

Investigation and RL-based Optimization of Offloading Decision Making in Industrial Edge Computing Scenarios

Eugen Volk ^{*}, Alexander Artemenko [†], Indrit Fejza [‡], Marco Aiello [§]

Abstract. Computation Offloading (CO) transfers computationally intensive tasks from hardware-limited devices (clients) to more powerful servers. This enables clients to overcome performance and power limitations and meet Quality of Service requirements. Research in CO focuses on “Why,” “What,” “If/When,” “Where,” and “How” to offload. This work investigates the “If/When” question from a performance perspective in dynamic environments. To optimize decision making, a Reinforcement Learning (RL) agent was designed, implemented, and trained to make efficient offloading decisions. The practicality of our solution was tested by emulating real hardware in industrial environments using 5G New Radio. Experimental results show that the proposed RL agent outperforms the existing closed-form solution from the literature, improves the total completion time of emulated apps, and reduces client device power consumption.

Keywords: Edge Computing, Computation Offloading, Task Complexity, Reinforcement Learning, Offloading Gain

1. Introduction

The usage of low power devices with limited hardware capabilities (CPU, GPU, RAM, storage) is swiftly increasing, especially in industrial environments [5, 20, 19, 29]. Mainly in the form of single-board computers [13, 28], these devices are often incorporated into other more complex units that are movable on the shop floor [24, 15].

^{*}Bosch Digital, Robert Bosch GmbH, Stuttgart 70469, Germany, eugen.volk@de.bosch.com, ORCID: 0009-0009-2574-4587

[†]Corporate Research and Advanced Engineering, Robert Bosch GmbH, Renningen 71272, Germany, alexander.artemenko@de.bosch.com, ORCID: 0009-0000-5607-3629

[‡]Database Cloud Infrastructure, SAP SE, Walldorf 69190, Germany, fejza.indrit@gmail.com

[§]Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart 70569, Germany, marco.aiello@iaas.uni-stuttgart.de, ORCID: 0000-0002-0764-2124

Hardware constraints limit the number of applications (App) that can be executed locally on these devices because the App's Quality of Service (QoS) standards will not be met. In order to achieve better performance, meet QoS requirements, and increase the overall Quality of Experience (QoE), the device can resort to Computation Offloading (CO) [14, 32, 18].

CO refers to the transfer of, possibly heavy, computational tasks from hardware limited devices (clients) to more powerful ones (servers) [14, 26]. The servers are connected to reliable power sources and have resources that are usually orders of magnitude better than those of the clients. The goal of CO is to ensure a better overall App performance and/or to reduce the power usage of the clients, in turn increasing their battery life time [20, 19, 18]. While there are different flavors of CO, like Fog/Edge Computing (EC) and Cloud Computing (CC), here we focus on EC, due to its closer proximity to clients and better latency behavior [11, 40]. Apps that use large payloads to be transmitted, like Augmented/Virtual Reality (AR/VR) and the ones which have demanding short latency requirements, like Automated Guided Vehicle (AGV) movement control, will not work as intended if the latency deadlines are not met. In EC, the latency is lower and more stable compared to CC, where the network distances among clients and servers can be substantial [11, 22]. Furthermore, the data are transmitted within a company's intranet and thus the connection is more secure.

Research in Computation Offloading (CO) aims to optimize the utility of resource-constrained mobile devices by addressing five fundamental questions [11, 36]. The motivation ("Why") focuses on overcoming device limitations to achieve specific objectives, such as minimizing response time for complex tasks and/or extending battery life by offloading when remote processing consumes less energy than local execution. To achieve this, systems must determine "What" to offload, either fully or partially, by partitioning application tasks into local and remote partitions, and decide "When" to offload by analyzing dynamic context — such as network bandwidth — to ensure the offloading is beneficial. Furthermore, the decision process involves selecting "Where" to offload and execute the task, choosing the appropriate target such as a nearby edge server or a powerful remote cloud, and establishing "How" to offload by scheduling operations over the most efficient communication channel [11, 36].

Answering all these questions is of critical importance, as CO might not always be beneficial. This is especially true in very dynamic environments, where system parameters change constantly [14]. Wrong decisions can lead to reduced App performance and increased power consumption. Making correct offloading decisions is based on many parameters to be considered. These include: the processing capability of the clients and servers, the available Bandwidth (BW) from client to server, the payload that needs to be transmitted, the complexity of the task to be computed, the type of underlying communication network, among others [11, 14, 36]. In multi-user scenarios, optimizing these set of parameters is similar to task scheduling [9], an NP-hard problem, because of the many dynamic effects caused by the wireless propagation, changing environmental conditions, movement of clients, constrained shared resources and multiple optimization objectives [19].

In the present work, we focus on computational offloading from the perspective

of a single client interacting with a single server. This simplification is intentionally made to isolate the offloading decision from the more complex resource allocation or server-side task scheduling problem. While a generic multi-objective CO problem is NP-hard [41], the presented single-client, single-server scenario with binary offloading is not. This is because the optimal decision can be determined in constant time by simply comparing the total cost for local execution (i.e., local processing time) versus the total cost for remote execution (which includes data transfer times and server processing time).

Whereas state-of-the-art approaches mainly consider individual offloading decision-making questions [36, 37], this work investigates the fundamental question of *when* and *if* an application should be fully offloaded within partially observable, stochastically dynamic environments. In particular, we focus on identifying the specific conditions under which Reinforcement Learning (RL) based computation offloading is successful in such stochastic settings. This problem is modeled as a binary offloading Markov Decision Process (MDP), where an RL-agent learns an optimal offloading policy by interacting with its dynamic environment to maximize cumulative rewards, detailed in Section 3.1.

