Reinforcement Learning for Energy-Efficient Cloud Offloading of Mobile Embedded Applications

Aditya Khune and Sudeep Pasricha

1 Introduction

Faster wireless network speeds and rapid innovations in mobile technologies have changed the way we use our computers. It is estimated that 207.2 million people in the United States own a smartphone today while the number of smartphone users worldwide is estimated to be more than two billion [1]. These mobile devices are not only used for making voice calls but also efficiently able to run complex mobile applications that interact with the Internet. The volume of data being accessed and processed by smartphones and the sophistication of mobile applications are rapidly increasing over time. However, the rapid evolution in hardware and software capabilities of mobile devices has not been paralleled by a similar advance in battery technology [2]. As expected, high-end mobile applications increase the burden on the battery life of smartphones. For example, it has been shown that a GPS-based smartphone app can drain a mobile phone's battery completely within 7 hours [3].

A promising solution that is being considered to support high-end mobile applications is to offload mobile computations to the cloud [4–11]. Offloading is an opportunistic process that relies on cloud servers to execute the functionality of an application that typically runs on a mobile device. The terms "cyber foraging" and "surrogate computing" are also sometimes used to describe computation offloading. Such computation offloading is being considered today as a means to save energy consumption (thereby improving battery lifetime) and increase the responsiveness of mobile applications. The potential of computation offloading lies in the ability to

A. Khune
Qualcomm, Longmont, CO, USA

S. Pasricha (✉)
Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO, USA
e-mail: sudeep@colostate.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_7

sustain power hungry applications by releasing the energy consuming resources of the smartphone from intensive processing requirements.

In this chapter, we present the details of a framework for mobile-to-cloud offloading that was first presented in [12]:

- We study the behavior of a set of popular smartphone applications, in both local and offload processing modes. We identify possible bottlenecks during mobile-to-cloud offloading, as a function of the applications functional characteristics, such as data intensiveness and computation intensiveness. This study is crucial to establish the pros and cons of offloading when using various wireless networks.
- We quantify the influence of different wireless network technologies on mobile-to-cloud offloading. We perform several experiments to gain a clear understanding of the impact of selecting the appropriate network when offloading, while considering advances in current high-speed wireless data communication networks such as 3G, 4G, and Wi-Fi.
- We propose a novel middleware framework that uses a machine-learning technique called reinforcement learning (RL) to make offloading decisions effectively on a smartphone. The proposed framework considers various types of information on the mobile device, such as network type, network bandwidth, and user-context, to decide when to offload in order to minimize energy consumption. Our strategy utilizes unsupervised machine learning to select between available networks (3G, 4G, or Wi-Fi) when offloading mode is active. Our experiments with real applications on a smartphone highlight the potential of our framework to minimize energy consumed in mobile devices.

2 Prior Work

Many prior research efforts have proposed strategies to reduce energy consumption in mobile devices via machine learning-based device resource management methods [13–18] and offloading strategies [4, 8, 19, 20]. Kumar et al. [19] presented a mathematical analysis of offloading. Broadly, the energy saved by computation offloading depends on the amount of computation to be performed (C), the amount of data to be transmitted (D), and the wireless network bandwidth (B). If (D/C) is low, then it was claimed that offloading can save energy. Flores et al. [8] proposed a fuzzy decision engine for code offloading. The mobile device uses a decision engine based on fuzzy logic to combine various factors and decide when to offload. *Our framework discussed in this chapter considers many more factors than these works, such as network type, data size, and degree of computations when making decisions about offloading.* Cuervo et.al [4] proposed a system called MAUI, based on code annotations to specify which methods from a software class can be offloaded. Annotations are introduced in the source code by the developer during the development phase. At runtime, methods are identified by a MAUI profiler, which performs the offloading of the methods, if the bandwidth of the network and data

transfer conditions are ideal. MAUI aims to optimize both energy consumption and execution time, using an optimization solver. However, this annotation method puts an extra burden on the already complex mobile application development phase. Moreover, such annotations can cause unnecessary code offloading that drains energy [20]. *To reduce the complexity of the application development process, we recommend transferring the entire application processing to the cloud rather than utilizing a design-time code partitioning method. Further, we propose a novel adaptive reward-based machine learning approach to make smart offloading decisions that can achieve high energy efficiency with offloading and also improve application response time.*

3 Challenges with Offloading

In spite of existing research highlighting the potential of offloading in mobile devices, current offloading techniques are far from being widely adopted in mobile systems. The implementation of these computation offloading techniques for many real-world mobile applications in real-world scenarios has not shown promising results [6], with the mobile device consuming more energy in the offloading process than the energy savings achieved due to computing on servers in an offloaded manner.

Offloading decision engines must consider not only the potential energy savings from offloading but also how the response time of the application is impacted by offloading. An effective decision to offload processing to the cloud must reduce energy without significantly increasing response time. Such decisions are heavily impacted by wireless network inconsistency. The power consumed by the network radio interface is known to contribute a considerable fraction of the total device power, and it varies depending on wireless signal strength [21]. With the recent advent of high bandwidth 4G networks, there has been increased interest in the offloading domain, but from our experiments and results presented in later sections of this chapter, we found that 4G consumes more energy than Wi-Fi and 3G. Some of the prior works [22] in this area also confirm this observation.

The network quality of a 4G connection at a mobile device's location greatly affects the battery life. If the device is in the area that does not have 4G coverage, there is no advantage to a 4G interface, and if 4G network search is not disabled, then the radio's search for a nonexistent signal will drain the battery quickly. In case of a weak signal, the device uses more power to send and receive data to and from the network. A strong 4G signal uses less battery, but the biggest problem is the constant switching from 4G to 3G and back again. Also, throughout a typical day, at different times, the performance of a wireless network varies because of changing traffic load on the network. We refer to all such problems due to the mobile network as "network inconsistency" problems.

To counter the impact of network inconsistency on a mobile device and to optimize the offloading experience, we propose a novel offloading framework based

on reinforcement learning. This framework not only decides when to offload but also helps a mobile device select between the different available wireless networks, to achieve consistent improvements by using offloading even in the presence of varying network conditions. In the following sections, we describe our framework in detail.

4 Offloading Performance of Mobile Applications

We analyzed the performance implications of offloading by comparing two scenarios – one where all computations are performed only on the mobile device without using the cloud at all (local mode) and the other where there was a complete reliance on the cloud computation (offload mode), with minimal computations on the mobile device. We selected five diverse and popular commercially available smartphone applications for our experiment. Our evaluation focuses on two metrics: (i) battery consumption and (ii) response time. We compared the results obtained with these applications for 3G, 4G (HSPA+), and Wi-Fi networks. This comparative study was meant to help us identify various factors that need to be considered for the design of cloud offloading strategies for mobile applications, for example, identifying the best possible network for offloading to the cloud, for a given mobile application, at a specific location.

4.1 Experimental Setup

The power estimation models required to estimate battery consumption were built using power measurements on the LG G3 device running the Android OS version 5.0.1. The contact between the smartphone and the battery was instrumented, and current was measured using the Monsoon Solutions power monitor [23] (Fig. 1). The monitor connects to a computer running the Monsoon Solutions power tool software, which allows real-time current and power measurements. We also used the Android Device Bridge, a software tool to perform battery drain measurements on the android device. The experiments were performed using AT&T's 3G and 4G (HSPA+) networks and Comcast's 100 Mbps (2.4 GHz Band) Wi-Fi network. We performed these experiments around the Colorado State University campus in Fort Collins, Colorado, the United States.

