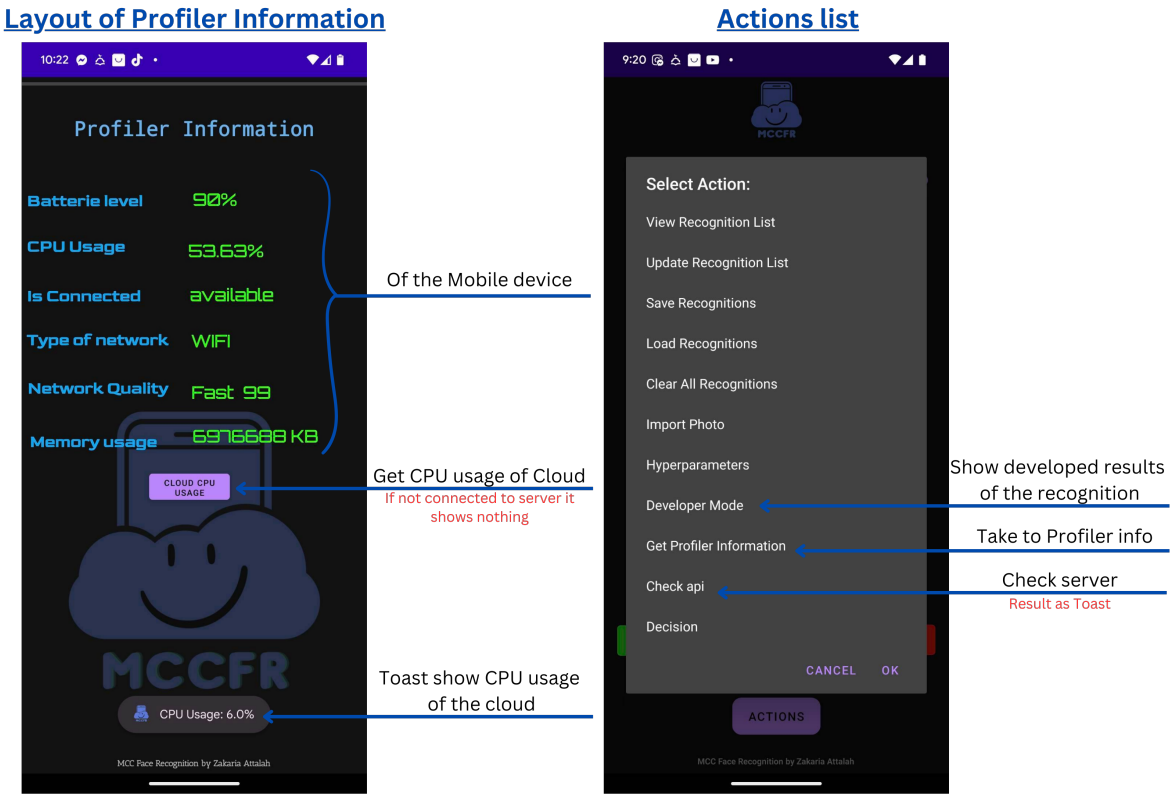


Figure 4.21: MCCFR Actions list and Profiler layout

## 4.6 Results

The graph is obtained by implementing the code for extraction within the analyzer module, and its visual representation can be observed in this figure 4.22:

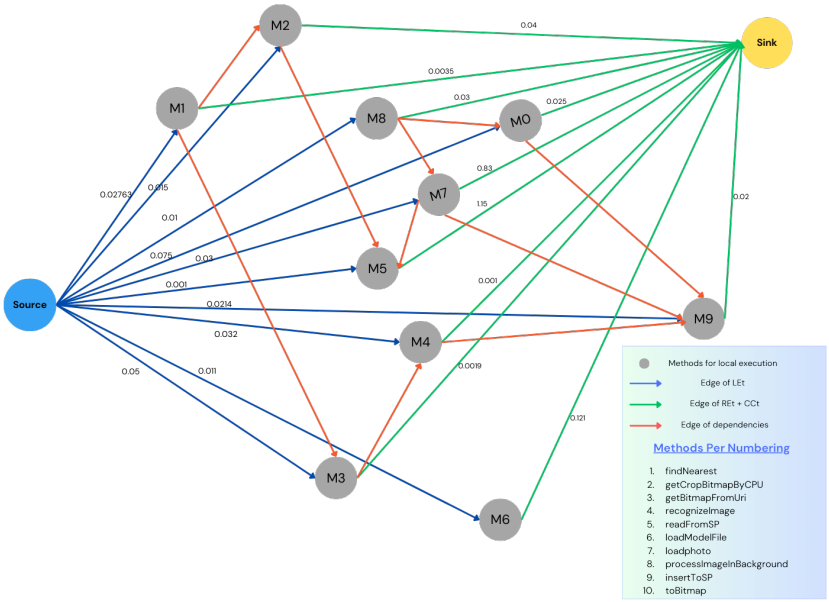


Figure 4.22: Extracted Graph" Visual Representation of Analyzed Data "

After implementing the Solver module in the code of Figure 4.18, we derived our final results.

To validate these results, we compared the total local execution time without offloading against the results with offloading. Figure 4.23 presents the code used to obtain these results, which were logged in Android Studio's Logcat. The satisfactory results demonstrated that the Solver module achieved the optimal execution time.

```
1 usage
public double calculateExecutionTime() {
    Set<String> methodsOnDevice = makeOffloadingDecisions().first;
    Set<String> methodsOnCloud = makeOffloadingDecisions().second;

    double totalExecutionTime = 0.0;

    // Calculate execution time on device
    for (String method : methodsOnDevice) {
        int index = getIndex(method);
        totalExecutionTime += execution_time_on_device[index];
    }

    // Calculate execution time on cloud
    for (String method : methodsOnCloud) {
        int index = getIndex(method);
        totalExecutionTime += execution_time_on_cloud[index] + data_transfer_time[index];
    }

    return totalExecutionTime ;
}

1 usage
public double calculateExecutionTimeWithoutOffloading() {
    double totalExecutionTime = 0.0;

    for (int i = 0; i < methods.length; i++) {
        totalExecutionTime += execution_time_on_device[i];
    }

    return totalExecutionTime ;
}
```

Figure 4.23: Code of getting total execution time in mobile and with offloading

```
new Thread(new Runnable() {
    @Override
    public void run() {
        OffloadingDecisions decisions = new OffloadingDecisions();
        Pair<Set<String>, Set<String>> offloadingDecisions = decisions.makeOffloadingDecisions();
        double totalExecutionTime = decisions.calculateExecutionTime();
        double totalExecutionTimeWithoutOffloading = decisions.calculateExecutionTimeWithoutOffloading();
        Log.d( tag: "OffloadingDecisions",  msg: "Methods to execute on device: " + offloadingDecisions.first);
        Log.d( tag: "OffloadingDecisions",  msg: "Methods to offload to cloud: " + offloadingDecisions.second);
        Log.d( tag: "OffloadingDecisions",  msg: "Total execution time by Computation Offloading: " + totalExecutionTime);
        Log.d( tag: "OffloadingDecisions",  msg: "Total execution time without offloading: " + totalExecutionTimeWithoutOffloading);
    }
}).start();
```

Figure 4.24: Code for showing the results in the Logcat

```
D  Methods to execute on device: [findNearest, getBitmapFromUri, recognizeImage, insertToSP]
D  Methods to offload to cloud: [loadphoto, toBitmap, loadModelFile, processImageInBackground, getCropBitmapByCPU, readFromSP]
D  Total execution time by Computation Offloading: 0.16385085892151072
D  Total execution time without offloading: 0.21038766133999345
```

Figure 4.25: Results in the Logcat of Android Studio

The following graph represents the outcome achieved by applying the Edmonds-Karp algorithm, which is a practical application of the Ford-Fulkerson method. It demonstrates the final result of implementing this algorithm with potential improvements.