In contrast to our earlier work [1], which evaluated an RL solution for a single AGV application, we now focus on a RL solution for a broad range of applications. In order to ensure repeatability and future extendability of the results, we use an existing emulator [23]. Applying this tool, we can emulate a broad range of Apps that address a wide range of task complexities to be performed and various payloads to be transmitted. Furthermore, a RL software agent is implemented to be used together with the emulator. To train and test the performance of this RL agent in various dynamic scenarios, we use a standard Raspberry Pi (RPi) model 3B as a client and a powerful laptop (Fujitsu CELSIUS H770) over a 5G communication network.

The novelty of this contribution lies in the combination of several aspects: (i) development of a lower-complexity RL-based CO decision-making approach, which is better suited for low-power devices, (ii) the integration of multiple optimization criteria, such as latency and energy consumption into CO decision-making process, (iii) its systematic evaluation in dynamic, stochastic, real-workload emulation environment, and (iv) its comparison to closed-form model-based approach.

The remainder of the paper is organized as follows. The current state-of-the-art about CO decision making is presented in Section 2. In Section 3, we formulate the binary CO problem as a Markov Decision Process and present our proposed solution: a RL agent that employs the Q-learning algorithm. Section 4 details the emulation environment used for our experiments. Subsequently, Section 5 presents the evaluation results of the implemented Q-learning agent and discusses the findings in detail. Finally, Section 6 provides concluding remarks and suggests directions for future work.

2. Related Work

Computation offloading is a cornerstone of modern Service Computing, yet the optimal strategy for making the offloading decision remains an open challenge, particularly in non-deterministic settings. To position our contribution, this section analyzes the state-of-the-art across four key dimensions: the problem’s formulation, its optimization objectives, the evaluation methodology, and the underlying decision models.

Problem Formulation. The literature bifurcates based on the complexity of the offloading problem. A significant body of work addresses multi-faceted scenarios, such as partial offloading where applications are partitioned [37], or multi-user environments where task scheduling becomes an NP-hard problem [9, 16]. Others tackle the intricate challenge of dependent task offloading [3]. While comprehensive, these models can obscure the dynamics of the core offloading trade-off. In contrast, our work intentionally focuses on the single-client, binary offloading problem. This simplification allows for a deep and focused analysis of the “If/When” decision under the unpredictable conditions of a stochastically dynamic environment, free from the confounding variables of system-wide resource contention.

Optimization Objectives. The goals driving offloading decisions also vary significantly. Much research pursues a single objective, such as minimizing latency, as demonstrated by the DRL technique in [7]. However, in practical industrial scenarios, objectives like performance and power consumption are often in direct conflict. This has spurred interest in multi-objective optimization, with approaches like the MORL framework in [39]. Our work adopts this holistic view but implements it through a pragmatic and tunable weighted-sum reward function. This provides a flexible mechanism to navigate the crucial trade-off between time and power savings, enabling the agent’s policy to be adapted to shifting operational priorities.

Evaluation Methodology. A critical gap exists between how most offloading strategies are validated and the environments in which they must operate. The overwhelming majority of studies rely on simulation, often assuming a deterministic or quasi-static environment [2, 3, 35, 17, 12, 16]. While this approach offers control, it often fails to capture the non-linearities of real-world hardware and wireless networks, a limitation our work directly addresses. A few studies have moved toward real-world validation [39, 38], but typically under constrained, non-dynamic conditions. A central contribution of our paper is bridging this gap by using an emulation-based approach—a step identified as essential by [25]. By grounding our agent’s training in measurements from real hardware and a live 5G network, we deliver a more credible assessment of performance in a truly dynamic environment.

Decision Models. The intelligence of an offloading strategy resides in its decision model. Many models operate on a limited set of parameters; the DQN in [25], for instance, overlooks application-specific characteristics. Our model incorporates a richer, more context-aware state space that includes fine-grained task complexity and data payload size. Regarding the algorithm, while Deep Q-Networks (DQN) are prevalent [12, 42, 41, 10], we contend that their computational overhead and slower adaptation make them less suitable for our target scenarios. We therefore make a deliberate choice for traditional Q-learning, prioritizing the lightweight footprint, im-

plementation simplicity, and rapid environmental adaptation required for practical deployment on resource-constrained industrial devices.

Summary and Contributions. Our review reveals a clear need for research that moves beyond idealized simulations and deterministic models. This paper makes a direct contribution by:

- Analyzing the binary offloading decision within a stochastically dynamic environment, which is far more representative of industrial conditions than the static models assumed by closed-form solutions.
- Employing a high-fidelity, emulation-based methodology using real hardware for credible and relevant results.
- Justifying and implementing a lightweight Q-learning approach tailored for the practical constraints of low-power devices, while providing a flexible framework for multi-objective optimization.

By taking this focused, emulation-grounded approach, we provide robust, actionable insights into the performance of RL-based offloading in realistic industrial scenarios.

Having established the limitations of existing deterministic models and simulation-based methodologies, we now turn to formally defining the problem of offloading in a stochastically dynamic environment. In the next section, we formulate this challenge as a Markov Decision Process (MDP) and detail our proposed Q-learning agent designed to solve it.

3. Problem Formulation and Proposed Solution

In an ideal, fully observable, and deterministic environment, the computation of offloading decision can be modeled as a closed-form optimization problem, as described in [36]. According to this model, offloading is beneficial from a latency perspective if the time for local execution exceeds the combined time for remote execution and data transmission. This condition is formally expressed in Equation (1):

$$\frac{C_{task}}{IPS_{client}} > \frac{1}{F} \times \frac{C_{task}}{IPS_{client}} + \frac{D_{task}}{B_{network}}, \quad (1)$$

where C_{task} is the task complexity (in instructions), IPS_{client} is the client's processing speed (in Instructions Per Second), F is the server-to-client processing speed ratio, D_{task} is the data payload size (in bits), and $B_{network}$ is the network bandwidth (in bits per second).