Before conducting our experiments, we followed a few rules to ensure meaningful and accurate results while avoiding human error. These rules are as follows: (1) set the device's screen to a consistent and fixed brightness level, to minimize interference from varying screen power consumption (e.g., for different ambient light scenarios); in our measurements, we used the lowest screen brightness

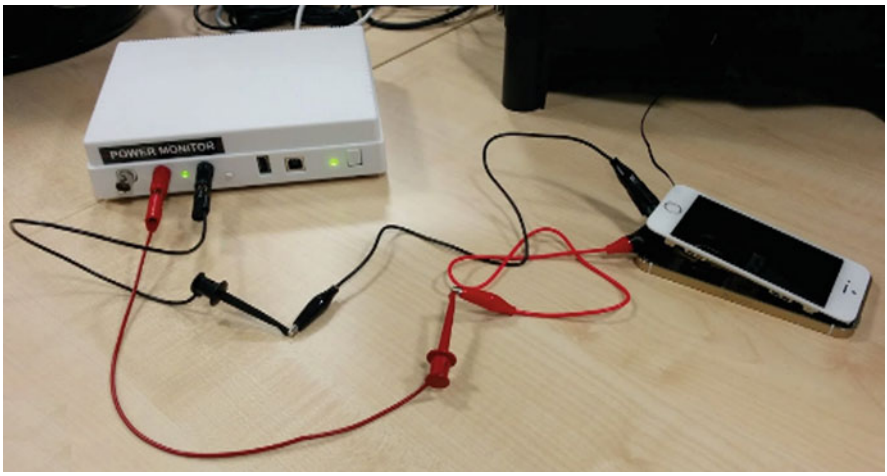

Fig. 1 Monsoon power monitor setup

level; (2) kill all background processes before measurements; and (3) repeat each experiment over 15 iterations to improve result confidence and minimize human error. We selected five diverse commercially available smartphone applications for our experiments: (i) matrix operations, (ii) Internet browser; (iii) zipper (file compression); (iv) voice recognition and translation; and (v) torrent (file download).

The next subsection gives the details of all the applications considered and the results of their execution for the two scenarios (local and offloading modes) outlined earlier.

4.2 Experimental Results

4.2.1 Matrix Operation App

The matrix calculator app [24] runs on android-based devices. The user is first asked to enter the size of the matrix and all the digits of the matrix manually, and then the user can direct the application to calculate the inverse of that matrix (using the adjoint method). For our experiments, we used a set of matrix sizes from 3×3 to 9×9. For the cloud part, we implemented the functionality of calculating the matrix inverse using the Amazon Web Services (AWS) EC2 cloud instance [25]. Figure 2 shows the results from our experiment. The energy consumption in local processing mode is equal to the battery drain in the device while performing the matrix operation, whereas in the cloud mode, energy consumption is the total of battery drain during the idle time of the mobile device while the operation is being performed remotely on the cloud and the time for data transfer between the mobile device and the cloud.

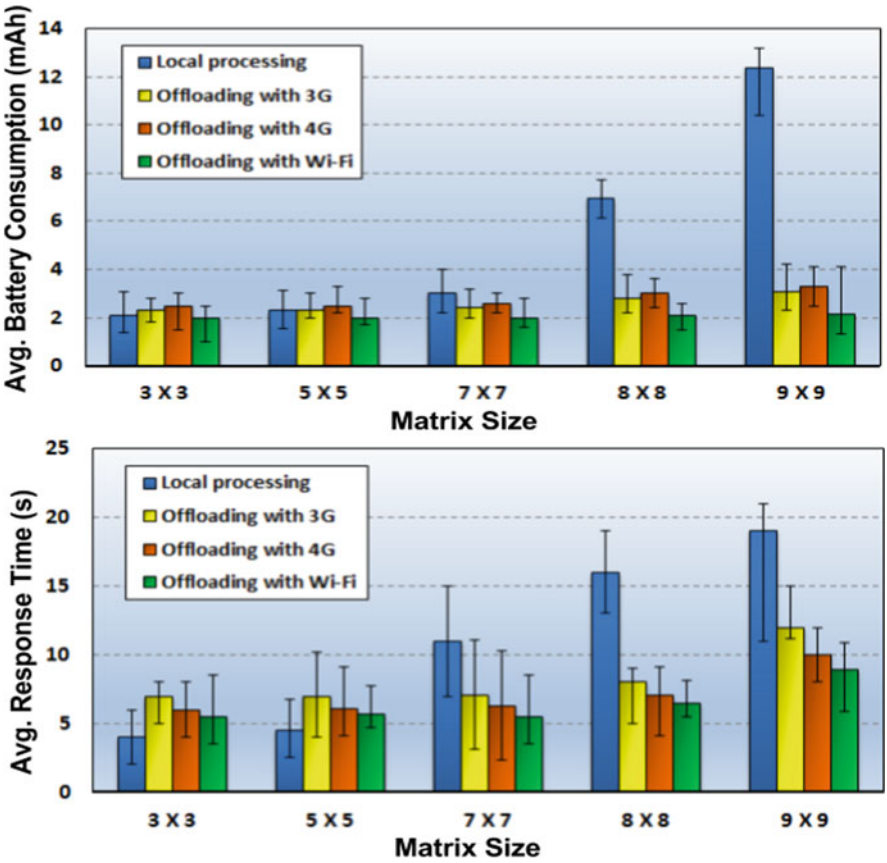

Fig. 2 Average battery consumption and average response time on a mobile device for a matrix operation with varying matrix sizes

It can be observed that in the local processing mode, the battery consumption of the device increases manifolds with the increasing matrix size, largely, because there is an increase in the CPU's energy consumption as the number of floating point operations increase. Local processing is found to be suitable for operations on small matrices (i.e., 3×3 and 5×5) allowing for low energy consumption on the device and low response time. On the other hand, offloading to the cloud saves energy and reduces response time when the matrix size increases. The device in offloading mode saves maximum energy (and also has minimum response time) when used with Wi-Fi. The results show that 3G performs slightly better than 4G as far as energy is concerned, whereas 4G gives better response time than 3G for the same operations.

4.2.2 Internet Browser App

Cloud-based web browsers use a split architecture where processing of a mobile web browser is offloaded to the cloud partially. This involves cloud support for most browsing functionalities such as execution of JavaScript, image transcoding and compression, and parsing and rendering of web pages. For our experiments, we used the Mozilla Firefox [26] and Puffin [27] browsers. Puffin is a commercially available cloud-based mobile browser, and Mozilla Firefox is a local browser available from the Google Play store. Our experiments are performed for a data range starting as low as 150 Kib to a session involving 5 MB of data transfer to load the web pages. Figure 3 shows the results obtained by measuring data transfer (response) time and energy consumed by these browsers for loading two different websites: (i) www.yahoo.com and (ii) www.wikipedia.org.

We observed that the results obtained fluctuated significantly due to network inconsistency. For example, the plots in Fig. 3 show that the response time/battery consumption of a browser session with around 3 MB data usage is sometimes more than that of a session which uses 5 MB of data. To counter such network inconsistency problems, we conducted 15 iterations of each experiment across different locations and at different times of the day. In general, our results show that cloud-based web browsers are faster but more expensive in terms of energy consumption. For small data transfers, it is suitable to use web browsers with local processing to save energy. For a typical user, the data transfer amount during a browsing session does not go beyond 5–6 MBs for a single session. Thus, for most websites in typical usage scenarios, a local browser will provide greater energy savings than when using offloading. The response time results indicate that for larger data usage scenarios, offloading can be beneficial. Fourth generation not only provides lower response time but also consumes more energy than 3G for the offloading scenarios. Wi-Fi outperforms both 3G and 4G in offloading mode, for response time and energy consumption.