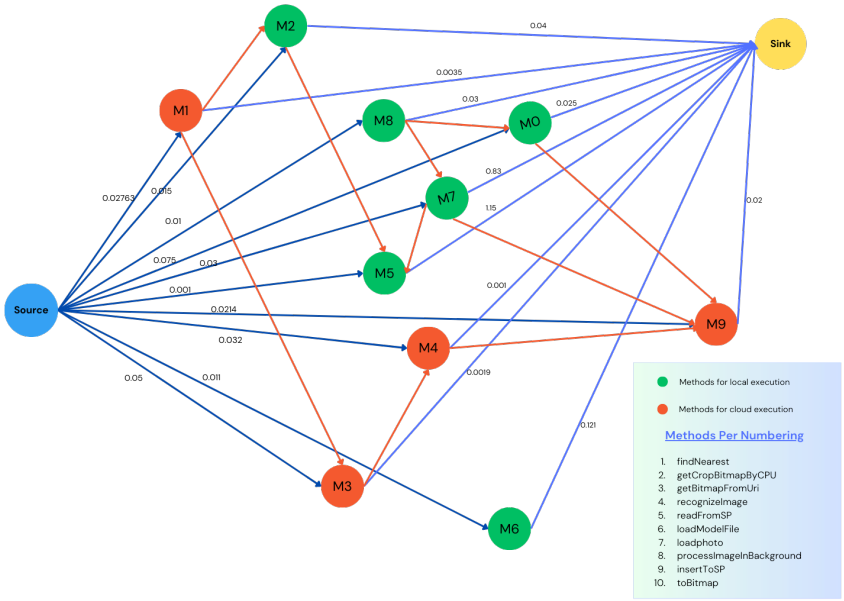


Figure 4.26: Final graph of the application of Edmonds-Karp algorithm

In the following chart, I aimed to visually represent the results that were displayed in the Logcat of Android Studio (see Figure 4.26) by three sequences, allowing for easier analysis and understanding the performance of our algorithm.

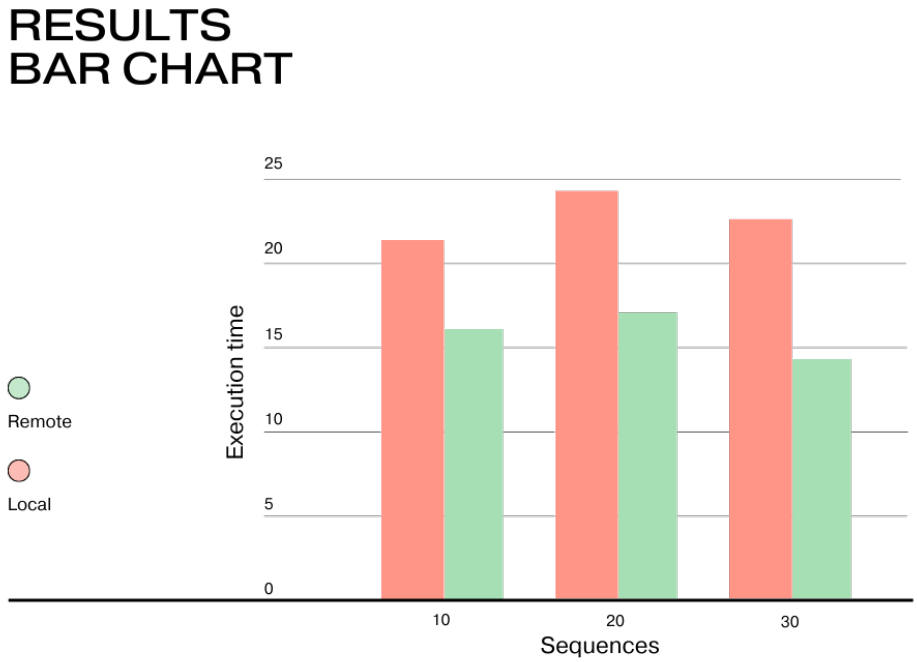


Figure 4.27: Bar chart display the results

Following the implementation of our approach and the utilization of two profilers, our battery profiler and the Android Studio energy profiler, we proceeded to evaluate the energy consumption of the application. Initially, during the short-term usage, we did not observe any noticeable changes in energy consumption. However, over a longer period of time, we noticed a substantial improvement. This improvement is visually represented a bar chart (see Figure 4.29).