Similarly, with respect to power consumption, offloading is advantageous if the energy consumed during local computation is greater than the energy consumed for data transmission and the device's idle state during remote execution. This is captured in Equation (2):

$$p_m \times \frac{C_{task}}{IPS_{client}} > p_i \times \frac{1}{F} \times \frac{C_{task}}{IPS_{client}} + p_{tr} \times \frac{D_{task}}{B_{network}}, \quad (2)$$

where p_m is the power of device during local computation, p_i is the idle power consumption of the client device and the p_{tr} is the power consumed during transmission of the task. For simplicity, this model only considers the uplink transmission from the client to the server.

Many contemporary industrial systems, including those at Bosch, often rely on such deterministic, closed-form approaches for decision-making. Theoretically, this approach is sufficient, provided the environmental parameters can be sampled and processed—fully addressing the computation offloading life-cycle—at a frequency satisfying the Nyquist criterion, i.e., at least twice the rate of their change [30]. Under this condition, the system can be treated as quasi-static, and the model holds. However, real-world industrial environments, particularly those with mobile clients like Automated Guided Vehicles (AGVs), are rarely this predictable. The wireless network conditions for a device moving across a factory floor are subject to dynamic and stochastic fluctuations from signal interference, obstructions, and mobility, making it impractical to satisfy the Nyquist rate. This unpredictable behavior invalidates the core assumptions of closed-form solutions, leading to suboptimal or incorrect offloading decisions.

To address the challenge of operating in such stochastically dynamic environments, we propose a solution based on Reinforcement Learning (RL). While recent surveys identify Deep Q-Networks (DQN) as a prevalent technique for decision-making in similar offloading scenarios [41, 10], we selected traditional Q-learning due to several distinct advantages for our target industrial scenarios:

1. **Faster Environmental Adaptation and Scalability:** A tabular Q-learning agent adapts to environmental changes dynamically without needing the complete retraining of a neural network. In contrast, a DQN's training functions and neural architecture often require significant adaptation for different environments, which poses a hurdle to the scalability and rapid deployment of the approach.
2. **Lower Computational Overhead and Data Requirements:** Q-learning is a simpler and less resource-intensive approach, making it a more viable option for real-time deployment on low-power client devices. DQN requires a larger dataset to be collected in advance to train its neural network, adding significant initial overhead that is not necessary for our model.
3. **Simpler Implementation:** The relative simplicity of the Q-learning algorithm facilitates more straightforward implementation, debugging, and validation compared to the complexities of a DQN architecture.

Our method relies on a pre-recorded database of measurements, meaning it is most effective when the payload and complexity of applications are known and fall within the range of previously trained entries. This makes Q-learning a better fit for our scenario, as it is a simpler tool that is highly effective for a well-defined state space without the unnecessary overhead of a neural network.

We acknowledge that for scenarios with extremely high-dimensional state spaces, DQN can offer superior performance and scalability, a topic we plan to investigate

in future work. The subsequent subsections are structured as follows: Section 3.1 formulates CO problem as a Markov Decision Process (MDP), and Section 3.2 details the proposed Q-learning agent.

3.1. Problem Formulation

The computation offloading decision process can be formulated as an finite state Markov Decision Process (MDP), which is defined as a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ symbolizes the action space, $\mathcal{P}$ symbolizes state transition probability function, defining probability of transitioning to state s' from state s by taking action a , and $\mathcal{R}$ depicts the immediate reward. The detailed information is provided below.

1. **State Space:** Based on analysis of the closed form solution equation 1 we know that application parameters like task complexity (C_{task} aka c), data transfer volume (D_{task} aka payload p) as well as a system parameter bandwidth (B_{network} aka b) are essential for any offloading decision, in addition to other parameters such as IPS_{client} and F , which we will consider as rather fixed. Therefore, state space comprises of these parameters. In each decision epoch t , every state of the emulator can be characterized by $S_t = (c_t, p_t, b_t)$. A more detailed description of the state space is provided in section 4.2.1.
2. **Action Space:** At the t -th time slot, the binary offloading action is denoted as $a(t) \in A$, where A is the action space. Accordingly, the available actions for the t -th time slot are given as, $A = [\neg \text{offload}, \text{offload}]$, which represents not offloading and offloading actions.
3. **State Transition Probability:** The state transition probability models the dynamics of the system or environment by quantifying the probability that taking action a in state s will lead to the next state s' .

$$P(s' | s, a) \quad (3)$$

This is important aspect of the stochastic, non-deterministic 5G real-world environment, where a bandwidth may drop at any time, as a result of the client-device movement or interferences caused by dynamic effects of the wireless propagation. To emulate such a dynamic behavior of the environment and investigate performance of the agent, we introduce dynamic bandwidth drops, occurring with various probabilities. Section 5.1 describes parameters of evaluation scenarios with dynamic bandwidth drop probabilities, ranging from 10% to 100%.

4. **Reward Function:** As noted, one of the objectives of our contribution is usage and integration of multiple optimization criteria, such as latency and energy consumption, into the CO decision-making process. The reward function is designed to reflect the gains in these objectives. Therefore, we use the reward

functions defined in (5) for time-saving rewards, denoted as $\mathcal{R}_{time}(s, a)$, and (6) for power-saving rewards, denoted as $\mathcal{R}_{power}(s, a)$. The time and power gains are calculated as the absolute difference between local ($local_{time}$) and remote ($remote_{time}$) processing times and power consumption, respectively, as shown in (4). These gain values are then used in (5) and (6) to compute the reward returned to the agent based on its chosen action.