4.2.3 Zipper App

Zipping large files in order to compress them is a widely used functionality on most computers. Zipper [28] is an android app that compresses files locally on a mobile device. For the cloud-based file compression, we used an AWS cloud instance and zipping tool available on the web [29]. Figure 4 shows the results for energy consumption and the response time when zipping various PDF and Word document files ranging in size from 15–255 MB. It can be observed that for the zipping operation, local computation is most efficient in terms of energy consumption. Offloading provides benefits only in response time and that too only for large file sizes. When offloading, 4G consumes more energy than 3G for smaller file sizes (15–105 MBs) whereas 3G consumes more energy than 4G for larger file sizes (175–255 MBs). Fourth generation is faster than 3G but slower than Wi-Fi. Wi-Fi gives the best results in terms of energy and response time when offloading.

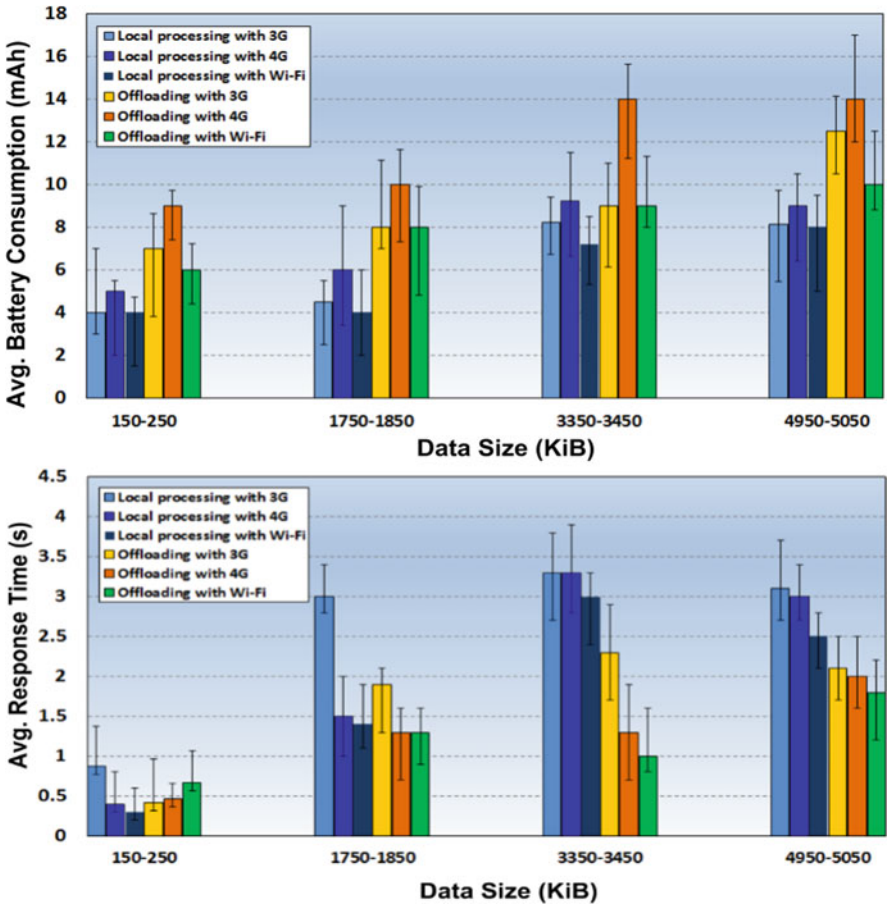

Fig. 3 Average battery consumption and response time on a mobile device for an Internet browsing session with varying data sizes

4.2.4 Voice Recognition and Translation App

There are several popular apps for voice recognition and translation available from app stores, for example, Google Translate [30] for android and Speak and Translate [31] for iOS. Google Translate is a cloud-based app, which also has an offline translation mode that performs local processing on the device with a small neural network. The application allows for downloading an installation package to support the local processing mode. It makes use of the statistical machine translation method, which relies on large amounts of data to train a machine translation engine.

Figure 5 shows the energy consumption of the Google Translate app for a range of words. These measurements were recorded while translating 20–140 words from the English (the United States) to Marathi language. From the results in

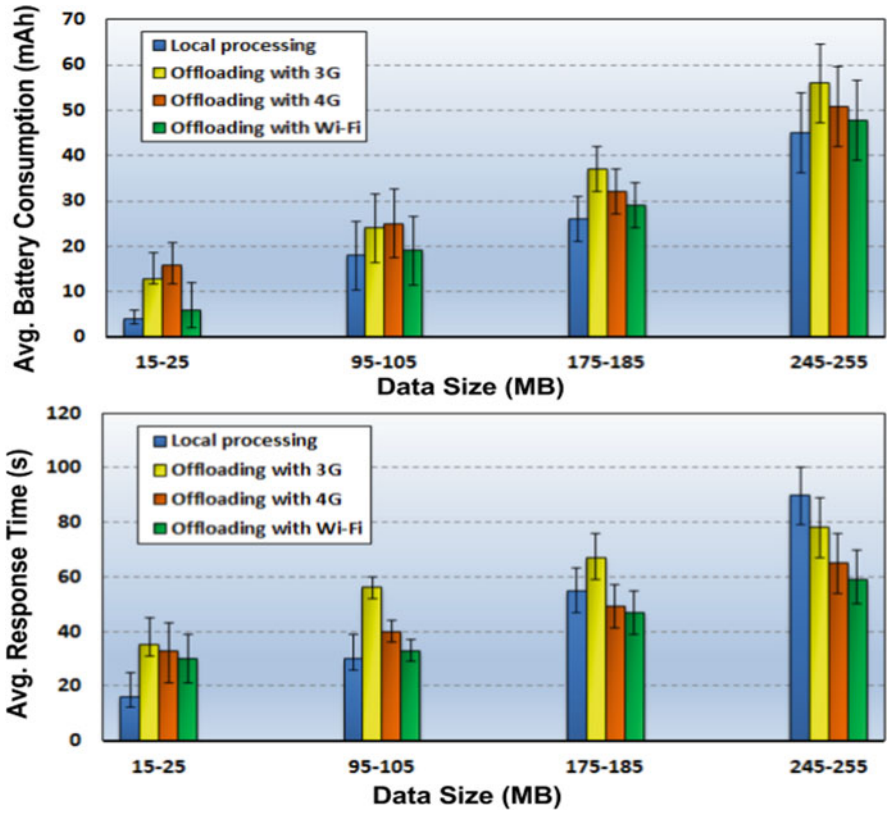

Fig. 4 Average battery consumption and response time on a mobile device for zipping/compressing files of varying sizes

Fig. 5, we can clearly observe that the local processing mode is more efficient in terms of energy consumption as compared with the cloud processing mode. The voice recognition and translation accuracy for local processing was 79.26% and for offloaded processing was 88.51%. This is because the offloaded voice data is processed by more powerful cloud servers, which are capable of running the complex computations of a larger neural network and other machine learning algorithms for more efficient translation.