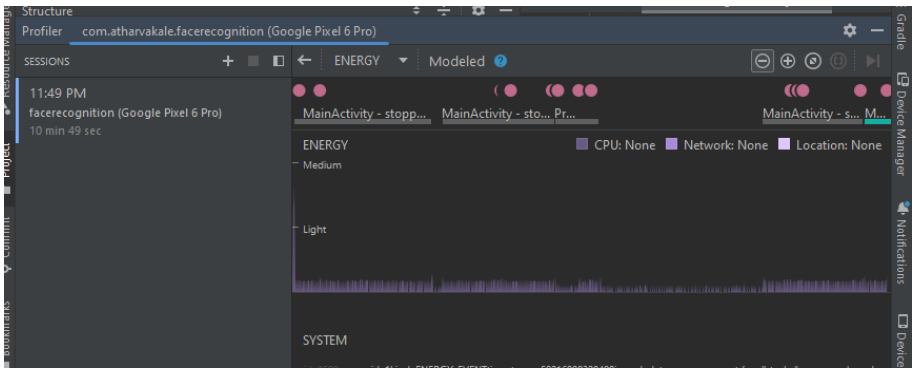


Figure 4.28: Android studio battery profiler

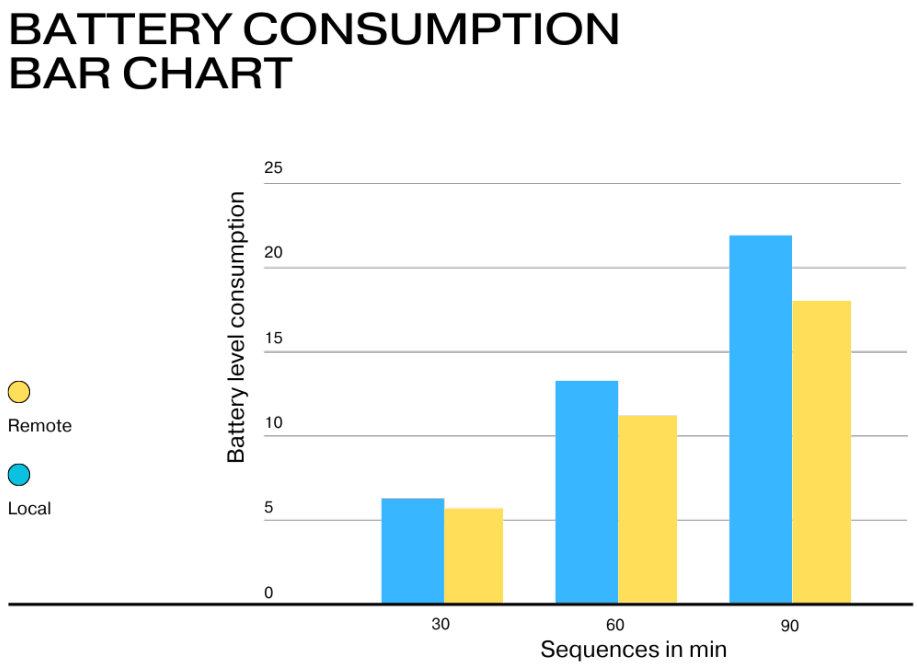


Figure 4.29: Battery consumption Bar Chart

The results clearly indicate a reduction in energy consumption, also indicating the effectiveness of our approach. It is important to note that these results may vary in cases of a very powerful cloud services. In such scenarios, even a short-term usage of the application could yield noticeable changes in energy consumption.

In the end, our analysis validates the effectiveness of our module in reducing energy consumption and optimizing the application's performance. These positive results encourage further exploration and implementation of energy-saving techniques in future projects.

## 4.7 Conclusion

In this final chapter of my manuscript, I dedicated my efforts to the implementation of the proposed approach discussed in the third chapter. For this purpose, I developed a mobile application called MCCFR, which is a facial recognition application. I extensively described the various tools and technologies employed to realize the approach. Moreover, I explained the different modules in detail, supported by relevant screenshots of both the implementation code and the application interface.

In the last stage, I tested the approach and thoroughly discussed the outcomes obtained. Through comprehensive evaluation and analysis, I observed significant advancements and improvements in the performance and efficiency of the MCCFR application. The results obtained from the tests reinforce the effectiveness of the proposed approach, showcasing its ability to achieve accurate facial recognition while optimizing energy consumption.

# Conclusion and Perspectives

# Synthesis

The increasing demand for mobile computing has transformed how tasks are performed, with constant connectivity to the Internet playing a crucial role. Computation offloading to the Cloud has emerged as a solution for tackling performance challenges on mobile devices.

Mobile Cloud Computing leverages computation offloading to reduce task execution time and enhance battery life. The core principle involves transferring resource-intensive computations from mobile devices to cloud servers.

This dissertation presents a comprehensive project encompassing a literature review, exploring relevant research areas, and investigating the concept of Mobile Cloud Computing, with a specific emphasis on energy consumption.