$$\begin{aligned} gain_{time} &= |local_{time} - remote_{time}| \\ gain_{pwr} &= |local_{pwr} - remote_{pwr}| \end{aligned} \quad (4)$$

$$\mathcal{R}_{time}(s, a) = \begin{cases} +gain_{time}, & \text{if } local_{time} > remote_{time} \text{ \& } a = \text{offload} \\ -gain_{time}, & \text{if } local_{time} < remote_{time} \text{ \& } a = \text{offload} \\ +gain_{time}, & \text{if } local_{time} < remote_{time} \text{ \& } a = \neg \text{offload} \\ -gain_{time}, & \text{if } local_{time} > remote_{time} \text{ \& } a = \neg \text{offload} \end{cases} \quad (5)$$

$$\mathcal{R}_{power}(s, a) = \begin{cases} +gain_{pwr}, & \text{if } local_{pwr} > remote_{pwr} \text{ \& } a = \text{offload} \\ -gain_{pwr}, & \text{if } local_{pwr} < remote_{pwr} \text{ \& } a = \text{offload} \\ +gain_{pwr}, & \text{if } local_{pwr} < remote_{pwr} \text{ \& } a = \neg \text{offload} \\ -gain_{pwr}, & \text{if } local_{pwr} > remote_{pwr} \text{ \& } a = \neg \text{offload} \end{cases} \quad (6)$$

Combining both time $\mathcal{R}_{time}(s, a)$ and power $\mathcal{R}_{power}(s, a)$ rewards into a single formula, we arrive at (7). The α parameter is used to weight between the time and/or power rewards.

$$\mathcal{R}_{combined}(s, a) = \alpha \times \mathcal{R}_{time}(s, a) + (1 - \alpha) * \mathcal{R}_{power}(s, a) \quad (7)$$

5. **Policy Function:** A policy π is a function which allows agent to choose actions in given states and can be described as a distribution over actions in given states [33].

$$\pi(a | s) = P[A_t = a | S_t = s] \quad (8)$$

The goal of MDP is to find the optimal policy $\pi^*(a | s) = P[A_t = a | S_t = s]$ that maximizes the total expected discounted cumulative rewards over time [31, 33].

In the next section we describe a model-free Q-Learning algorithm that learns optimal policy for computation offloading.

3.2. Q-learning Method

Q-learning is a model-free Reinforcement Learning (RL) algorithm designed to find the optimal action-value function, denoted as Q^* . It achieves this by iteratively updating

a lookup table (the Q -table), which stores the expected utility for every state-action pair (s, a) within a given Markov Decision Process (MDP). As a model-free approach, Q-learning does not require an explicit model of the environment's dynamics, such as transition probabilities, learning instead through trial and error [31].

At each time step t , the agent, being in state s_t , must select an action a_t . This decision is governed by a policy that balances the trade-off between exploration (sampling new actions) and exploitation (leveraging existing knowledge). We employ the widely-used ϵ -greedy strategy [31] for this purpose. With a probability of ϵ , the agent performs an exploratory action by making a random offloading decision. Conversely, with a probability of $1 - \epsilon$, the agent exploits its learned knowledge by selecting the action a_t that maximizes the Q -value for the current state s_t .

To enhance convergence, we implement a decaying ϵ -greedy strategy. The exploration rate ϵ is initialized to a maximum value, $\epsilon_{\max}$, and is exponentially decayed with each training episode at a rate of ϵ_{decay} , eventually reaching a minimum value, $\epsilon_{\min}$ (see Table 1).

Following the execution of action a_t , the emulator environment provides a reward $R(s_t, a_t)$ and transitions to the next state, s_{t+1} . This information is used to update the corresponding entry in the Q -table according to the Bellman equation. Through iterative updates, the Q-learning algorithm converges towards the optimal policy.

The primary hyperparameters for our Q-learning model are summarized in Table 1. The training and testing procedures are detailed in Section 4.2.2 and Section 4.2.3, respectively.

Table 1. A summary of the parameters of the Q-Learning model

Parameter	Value
State Space	[Complexity, Payload, Bandwidth]
Action Space	[Offload, Not-Offload]
Number of Epochs	100
Learning Rate	0.1
Discount Factor	0.95
ϵ (max, min, decay)	(1.0, 0.01, 0.03)
Reward	[Time Gain, Power Gain, Time & Power gain]

4. Emulation Environment

For a realistic and meaningful evaluation, we begin with a setup based on a powerful laptop, a Fujitsu CELSIUS H770 as the server, and a standard Raspberry Pi (RPI) as the client device.

4.1. Setup

The setup consists of a single user scenario. A client device, an RPi, is connected via LAN (with max data rates of 100 Mbps) to a 5G infrastructure. A laptop is connected via a 5G module (with max Downlink data rates of 900 Mbps) to the same infrastructure. The laptop acts as the server, to which the computation is to be offloaded.

Table 2 provides an overview of the hardware parameters used for the client and server devices. One notices that, in terms of processing speed measured in Dhrystone Million Instructions per second (DMIPS), the server has close to 10 times more powerful CPU than the RPi 3B. This parameter is very important in CO, as it is used for offloading decision making.

Table 2. A summary of the client & server device parameters

Device Type	Client Device	Server Device
Device Model	RPi 3B	Fujitsu CELSIUS H770
SoC Model	BCM2837	n.a.
CPU Model	ARM Cortex-A53	Intel Xeon E3-1505M v6
CPU Frequency	1.2 GHz	3.0 GHz, up to 4.0 GHz
CPU DMIPS	2414 DMIPS	23449 DMIPS
LAN	LAN9514	Intel Ethernet I219-LM
WiFi	BCM43438	Intel Dualband-AC 8265
WiFi Connection	802.11 b/g/n (WiFi 4)	802.11 b/g/n/ac (WiFi 5)
Operating System	Raspbian Buster Lite	Ubuntu 20.04 LTS
Memory	1 GB LPDDR2	64 GB DDR4-2133

For the 5G network infrastructure we used 5G-NR Release 15 SA, whereas channel bandwidth was 100 MHz. The programming of the emulation and Apps running on the nodes is implemented using Python.