4.2.5 Torrents App

We used the android-based torrent app Flud [32] to perform torrent downloads in local mode. In the cloud mode, a cloud server is used as a BitTorrent client to download torrent pieces on behalf of a mobile device. While the cloud server downloads the torrent, the mobile device switches to the sleep mode until the

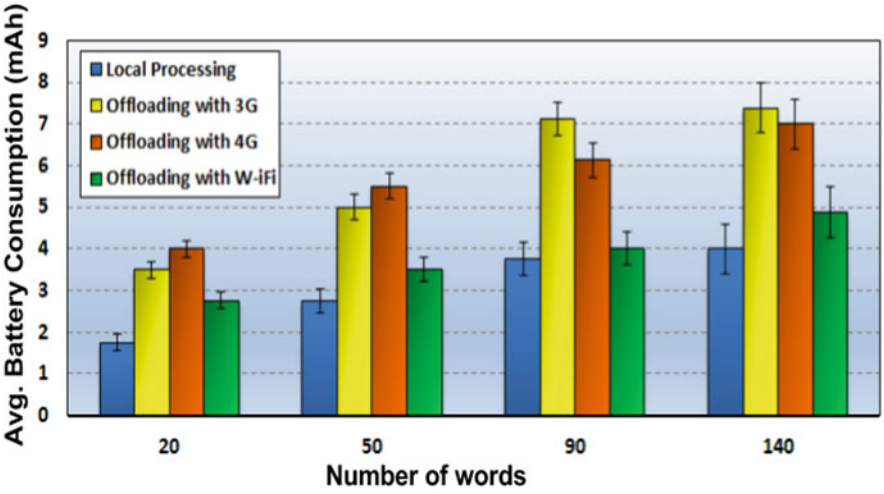

Fig. 5 Average battery consumption on a mobile device for voice recognition and translation operations

cloud finishes the torrent processes, and then the cloud uploads the downloaded torrent file in a single process to the mobile device. Kelenyi et al. [33] presented a similar strategy for torrent file download. This strategy saves energy consumption in smartphones as downloading torrent pieces from multiple peers consumes more energy than downloading one burst of pieces from the cloud.

For our experiments, we used torrent file sizes ranging from 25–85 MB, with an AWS cloud instance being used for the cloud mode. Figure 6 shows the results of our experiments for this application. It is interesting to note that out of all the applications that we consider, offloaded processing proves to be most beneficial in terms of both energy savings and response time for the torrent download application, which is data intensive but not compute intensive. Fourth generation is faster than 3G but slower than Wi-Fi, which is consistent with earlier observations. Fourth generation performs slightly better than 3G in terms of energy consumption for higher data sizes (45–85 MBs), but for smaller data sizes, 3G is more energy efficient.

4.3 Summary of Findings

The overall performance when offloading depends on various factors such as the amount of data required by the application, wireless network signal type and strength and the functionality of the application under consideration. In some prior work [19, 34], it was concluded that offloading is beneficial when an application is compute intensive and at the same time less data intensive. However, we found

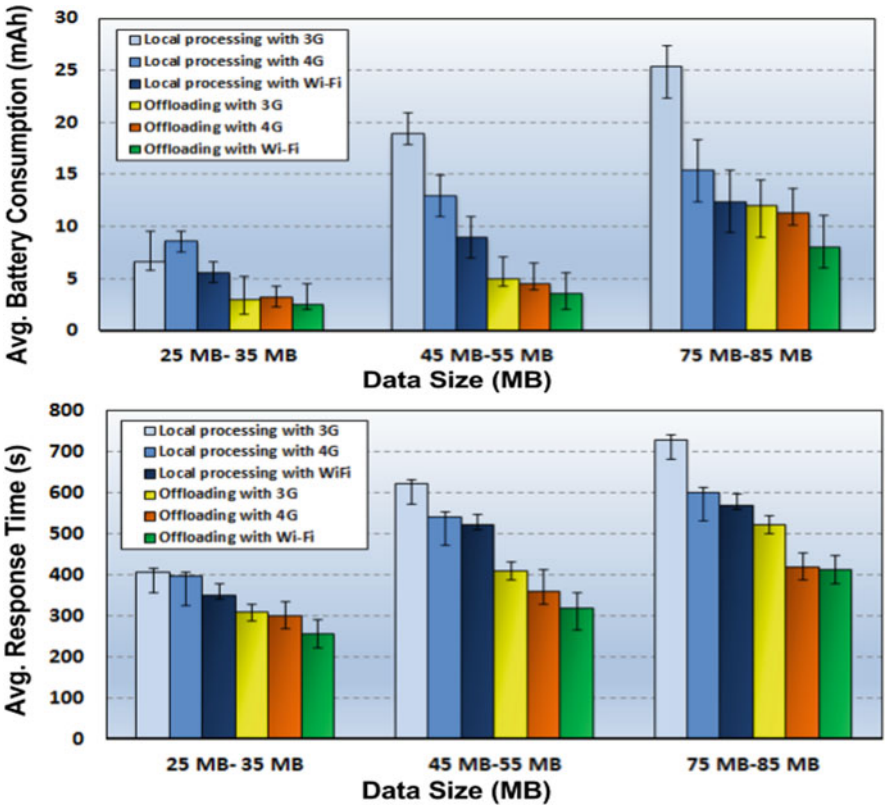

Fig. 6 Average battery consumption and response time on a mobile device for torrent file download operations

that this is not always the case. For instance, offloading is beneficial for applications that may not be compute intensive, but are data intensive, for example, the torrent application.

To make offloading more practical, it is important to reduce the energy spent in the communication between the mobile device and the cloud. Our experiments indicate that choosing the best possible network for offloading is a critical decision. One may assume that because 4G is faster than 3G, we should always rely on it for offloading when Wi-Fi is not available. However, our results indicate that 4G is more power hungry than 3G most of the time.

Network quality is also a factor that cannot be ignored. We found that a good 3G-coverage performs far better as opposed to poor 4G-coverage and vice versa. In the region of cell tower edges or where the coverage of 3G/4G ends, we found that the handover process results in high battery drain. This is because the device in such scenarios is constantly searching for the network, frequently scanning the wireless spectrum around it to determine which tower it should tether itself to. The more

networks there are to choose from, the longer the scans take. Some apps require a channel to be established between the base station and the mobile device at regular intervals, which can significantly drain the device battery.

Another observation is that as 4G generally provides faster data rates than 3G, users tend to consume more data when connected on 4G than 3G. The radio-network interface in the 4G (or LTE) device is functionally a lot more sophisticated and does a lot more than a 3G interface. This interface is the single biggest source of battery drain in a mobile device, apart from its display. Unlike the display, however, the network interface radio is always on.

In conclusion, we observed from our experiments on real applications running on a real mobile device that the overall performance of offloading depends on various factors, such as the amount of data used by the application, network signal type (3G, 4G, and Wi-Fi), network signal strength, and the complexity of the functionality of the application under consideration.

5 Adaptive Offloading

The decision to offload a mobile application to the cloud is a complex one due to the distributed nature and many real-time constraints of the overall system. To make an effective offloading decision, it is vital to consider various factors as we discovered after our experimental analysis presented in the previous section. As these factors vary at runtime, there is a need for an adaptive offloading approach that takes the variations of these factors at runtime into consideration when making decisions.

A few prior works [8, 9] propose an offloading decision engine that considers the contextual parameters on a device and on the cloud to make an offloading decision adaptively. Flores et al. [8] proposed a fuzzy decision engine for code offloading. The mobile device runs the fuzzy logic decision engine, which is utilized to combine n number of variables (e.g., application data size and network bandwidth) that are obtained from the overall mobile cloud architecture. The fuzzy logic decision engine works in three steps, namely: fuzzification, inference, and defuzzification. In fuzzification, input data is converted into linguistic variables, which are assigned to a specific membership function. A reasoning engine is applied to the variables, which makes an inference based on a set of rules. Lastly, the outputs from the reasoning engine are mapped to linguistic variable sets again in the defuzzification step. This offloading decision engine in [8] assumes a consistent network performance during offloading. However, as observed in our experiments, such consistency is difficult to achieve because of frequent mobile user movements and variable network quality (due to factors such as location of the device and load on the network [21]). Moreover, the offloading decision engine in [8] mainly emphasizes energy savings; however, response time is also a crucial metric for various applications that should not be ignored, otherwise user quality of service degradation can become so severe that any effort to save energy becomes irrelevant.