To address application partitioning in mobile environments, we propose a novel algorithm that optimally partitions applications under different cost models. Our approach aims to strike the best balance between time and energy efficiency, achieved through the construction of weighted call graphs for the applications.

The experimental results demonstrate that the proposed algorithm can achieve optimal partitioning results in terms of time and energy efficiency, depending on the environmental factors such as bandwidth and server speed. Computation offloading greatly benefits from high bandwidth and significant acceleration factors, while low bandwidth favors the non-offloading scheme.

In conclusion, this final project has provided me with the opportunity to delve into various aspects of computing, including Cloud Computing, Mobile Computing, Mobile Cloud Computing, computation offloading, partitioning algorithms, and programming. This experience has been immensely valuable, enabling me to acquire in-depth programming skills and enhance my abilities in analysis and synthesis. I am confident that these acquired concepts and skills will significantly contribute to my professional growth and success.

# Perspectives

Given the nature of the field of computer science, achieving a perfect and limitation-free solution is impossible, and the same holds true for our approach and implementation.

Therefore, there are several potential avenues for future work to further improve our research:

- Further improvements can be made to enhance the security aspects of computation offloading, ensuring the protection of sensitive data during the offloading process.

- Exploring energy-efficient strategies for workload allocation and task scheduling in the Mobile Cloud Computing environment.

- Add a historical module that makes the solver avoid taking a new decision for same situation.

- Make parallel execution of some methods on the clouds servers.

By addressing these areas, future research can contribute to advancing the field of Mobile Cloud Computing and optimizing energy consumption, resource utilization, and overall system performance.

# Bibliography


[1] Shahryar Shafique Qureshi, Toufeeq Ahmad, Khalid Rafique, et al. "Mobile cloud computing as future for mobile applications-Implementation methods and challenging issues." In: *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*. IEEE. 2011, pp. 467–471.

[2] National Institute of Standards and Technology. "Mobile Cloud Computing." In: (n.d.). Available at: https://www.nist.gov/programs-projects/mobile-cloud-computing.

[3] Yaser Jararweh. "Mobile Cloud Computing." In: *Mobile Cloud Computing: Architectures, Algorithms and Applications*. Springer International Publishing, 2015, pp. 3–10.

[4] Saeid Abolfazli et al. "Rich Mobile Applications: Genesis, taxonomy, and open issues." In: *Journal of Network and Computer Applications* 40 (2014), pp. 345–362. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2013.09.009. URL: https://www.sciencedirect.com/science/article/pii/S1084804513001975.

[5] Waseem Akhtar and Athanasios V. Vasilakos. "Mobile Cloud Computing: A survey, state of art, and future directions." In: *Mobile Networks and Applications* 23.2 (2018), pp. 207–230.

[6] Verónica Fernández, Félix Gómez Mármol, and Gregorio Martínez Pérez. "Mobile cloud computing: A survey." In: *Future Generation Computer Systems* 29.1 (2013), pp. 84–106.

[7] Sergio Barbarossa, Fragkiskos Sardis, and Iordanis Koutsopoulos. "Cloud and Mobile Cloud Computing: Kismet or New Avenue?" In: *2014 IEEE International Conference on Communications (ICC)*. IEEE. 2014, pp. 2576–2582.

[8] Asoke K. Talukder and Roopa Yavagal. *Mobile Computing: Technology, Applications, and Service Creation*. McGraw-Hill Education, 2010.

[9] Indu Sahu and US Pandey. "Mobile cloud computing: Issues and challenges." In: *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. IEEE. 2018, pp. 247–250.

[10] Fang Liu, Jin Tong, et al. "NIST Cloud Computing Reference Architecture." en. In: ().

[11] Peter Mell and Timothy Grance. "The NIST Definition of Cloud Computing." en. In: ().

[12] Debashis De. *Mobile Cloud Computing: Architectures, Algorithms and Applications*. en. 0th ed. Chapman and Hall/CRC, Jan. 2016. ISBN: 978-0-429-16240-4. DOI: 10.1201/b19208. URL: https://www.taylorfrancis.com/books/9781482242843 (visited on 04/11/2023).

[13] Jonathan Ng Zhen Yuan and Yu Hui Wen Wahidah Husain Wong Wan Ping. "HEALTHCARE APPLICATIONS ON MOBILE CLOUD COMPUTING." In: 2013.

[14] Hao Qian. "EXTENDING THE BATTERY LIFE OF MOBILE DEVICE BY COMPUTATION OFFLOADING." en. In: (2008).