Having shown all the details of the used devices and 5G network, we go through the whole workflow design of our emulation process, presented in the next subsection.

4.2. Emulation Process

The emulation process is split into three distinct phases: (i) Measurement & Recording, (ii) Training, and (iii) Testing. We describe each step below.

4.2.1. Measurement & Recording

The goal of the Measurement & Recording phase is to build a large database with timing measurements. Based on previous experience [23], we choose to enlarge the evaluation and emulate a multitude of Apps that can be offloaded. These Apps have

a variety complexity in terms of the tasks to be computed, and also in terms of the transmitted payload size. As shown in Figure 1, a recorded database matrix consists of 1,120 App variations, ranging from 0 to 5 MB in 20 steps for the payload, and from 0 to 1,185 Million Instructions in 56 steps for the complexity of the task. The payload

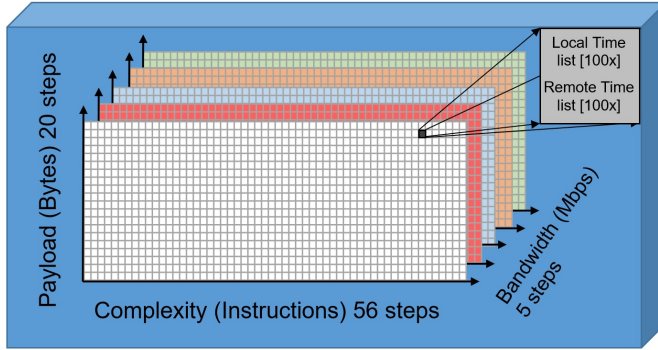


Figure 1. Applications matrix

of the task is emulated as a single byte-string to be offloaded and the complexity of the task is being emulated as a number of square root (SQRT) math operations that need to be computed. For all Apps in the matrix, we measure the time to compute the task, the number of SQRT operations performed locally, and the time to compute the same task remotely. The timing of remote computation starts before the first byte of the payload is transmitted and it ends when the feedback is received from the server. In order to have a baseline comparison between the performance of the server's and client's CPUs, we have used the Dhrystone benchmark [34]. It is calculated that a single SQRT operation is equivalent to 2,370 Dhrystone instructions [23]. Having this information, we can switch seamlessly between the number of SQRT operations and the number of instructions a task is comprised of. The performance of the CPUs of the client and server devices are shown in Dhrystone Million Instructions per second (DMIPS) in Table 2.

For the 5G network, we have considered five possible intervals for the client's transmission bandwidth (Full BW/100 Mbps, 70 Mbps, 50 Mbps, 30 Mbps, and 1 Mbps). The BW has been limited from the client's side using the Wondershaper tool [21]. For each network case and BW, we have made one hundred measurements of the time of completion for "Local Case" (no offloading) and "Remote Case" (offloading). All the time recordings are saved in binary files in a form of a dictionary, where the App's parameters (complexity and payload size) and network BW are the keys, and the two lists of the aforementioned timing measurements are the values for the local and remote processing time.

For the estimation of the power-consumption of the client device, we use typical bare-board current consumption for the RPi 3B: 400 mA in idle state, and up to 1.34 A during heavy computations, based on [27]. After creating this database of

measurements, we split it into training and testing datasets used to train and test the RL agent.

4.2.2. Training

To train the RL agent, we adopted Q-Learning, as described in section 3.2. The training process consists of several consecutive steps. After setting the parameters of the App payload, complexity and initial maximum BW, the offloading agent makes an offloading decision either at random or on the basis of the Q-values that exist in the Q-table. The random offloading decision is made with ϵ probability (Table 1), starting with $\epsilon_{\max}$ and exponentially decaying with each episode by the ϵ_{decay} rate, based on the ϵ -greedy algorithm [31]. This allows the agent to explore the space and learn. After the offloading decision has been made, the BW of the network is artificially changed, dropping to a new value, and the local and remote processing times are calculated using this new BW. This drop occurs with a preset probability to a new BW value. The new BW values are either 70, 50, 30, or 1 Mbps and the drop probability can occur with 100, 70, 50, 30, 10 %. In this way, we emulate various levels of system dynamics.

After the BW changes, a remote and local time are randomly selected from the measurement records in data-dictionary (for specific payload, complexity and new BW value), created during “Recording phase.” Taking into account these two timing values, as well as the offloading decision that was made by the agent, the gain can now be calculated. The gain can be positive or negative and it is used to update the Q-values of the Q-table. After the Q-table has been updated, the new state is set, and the whole procedure runs again cyclically: for each App one hundred times, in total 112,000 times (1120 Apps x 100 runs).

As noted, for Rewards, we use equation (5) for time saving rewards $R_{\text{time}}(s, a)$, and equation (6) for power saving rewards $R_{\text{power}}(s, a)$. For combination $R_{\text{combined}}(s, a)$ of both time and power rewards we use equation (7).

4.2.3. Testing

The process flow for the testing phase is similar to the one described in the training phase. However, the offloading decision process is based only on the Q-values in the Q-table, without the random choice. Correspondingly, the agent selects from the Q-table for a given state an action (to offload or not) with the highest Q-value.

5. Evaluation

The evaluation is based on the training and testing phases by analyzing log files. In particular, we compare the results of our proposed RL-based agent with the closed-form formula approach as found in the literature [36].

As noted, according to [36], offloading is beneficial in regard to time savings if equation (1) is satisfied. With regard to power consumption savings, offloading is beneficial if equation (2) is satisfied.