In the next section, we describe our reward-based middleware framework for adaptive offloading that overcomes the challenges mentioned above, to make more efficient decisions related to when and how to offload applications from a mobile device to the cloud.

6 Middleware Framework for Efficient Offloading of Mobile Applications

To simplify the mobile application development process and at the same time avoid problems caused by hard coded annotations, our framework proposes to transfer all the computation for an application to the cloud instead of partial (selective) offloading of the application. Our framework involves a novel decision engine on the mobile device that works together with a clone virtual machine (VM) of the mobile software environment to execute applications on cloud servers. Figure 7 shows a high-level overview of the proposed framework. The framework is implemented at the middleware level in the software stack of the Android OS and runs in the background as an android service. As a result, our framework requires no changes to any of the applications or the Android OS. The runtime monitor component periodically triggers the reinforcement learning (RL) module to generate/update a Q-learning table. At any time, this Q-table contains information to guide the decision for when and how to offload an application to the cloud, depending on multiple factors. The remainder of this section provides a detailed overview of the RL mechanism and our algorithm to generate and use the Q-table.

6.1 Reinforcement Learning (RL)

RL is an unsupervised learning approach, which focuses on learning by having software agents interact with an environment and then taking actions to maximize some notion of a reward. In supervised learning (e.g., using neural networks), a training set of correctly identified observations is required to train a prediction model. RL differs from supervised learning in that correct input/output pairs of identified observations do not need to be presented, so there is no need for a pretrained model. Moreover, an RL algorithm performs well as it has better exploration capabilities than unsupervised learning methods. For this reason, RL is being widely used in gaming and control problems, for example, to determine the next best move in games [35, 36]. RL cuts down the need to manually specify rules, and agents learn simply by playing the game or exploring different moves in an automated manner.

In RL, the state-action value function is a function of both state and action, and its value is a prediction of the expected sum of future reinforcements. The state-

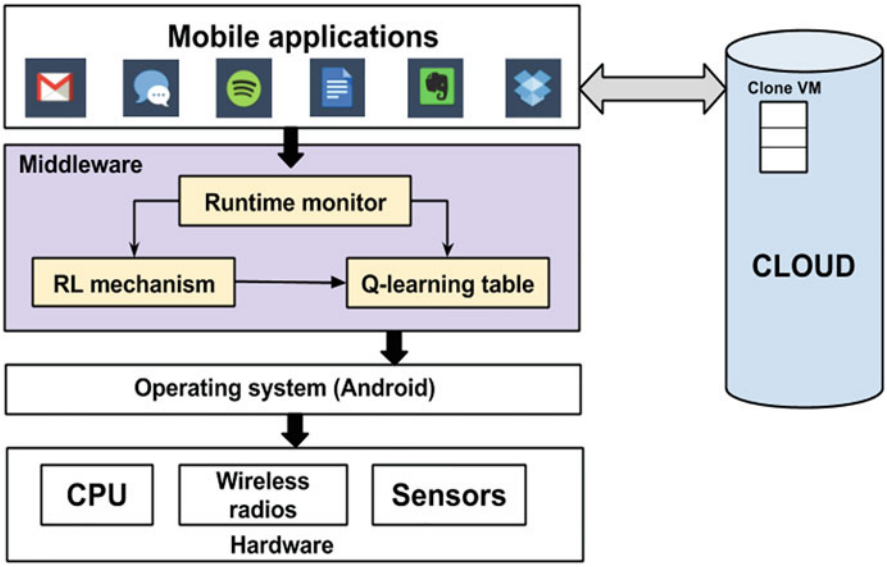

Fig. 7 Reinforcement learning (RL)-based middleware framework for efficient application offloading to cloud

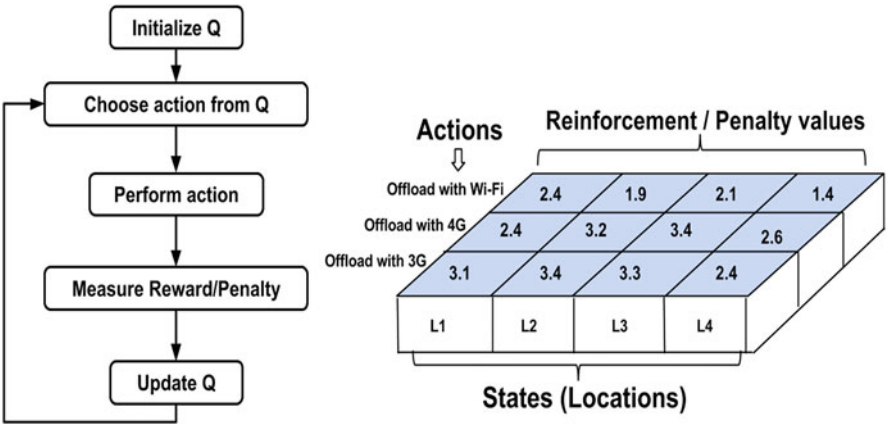

Fig. 8 Q-learning flow with example of Q-table

action value function is referred to as the Q-function [37]. Figure 8 summarizes how a typical Q-learning reinforcement algorithm works. Q-learning is a reward-based mechanism that generates a Q-table with reinforcement or penalty values. The figure illustrates a section of a Q-table where the possible actions are offloading with 3G, 4G, or Wi-Fi network, when the user is at different locations L1–L4. Actions are chosen, and the penalty values are calculated for respective actions to update the Q-table.

Suppose the system is at a defined state s_t at time t. Upon taking action a_t from that state, we observe the one step reinforcement r_{t+1}, and the next state becomes s_{t+1}. This continues until we reach a goal state, K steps later. The reward R_t in this goal state is shown below:

$$R_t = \sum_{k=0}^{K} r_{t+k+1} \tag{1}$$

The objective with RL is to find actions a_t that maximize (or minimize) the sum of reinforcements or rewards r_t in Eq. (1). This can be reduced to the objective of acquiring the Q-function $Q(s_t, a_t)$ that predicts the expected sum of future reinforcements, where the correct Q-function determines the optimal next action. So, the RL objective is to make the following approximation as accurate as possible:

$$Q\left(s_t, a_t\right) \approx \sum_{k=0}^{\infty} r_{t+k+1} \tag{2}$$

The Q-function stores reinforcement values for each state and action pair of the system. Eq. (2) formulates the RL for a multistep decision problem (e.g., predicting sequential actions in a Tic-Tac-Toe game [37]). In our middleware framework, we use RL for a single-step decision problem as there are no sequential states that are dependent on the previous state of the system. This version of the problem is formulated as:

$$Q\left(s_t, a_t\right) \approx \sum_{t=1}^{n} r_t \tag{3}$$

The Q-function is ultimately queried by the system to select the optimal action a_t, in state s_t:

$$a_t = \arg\ \min\ Q\left(s_t, a\right) \tag{4}$$

6.2 RL Algorithm to Generate Q-Function

The state of a mobile device is defined using the contextual information of the device such as its location, available network type, and network strength. These contextual factors are chosen as we consider them to be crucial for efficient offloading. The runtime monitor extracts the contextual information of the device to form state values of the system. For example, consider a mobile device that is at location L1, where it has access to a 3G network type with "strong" network strength. From this state, if an application processing needs to be offloaded, then the Q-function

is called to select the appropriate network that would result in the least penalty in terms of energy or response time (or both). In our framework, the following state and action values are used to generate the Q-function:

Set of state values (discrete values):

- Location = L1, L2, L3, . . . , L*n*
- Network carrier = 3G, 4G, Wi-Fi
- Network strength = Strong, Medium, Weak
- Data Size = data_small, data_medium, data_large

Set of action values

- Offload using 3G network
- Offload using 4G network
- Offload using Wi-Fi network

The location L1-Ln can be any geographic area where the user uses the offloading application, for example, office and home. More state-action pairs can be added to the above list to account for factors that might affect offloading, for example, we can add "Time of Day" as another state value, as it is observed that network performance is slow at certain times of day when the network load is high. However, a larger set of state-value pairs will result in a larger Q-function requiring greater overhead to manage it.