[15] Ahmed Alzahrani, Nasser Alalwan, and Mohamed Sarrab. "Mobile cloud computing: Advantage, disadvantage and open challenge." In: *ACM International Conference Proceeding Series*. 2014. DOI: 10.1145/2590651.2590670.

[16] George H. Forman and John Zahorjan. "The challenges of mobile computing." In: *Computer* 27.4 (1994), pp. 38–47.

[17] Charles E Perkins. "Handling multimedia data for mobile computers." In: *Proceedings of 20th International Computer Software and Applications Conference: COMPSAC'96*. IEEE. 1996, pp. 147–148.

[18] ME Anagnostou, Arto Juhola, and ED Sykas. "Context Aware services as a step to pervasive computing." In: *Lobster workshop on location based services for accelerating the European-wide deployment of services for the mobile user and worker*. 2002, pp. 4–5.

[19] Karthik Kumar et al. "A survey of computation offloading for mobile systems." In: *Mobile networks and Applications* 18 (2013), pp. 129–140.

[20] Zhiyuan Li, Cheng Wang, and Rong Xu. "Computation offloading to save energy on handheld devices: a partition scheme." In: *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*. 2001, pp. 238–246.

[21] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li. "Adaptive computation offloading for energy conservation on battery-powered systems." In: *2007 International conference on parallel and distributed systems*. IEEE. 2007, pp. 1–8.

[22] Khalid Elgazzar, Patrick Martin, and Hossam S Hassanein. "Empowering mobile service provisioning through cloud assistance." In: *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. IEEE. 2013, pp. 9–16.

[23] Asrar Ahmed Baktayan and Ibrahim Ahmed Al-Baltah. "A survey on intelligent computation offloading and pricing strategy in UAV-Enabled MEC network: Challenges and research directions." In: *arXiv preprint arXiv:2208.10072* (2022).

[24] Minhaj Ahmad Khan. "A survey of computation offloading strategies for performance improvement of applications running on mobile devices." en. In: *Journal of Network and Computer Applications* 56 (Oct. 2015), pp. 28–40. ISSN: 10848045. DOI: 10.1016/j.jnca.2015.05.018. URL: https://linkinghub.elsevier.com/retrieve/pii/S1084804515001320 (visited on 04/12/2023).

[25] Shumao Ou, Kun Yang, and Jie Zhang. "An effective offloading middleware for pervasive services on mobile devices." In: *Pervasive and Mobile Computing* 3.4 (2007), pp. 362–385.

[26] Lei Yang, Jiannong Cao, Yin Yuan, et al. "A framework for partitioning and execution of data stream applications in mobile cloud computing." In: *ACM SIGMETRICS Performance Evaluation Review* 40.4 (2013), pp. 23–32.

[27] Radu-Corneliu Marin. "Hybrid contextual cloud in ubiquitous platforms comprising of smartphones." In: *International Journal of Intelligent Systems Technologies and Applications* 12.1 (2013), pp. 4–17.

[28] Mhammed Chraibi et al. "Policy based context aware service level agreement (SLA) management in the cloud." In: *CLOUD COMPUTING 2017* (2017), p. 132.

[29] Huaming Wu et al. "An optimal offloading partitioning algorithm in mobile cloud computing." In: *Quantitative Evaluation of Systems: 13th International Conference, QEST 2016, Quebec City, QC, Canada, August 23-25, 2016, Proceedings 13*. Springer. 2016, pp. 311–328.

[30] Jieyao Liu, Ejaz Ahmed, et al. "Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions." In: *Journal of Network and Computer Applications* 48 (2015), pp. 99–117.

[31] Tahmid Nabi et al. "Assessing the benefits of computational offloading in mobile-cloud applications." In: *Proceedings of the 3rd International Workshop on Mobile Development Lifecycle*. 2015, pp. 17–24.

[32] Ermyas Abebe and Caspar Ryan. "Adaptive application offloading using distributed abstract class graphs in mobile environments." In: *Journal of Systems and Software* 85.12 (2012), pp. 2755–2769. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2012.05.091. URL: https://www.sciencedirect.com/science/article/pii/S0164121212001653.