The comparison between closed-form approach and RL approach is based on a dynamic scenario, with varying probabilities for bandwidth drop after an offloading decision has been made. The comparative plots are organized in the following way:

- The plots have two columns, showing the closed-form solution (left), also denoted as theoretical decision making, and our proposed RL model (right), and five rows representing bandwidth drop probabilities.
- Blue lines represent the closed-form decision-making boundary condition.
- Blue crosses represent an offloading decision mistake (false positive: App task should not have been offloaded).
- Black dots represent an offloading decision mistake (false negative: App task should have been offloaded).
- Correct Decisions in %, also denoted as accuracy, represents how many correct offloading decisions were made in total.
- Total Local Time (TLT) is the total sum completion-time of all Apps when no offloading takes place.
- Total Time is the total completion-time of all Apps, whether offloaded or not, based on the agent's decision, expressed as % of TLT.
- Total Local Power (TLP) is the total power consumption of the client device when no offloading takes place.
- Total Power is the total power consumption of the client device based on the agent's offloading decision, expressed as % of TLP.
- Offloading gain is the ratio between local to offloaded completion time (local/offloaded).

The blue line in each subplot represents the boundary condition derived from the closed-form offloading-decision formula (1), detailed in [36]. Every App that falls underneath this line should be offloaded, because the gain is bigger than 1 and the App completion time will be reduced. If the App falls above this line then local computation should take place, because offloading will result in a loss, i.e., increase in completion time. Apps that fall close to this line, have a gain close to 1, so remote and local computation take approximately the same time. "Correct Decisions" is a simple parameter that shows how many correct decisions were made in percentage. An offloading decision is correct in these two cases:

- Task offloaded, when local time $>$ remote time;
- Task not-offloaded, when local time $<$ remote time.

This parameter however does not show the entire picture, because the net value of the mistake differs from what type it is. For example, the wrong decision to offload a task, that results in a loss of time, may not have the same net loss compared to when the wrong decision to make local computation is made. It can happen that even though the number of correct decisions is low, the time saved is high. So this parameter, also used in [23], is not effective on its own and it should be used in conjunction with other indicators in order to make meaningful comparisons between different decision making alternatives.

“Total Time” is the sum of completion time of all Apps, offloaded or not, based on the decision-making agent. So both the remote processing time and the local processing time contribute to this overall sum. This value is a direct comparison parameter that is used to show the benefits of using CO. In some cases, the value can be more than 100 % of TLT. When this happens, it means that a better strategy is to not offload any task. If this parameter is below 100 %, the decision-making agent has made correct offloading decisions, and overall it has saved time. We have to mention here again, that this value is not strictly correlated to the “Correct Decisions” parameter. It might very well happen that the “Correct Decisions %” is high, but that does not immensely affect the “Total Time.”

5.1. Time Savings Analysis

The dynamic case, for which the bandwidth changes over time after the decision to offload or not was taken, is analyzed next. Table 3 provides an overview of the emulation parameters used for such dynamic cases.

Table 3. Emulation parameters

Parameters	Values
Complexity range	0-500000 (sqrt operations)
Complexity step	9000 (sqrt operations)
Payload range	1-5000000 (bytes)
Payload step	256000 (bytes)
Network	5G (BW=100 Mbps)
New BW	[70, 50, 30, 1] Mbps
BW drop probability	[100, 70, 50, 30, 10] %

The BW drop is used to simulate the effect of the mobility of the client device and other possible network congestions. While one might expect the theoretical decision-making formula to fail to make correct decisions in these dynamic systems, actually, the RL-based agent learns the BW changing behavior and it adjusts its decision policy based on the learned experience.

The first case is when the BW drops to the new value of 70 Mbps, corresponding to 70 % of the maximum (100 Mbps); as presented in Figure 2. Both the theoretical and RL-based decision-making agents perform better than the total local computa-