The Q-function is generated as follows: Initially, when the mobile device is at a location L1, the runtime monitor accesses contextual information from the device such as location, networks available, and network strength. A small data file is then uploaded from the mobile device to the cloud, using the primary network carrier. The battery consumed and total response time taken for this operation are measured. The uploading operation is repeated with varying (small, medium, and large) data sizes with all available networks at the location (3G, 4G, and Wi-Fi) activated one by one. For each of these uploading operations, the runtime monitor measures the battery amount consumed and response time to complete the operation. The Q-table is then populated with the penalty values calculated using Eqs. (5), (6), and (7):

$$P_{3G} = P_{b3G} * x + P_{t3G} * y \tag{5}$$

$$P_{4G} = P_{b4G} * x + P_{t4G} * y \tag{6}$$

$$P_{WiFi} = P_{WiFi} * x + P_{WiFi} * y \tag{7}$$

Thus, in our RL framework, the reinforcement values are essentially the penalty values P_{3G}, P_{4G}, and P_{WiFi}. The set of possible individual penalty values are shown in Table 1. Once populated, the Q-table can be updated periodically in the background when the user is not actively using the device. In Eqs. (5), (6), and (7), to optimize battery consumption and response time, we used weights x and y, respectively, with penalty values. Both x and y parameters take values between 0

Table 1 Penalty values in RL algorithm

Penalty values	Offload w/ 3G	Offload w/ 4G	Offload using Wi-Fi
Battery (P_b) processing	P_{b3G}	P_{b4G}	P_{bWiFi}
Response time (P_t)	P_{t3G}	P_{t4G}	P_{tWiFi}
Total penalty	P_{3G}	P_{4G}	P_{WiFi}

Fig. 9 Decision-making using Q-table (vector of key value pairs)

and 1. For our experiments in Sect. 7, we used $x = 0.5$ and $y = 0.5$ to balance minimizing battery consumption and response time.

Figure 9 shows an example of the decision-making process with the help of two simple scenarios. For a data intensive application at location L1, we have 3G and 4G networks available as shown in first two lines of the Q-table in the figure. The penalty value for 4G at location L1 is lesser; therefore, the 4G network is selected for offloading the application to the cloud. For a less data intensive application at location L2, out of all the networks available, 3G is selected because Wi-Fi has weak signal strength with higher penalty and 4G also has a higher penalty.

7 Experimental Results

To evaluate the efficacy of our proposed framework, we conducted a set of experiments. We implemented our middleware framework and its decision engine on an android-based mobile device. To form the Q-function of our RL algorithm, real user data was collected at different geographical locations around the Colorado

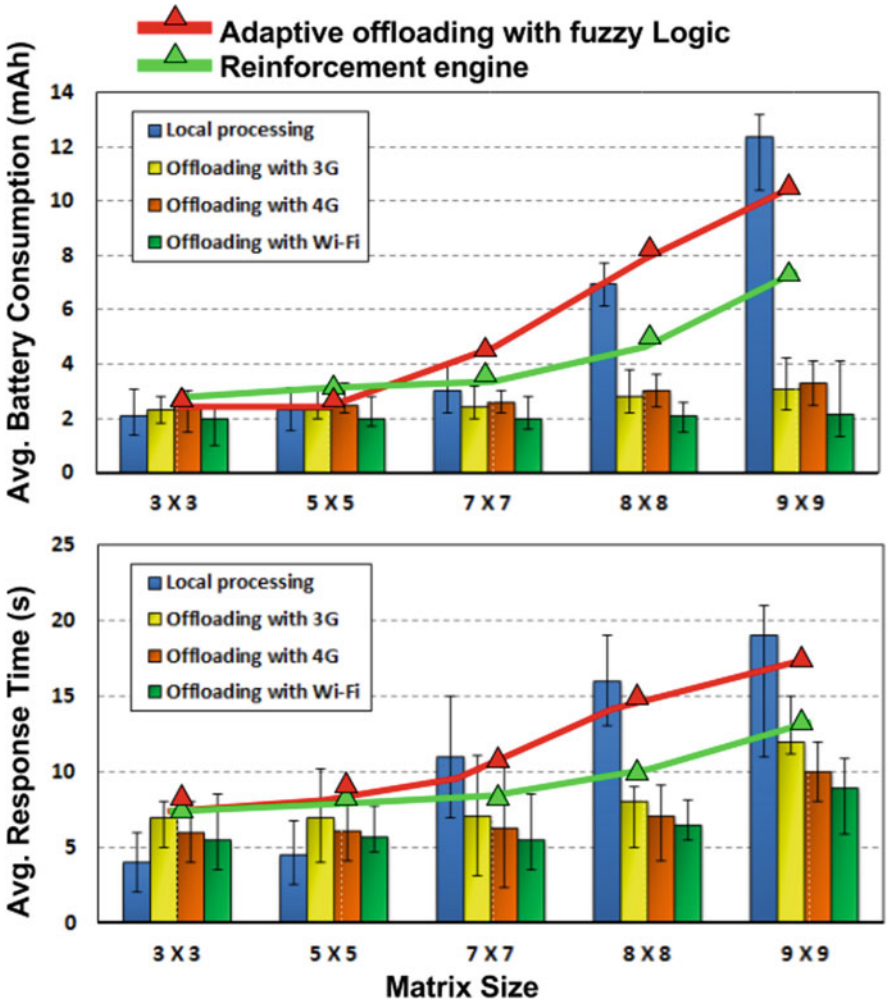

Fig. 10 Average battery consumption and response time of Matrix operations app with learning methods

State University campus area, in Fort Collins, Colorado. We compared our work with the fuzzy logic decision engine proposed by Flores et al. [8], which we discussed in Sect. 5 and which we also implemented on the android-based mobile device.

Figure 10 shows the results for the matrix operation app with our proposed RL-based decision engine and the fuzzy logic-based decision engine from [8]. Similarly, Fig. 11 shows the results for the zipper app, and Fig. 12 shows results for the torrent app. In all the scenarios, the task of a decision engine is to decide whether to offload and select the network to offload with. In these figures, the red trend line shows