[33] Jieyao Liu, Ejaz Ahmed, et al. "Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions." en. In: *Journal of Network and Computer Applications* 48 (Feb. 2015), pp. 99–117. ISSN: 10848045. DOI: 10.1016/j.jnca.2014.09.009. URL: https://linkinghub.elsevier.com/retrieve/pii/S1084804514002161 (visited on 04/13/2023).

[34] Dejan Kovachev and Ralf Klamma. "Framework for Computation Offloading in Mobile Cloud Computing." en. In: *International Journal of Interactive Multimedia and Artificial Intelligence* 1.7 (2012), p. 6. ISSN: 1989-1660. DOI: 10.9781/ijimai.2012.171. URL: http://www.ijimai.org/journal/node/361 (visited on 04/13/2023).

[35] Lei Yang, Jiannong Cao, Hui Cheng, and Yusheng Ji. "Multi-User Computation Partitioning for Latency Sensitive Mobile Cloud Applications." In: *IEEE Transactions on Computers* 64.8 (2015), pp. 2253–2266. DOI: 10.1109/TC.2014.2366735.

[36] Eduardo Cuervo et al. "Maui: making smartphones last longer with code offload." In: *Proceedings of the 8th international conference on Mobile systems, applications, and services*. 2010, pp. 49–62.

[37] Rajesh Krishna Balan et al. "Simplifying cyber foraging for mobile devices." In: *Proceedings of the 5th international conference on Mobile systems, applications and services*. 2007, pp. 272–285.

[38] Xiaohui Gu et al. "Adaptive offloading inference for delivering applications in pervasive computing environments." In: *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003.(PerCom 2003)*. IEEE. 2003, pp. 107–114.

[39] Ermyas Abebe and Caspar Ryan. "A hybrid granularity graph for improving adaptive application partitioning efficacy in mobile computing environments." In: *2011 IEEE 10th International Symposium on Network Computing and Applications*. IEEE. 2011, pp. 59–66.

[40] Lei Yang, Jiannong Cao, and Hui Cheng. "Resource constrained multi-user computation partitioning for interactive mobile cloud applications." In: *Technical reort. Department of Computing, Hong Kong Polytechnical University* (2012).

[41] Eduardo Cuervo et al. "Maui: making smartphones last longer with code offload." In: *Proceedings of the 8th international conference on Mobile systems, applications, and services*. 2010, pp. 49–62.

[42] Byung-Gon Chun et al. "Clonecloud: elastic execution between mobile device and cloud." In: *Proceedings of the sixth conference on Computer systems*. 2011, pp. 301–314.

[43] Ryan Newton et al. "Wishbone: Profile-based Partitioning for Sensornet Applications." In: *NSDI*. Vol. 9. 2009, pp. 395–408.

[44] Eli Tilevich and Yannis Smaragdakis. "J-orchestra: Automatic java application partitioning." In: *ECOOP 2002—Object-Oriented Programming: 16th European Conference Málaga, Spain, June 10–14, 2002 Proceedings 16*. Springer. 2002, pp. 178–204.

[47] Topcoder. *Edmonds-Karp and Dinic's Algorithms for Maximum Flow*. Date Accessed.

# Webography


[45] E-maxx Algorithms. *Edmonds-Karp Algorithm*. https://cp-algorithms.com/graph/edmonds_karp.html.

[46] *Ford-Fulkerson Algorithm*. https://brilliant.org/wiki/ford-fulkerson-algorithm/.

[48] *Java*. https://www.java.com/fr/.

[49] *Java Logo*. https://fr.wikipedia.org/wiki/Fichier:Java_Logo.svg.

[50] *Python*. https://www.python.org/.

[51] *Python Logo*. https://logos-world.net/python-logo/.

[52] *Flask Documentation*. https://flask.palletsprojects.com/en/2.3.x/. Accessed on April 3, 2023.

[53] *TensorFlow*. https://www.tensorflow.org/. Accessed on April 3, 2023.

[54] *JGraphT*. https://jgrapht.org/. Accessed on April 9, 2023.

[55] *Android Studio*. https://developer.android.com/studio.

[56] JetBrains. *PyCharm*. https://www.jetbrains.com/fr-fr/pycharm/.

[57] *StickPNG Image*. https://www.stickpng.com/img/download/58481537cef1014c0b5e4968#google_vignette.

[58] LogRocket. *LogRocket Blog*. https://blog.logrocket.com/. Accessed: January 8, 2023.

[59] The JGraphT Community. *JGraphT*. Website. URL: https://jgrapht.org/.