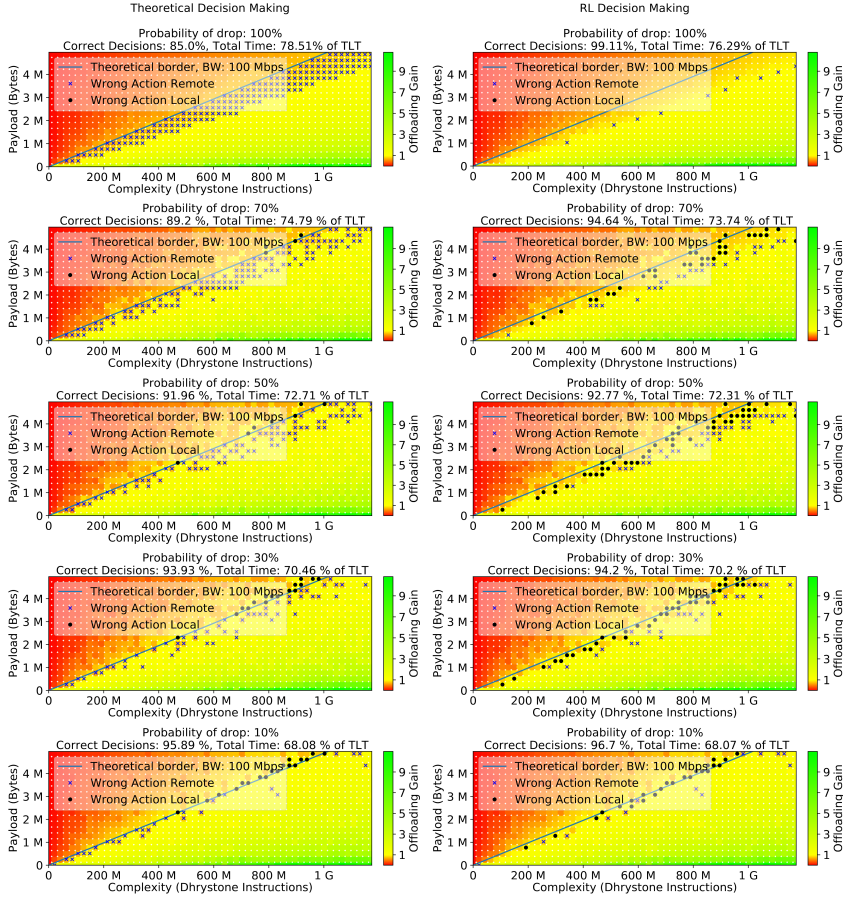


Figure 2. Offloading results: dynamic case, bandwidth drop to 70 Mbps

tion scheme, achieving a lower total completion time than TLT. We also observe that in all BW drop probability cases, in Figure 2, the number of correct offloading decisions changes differently for the two algorithms. For the theoretical decision-making scheme, the number of correct decisions is increasing with the decrease of the probability of BW drop. This is expected, because the lower the percentage of BW drop, the lower the number of mistakes the algorithm makes. This is also represented by the reduction of the total completion time of all Apps. On the contrary, on the RL-based side (right columns), the percentage of correct decisions is decreasing up to the minimum value of 92.77 %, which occurs at a 50 % probability of BW drop. After this point, when the BW is dropping with probabilities of 30 % and 10 %, the percentage of correct offloading decisions increases again. This behavior is due to the fact that, for the RL agent, the 50 % drop is the most non-deterministic system behavior. If the probability of BW change is lower or higher than 50 % then the decision is more

straightforward for the agent.

The same behavior is examined also for the bandwidth drop to 50 Mbps case, presented in Figure 3. Again, we remind that the number of correct decisions “Decision Accuracy” is not closely related to the “Total Time,” as both the algorithms achieve a lower sum completion time, with the smaller BW drop probabilities, regardless of the number of correct offloading decisions.

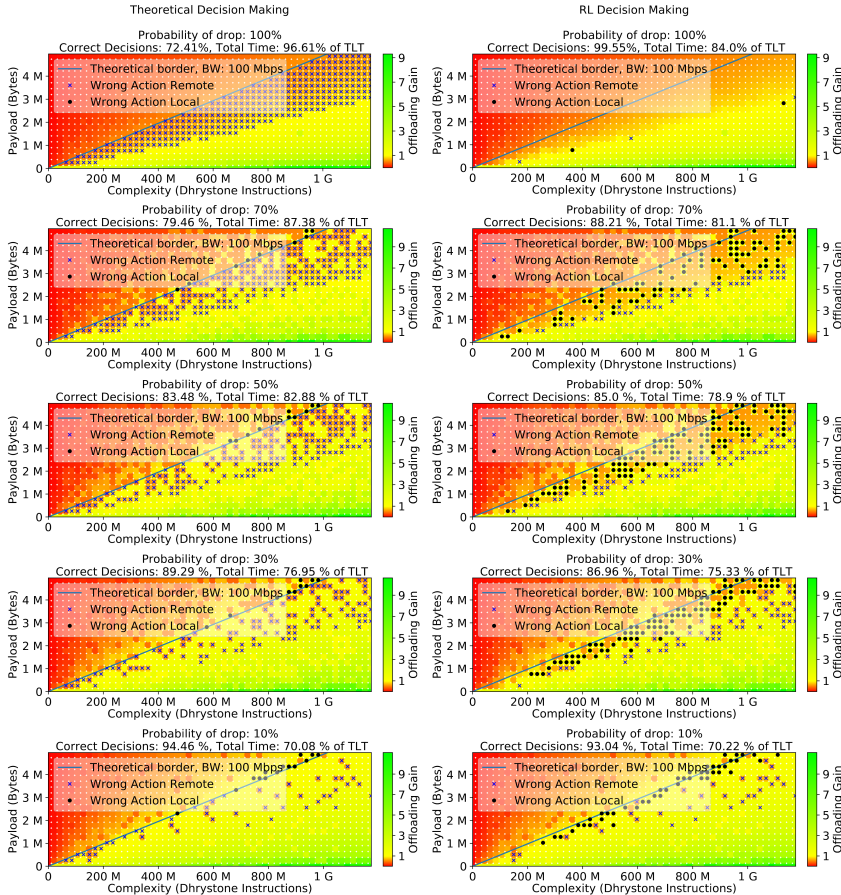


Figure 3. Offloading results: dynamic case, bandwidth drop to 50 Mbps

Figure 3 shows the effects of the policy of the RL agent for offloading decisions. The algorithm determines when to offload Apps that fall close to the new inferred decision boundary and to not offload apps that are close to the original decision line. This shaping of offloading mistakes makes it possible for the agent to achieve better overall results than the closed-form solution.

Figure 4 contains the comparison of the “Decision Accuracy” (left) and “Total Time” (right) metrics for the theoretical and RL-based agents across all previously

mentioned bandwidth drops and extended BW values (1, 30, 50, 70 Mbps). To improve readability, the y-axis (Time % of TLT) in the “Total Time” plot is limited to a maximum of 110 %.

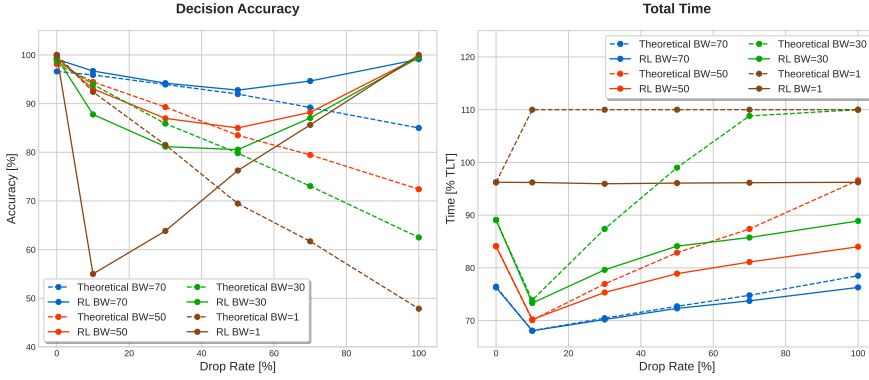


Figure 4. Offloading results: performance graph on accuracy and total time

Figure 4 confirms our previous observations for dynamic scenarios (starting at 10 % drop probability). For the theoretical agent (dashed lines) and all BW drop values, accuracy decreases linearly with increasing bandwidth drop probability, a consequence of the linear correlation between drop rate and decision errors. This increased error rate also correlates with the higher total completion time.