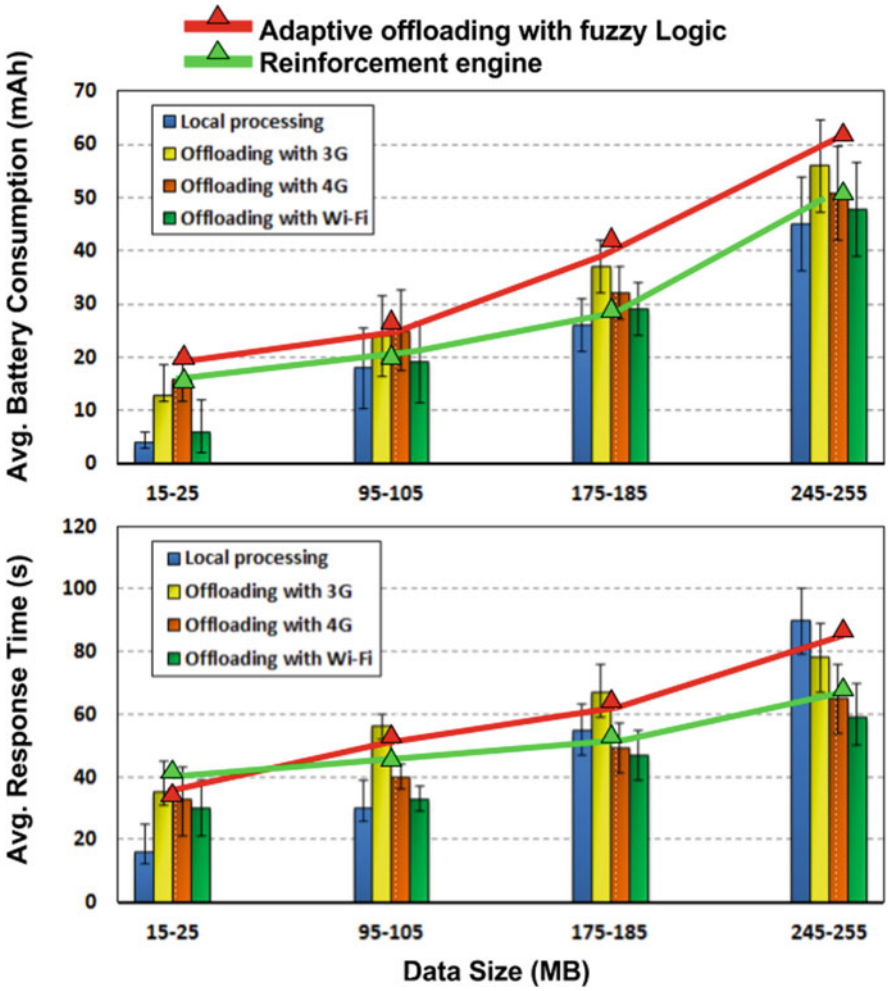

Fig. 11 Average battery consumption and response time of zipper app with learning methods

results with the fuzzy decision engine [8] whereas the green trend line shows the results with our RL-based middleware framework. We have also shown bars with the results for offloading with each available network and local processing (from Sect. 4) as a reference.

In general, our results show that our proposed RL-based decision engine outperforms the fuzzy logic approach from [8]. For less data intensive operations, the results of RL and fuzzy logic overlap. For instance, in the case of the zipper application (Fig. 11), for lower data sizes fuzzy logic shows better results, possibly because the Q-table generated using our RL algorithm uses 25 MB as the minimum data size. For any data size lower than this minimum value, the RL-based framework

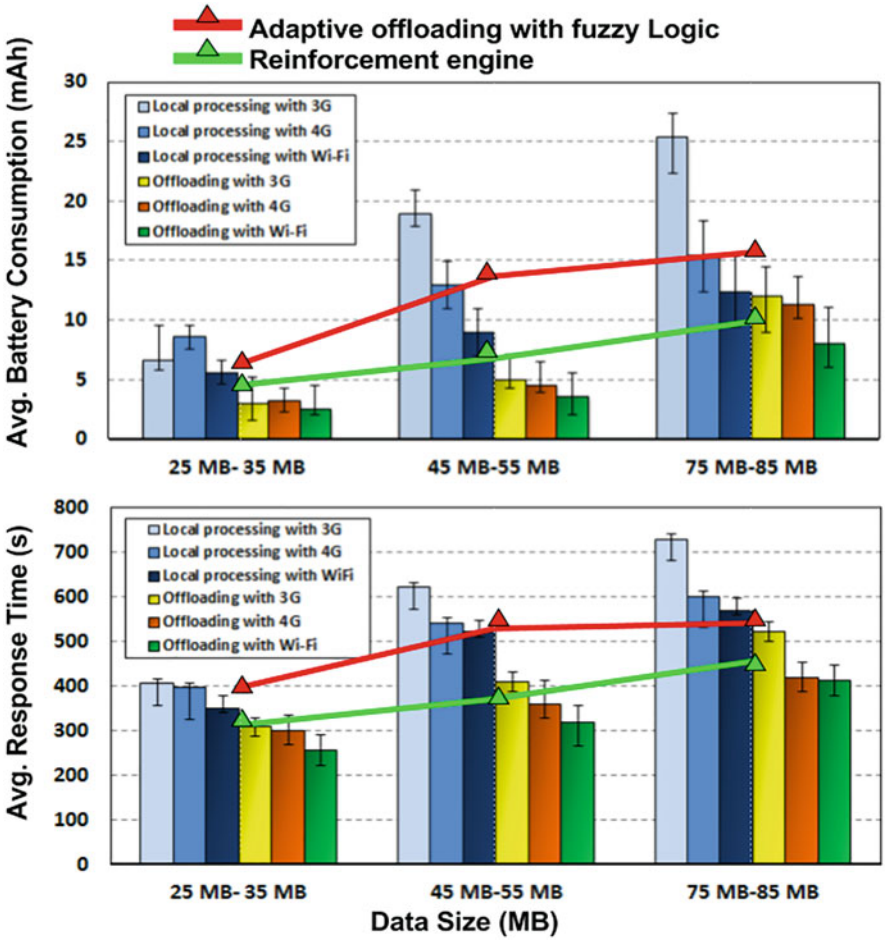

Fig. 12 Average battery consumption and response time of torrent app with learning methods

is thus less effective at making predictions. This can be improved using a wider range of data files/sizes when populating the Q-table. For higher data sizes and more complex computations, our RL approach gives improved battery consumption and response time than [8].

Figure 13 summarizes the prediction accuracy of both the learning methods being compared. It can be observed that our RL-based engine has better prediction accuracy, which is crucial for making effective offloading decisions. The overall performance of offloading depends on various factors, such as the amount of data required by the application, network signal type (3G, 4G, and Wi-Fi) and network signal strength, and the complexity of the functionality of the application under observation. By considering all of these individual factors in the decision process, unlike the fuzzy logic approach from [8], and by utilizing a more sophisticated and

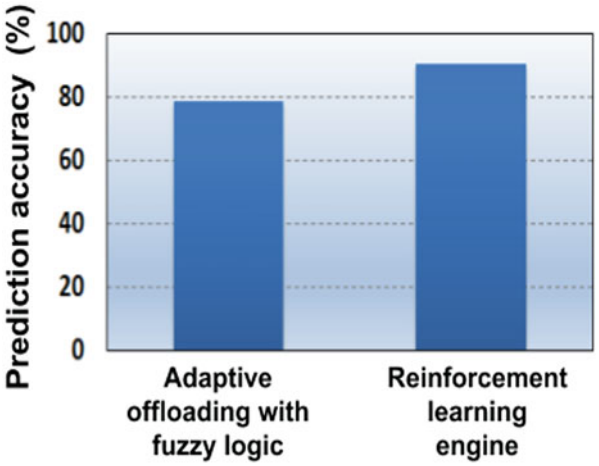

Fig. 13 Prediction accuracy of learning methods

powerful learning algorithm, our framework is able to achieve notably better results compared with [8]. Our results show that proposed RL-based offloading system can save up to 30% battery power with up to 25% better response time as compared with the fuzzy logic-based approach.

8 Conclusions and Future Work

In this chapter, we analyzed real mobile applications to determine the benefits of application offloading. We found that overall performance with offloading depends on various factors such as the amount of data and type of usage, available network carrier, and signal strength. These factors should be considered while making a decision to offload a mobile application. To make offloading more practical, it is important to reduce the energy spent in the communication between the mobile device and the cloud. In our experiments, we compared energy consumption in mobile devices for varying network types (3G, 4G, and Wi-Fi). This comparison shows that selecting an appropriate wireless network for offloading is crucial. We subsequently presented a novel network-aware mobile middleware framework based on reinforcement learning to accomplish energy-efficient offloading in smartphones. Our results show that we can save up to 30% battery power with up to 25% better response time when using our proposed framework compared with a state-of-the-art fuzzy logic-based offloading approach from prior work. As part of future work, researchers can consider the evaluation of a more diverse set of mobile applications and characterizing their bottlenecks, explore new algorithms for low-overhead offloading decision-making on smartphones and other mobile devices

(e.g., wearables), and consider the characterization and use of additional wireless networks for offloading, such as emerging 5G networks.

Acknowledgments This work was supported by the National Science Foundation (NSF), through grant CNS-2132385.

References

1. The Statistical Portal. [Online]. Available: www.statista.com/statistics (2016). Accessed 7 Mar 2022
2. Ali, F.A., Simoens, P., Verbelen, T., Demeester, P., Dhoedt, B.: Mobile device power models for energy efficient dynamic offloading at runtime. J. Syst. Softw. **113**, 173–187 (2016)
3. Gaonkar, S., Li, J., Choudhury, R.R., Cox, L., Schmidt, A.: Micro-blog: sharing and querying content through mobile phones and social participation. In: Proceedings ACM Mobisys, pp. 174–186. ACM (2008)
4. Cuervo, E., Balasubramanian, A., Cho, D.K., Wolman, A., Saroiu, S., Chandra, R., Bahl, P.: Maui: making smartphones last longer with code offload. In: Proceedings ACM Mobisys, pp. 49–62. ACM (2010)
5. Chun, B.G., et al.: Clonecloud: elastic execution between mobile device and cloud. In: ACM EuroSys, pp. 301–314. ACM (2011)
6. Flores, H., Srirama, S.: Mobile code offloading: should it be a local decision or global inference? In: Proceedings ACM Mobisys, pp. 539–540. ACM (2013)
7. Kosta, S., et al.: Thinkair: dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: Proceedings IEEE INFOCOM, pp. 945–953. IEEE (2012)
8. Flores, H.R., Srirama, S.: Adaptive code offloading for mobile cloud applications: exploiting fuzzy sets and evidence-based learning. In: Proceedings ACM Mobisys, pp. 9–16. ACM (2013)
9. Khairy, A., et al.: Smartphone energizer: extending smartphone's battery life with smart offloading. In: IEEE IWCMC, pp. 329–336. IEEE (2013)
10. Banga, G., Crosby, S., Pratt, I.: Trustworthy computing for the cloud-mobile era: a leap forward in systems architecture. IEEE Consum. Electron. Mag. **3**(4), 31–39 (2014)
11. Corcoran, P., Datta, S.K.: Mobile-edge computing and the internet of things for consumers: extending cloud computing and services to the edge of the network. IEEE Consum. Electron. Mag. **5**(4), 73–74 (2016)
12. Khune, A., Pasricha, S.: "Mobile network-aware middleware framework for energy efficient cloud offloading of smartphone applications", IEEE. Consum. Electron. **8**(1), 42 (2019)
13. Donohoo, B., Ohlsen, C., Pasricha, S., Anderson, C., Xiang, Y.: Context-aware energy enhancements for smart mobile devices. IEEE Trans. Mob. Comput. **13**(8), 1720–1732 (2014)
14. Donohoo, B., Ohlsen, C., Pasricha, S.: A middleware framework for application-aware and user-specific energy optimization in smart mobile devices. J Pervasive Mob. Comput. **20**, 47–63 (2015)
15. Donohoo, B., Ohlsen, C., Pasricha, S., Anderson, C.: Exploiting spatiotemporal and device contexts for energy-efficient mobile embedded systems. In: IEEE/ACM Design Automation Conference (DAC). IEEE (2012)
16. Tiku, S., Pasricha, S.: Energy-efficient and robust middleware prototyping for smart mobile computing. In: IEEE International Symposium on Rapid System Prototyping (RSP). IEEE (2017)
17. Pasricha, S., Doppa, J., Chakrabarty, K., Tiku, S., Dauwe, D., Jin, S., Pande, P.: Data analytics enables energy-efficiency and robustness: from mobile to manycores, datacenters, and networks. In: ACM/IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). IEEE (2017)

18. Pasricha, S., Raid Ayoub, M., Kishinevsky, S.K., Mandal, U.Y.O.: A survey on energy management for mobile and IoT devices. IEEE Des. Test. **37**(5), 7–24 (2020)
19. Kumar, K., Lu, Y.-H.: Cloud computing for mobile users: can offloading computation save energy? Computer. **43**(4), 51–56 (2010)
20. Flores, H., Hui, P., Tarkoma, S., Li, Y., Srirama, S., Buyya, R.: Mobile code offloading: from concept to practice and beyond. IEEE Commun. Mag. **53**(3), 80–88 (2015)
21. Li, J., Bu, K., Liu, X., Xiao, B.: Enda: embracing network inconsistency for dynamic application offloading in mobile cloud computing. In: Proceedings ACM SIGCOMM, pp. 39–44. ACM (2013)
22. Sivakumar, A., et al.: Cloud is not a silver bullet: a case study of cloud-based mobile browsing. In: Proceedings ACM Mobisys, p. 21. ACM (2014)
23. Monsoon Solutions Inc. official website. [Online]. Available: http://www.msoon.com/LabEquipment/Power-Monitor (2016). Accessed 9 Nov 2016
24. Matrix Calculator app at Google play store. [Online]. Available: https://play.google.com/store/apps/details?id=ru.alex-anderskokov.matrix&hl=en (2016). Accessed 7 Mar 2022
25. Amazon web services (AWS). [Online]. Available: https://aws.amazon.com/ (2016). Accessed 7 Mar 2022
26. Mozilla Firefox. [Online]. Available: https://www.moz-illa.org/en-US/firefox/android (2016). Accessed 7 Mar 2022
27. Puffin Web Browser. [Online]. Available: http://www.puffinbrowser.com/ (2016). Accessed 7 Mar 2022
28. Zipper Android App. [Online]. Available: https://play.google.com/store/apps/details?id=org.joa.zipperplus&hl=en (2016). Accessed 7 Mar 2022
29. Ezyzip: The simple online zip tool. [Online]. Available: http://www.ezyzip.com/ (2016). Accessed 7 Mar 2022
30. Google Translate Android App. [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.translate&hl=en (2016). Accessed 7 Mar 2022
31. Speak & Translate iOS app. [Online]. Available: https://itunes.apple.com/us/app/speak-translate-free-live/id804641004?mt=8 (2016). Accessed 7 Mar 2022
32. Fuld – Torrent Downloader app. [Online]. Available: https://play.google.com/store/apps/details?id=com.delphicoder.flud&hl=en (2016). Accessed 7 Mar 2022
33. Kelenyi, I., Nurminen, J.K.: Cloudtorrent-energy-efficient bittorrent content sharing for mobile devices via cloud services. In: Proceedings IEEE CCNC, pp. 1–2. IEEE (2010)
34. Altamimi, M., et al.: Energy cost models of smartphones for task offloading to the cloud. IEEE Trans. Emerg. Topics Comput. **3**(3), 384–398 (2015)
35. Li, H., Liu, D., Wang, D.: Integral reinforcement learning for linear continuous-time zero-sum games with completely unknown dynamics. IEEE Trans. Autom. Sci. Eng. **11**(3), 706–714 (2014)
36. Abouheaf, M., et al.: Multi-agent discrete-time graphical games and reinforcement learning solutions. Automatica. **50**(12), 3038–3053 (2014)
37. Anderson, C.: Introduction to machine learning. Website [Online]. Available: https://www.cs.colostate.edu/~anderson/cs545/index.html/doku.php (2016). Accessed 7 Mar 2022