For the RL-based agent (solid lines) and BW drops to 30, 50 and 70 Mbps, accuracy is lowest at a 50 % drop probability, reflecting the system’s most non-deterministic state, and improves as the probability deviates from 50 %. The total completion time is lowest at 10 % drop rate, and increases linearly with the drop rate for all the BW values. This confirms also our earlier statement, that the accuracy is not closely related to total completion time.

For the extreme case (BW drop to 1 Mbps), the RL agent has lowest accuracy at 10 % drop rate, choosing conservatively to make local computation for the majority of the Apps. This is also reflected in the almost constant total completion time (ca. 96 % of TLT) for all drop rates. The reason for this behavior lies in the cumulative reward and penalty strategy of the RL agent, where few false positive offloading decisions lead to high penalty values due to high transfer costs, resulting in a very low total net value.

Figure 4 shows that in all BW change cases, the RL approach is achieving a better total completion time than TLT and theoretical closed-form solution. The theoretical closed-form solution is performing for 30 Mbps (>50 % drop rate) and 1 Mbps (>10 % drop rate) cases even worse than the total local processing scheme, achieving a “Total Time” higher than 100 % of TLT.

In summary, we observe that CO in conjunction with an RL approach performs better in a dynamic environment when compared to the closed-form solution. The theoretical formula approach to the “When” to offload is a viable solution, when the

system is mostly stable and the parameters are not changing drastically. For any type of dynamic system, an RL-based agent is strongly advised to be used.

5.2. Power Saving Analysis

After studying the time-savings during the training of our RL agent, we move to power consumption considerations. This entails the modification of the α variable in equation (7). By changing the α parameter, one shifts the weight to optimize time ($\alpha = 1$) or the power consumption ($\alpha=0$), respectively.

For calculation of the power consumption, we assume that the RPi consumes more power during the transmission of the task, compared to local computation. Based on the literature, we determine a realistic value of 15% more power during transmission [4, 6, 8]. However, while the device is waiting for the response from the server, it can make other computations in the background. Thus, in the present analysis, the power consumption of the device is always higher in the case of CO.

Figure 5 shows the comparison between having an α set to 1 (orange dashed line) or to 0 (blue line), for the static case, in which the device is set to consume 15 % more power during the transmission than otherwise. We can see that when α is 0, the total power consumption of the device is slightly lower than on the case when α is 1. This is to be expected because saving time can lead to higher power consumption and vice versa.

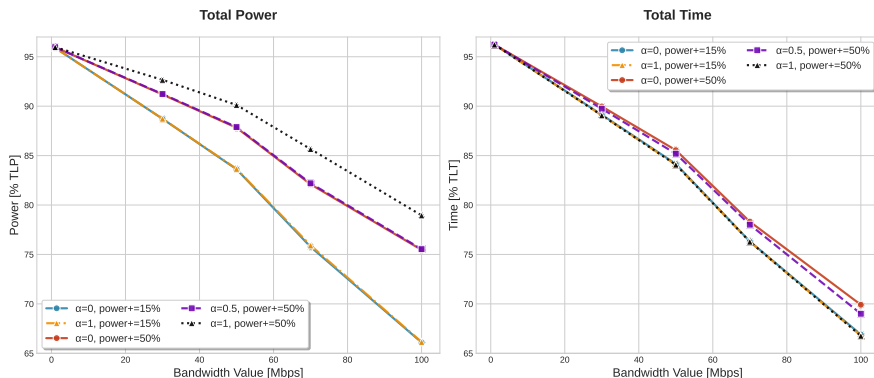


Figure 5. Comparison of power savings in RL agents with varying α & power settings

If we use a much larger value for the power consumption during offloading, 50%, the differences between the saving time and saving energy are more pronounced. This case is also shown in Figure 5. When α is 0 (red line), the RL agent is making more local computation decisions for Apps that are close to the theoretical decision line. For these Apps, the local computation and remote computation times are very similar. However, because the device consumes more power for transmission than otherwise, the agent learned not to offload them. In Figure 5, the black dotted line shows results

of offloading in the case of α being set to 1, aiming at minimizing total time. We see that “Total Time” (right column) for $\alpha=1$ is lower than for $\alpha=0$, for all BW values. This is the expected behavior of the agent, according to set goal to minimize total time. At the same time, we observe “Total Power” for $\alpha=1$ is higher than for $\alpha=0$, a result of increased power consumption during the preferred offloading strategy of the agent to minimize total time. The violet dashed line shows the results of offloading in the case of α being set to 0.5. This value (0.5) means that we want to minimize both the power consumption as well as the App completion time, in equal weights. In the right column of Figure 5, we observe that the “Total Time” parameter for $\alpha = 0.5$ (violet dashed line) is slightly lower for all BW values, compared to the corresponding cases for $\alpha = 0$ (red line), and higher than for $\alpha = 1$ (black dotted line). On the contrary, the “Total Power” parameter is slightly higher when $\alpha=0.5$ (violet dashed line) than when $\alpha = 0$ (red line), but lower than when $\alpha = 1$ (black dotted line). In conclusion, when $\alpha=0.5$, the RL-based agent learns to make offloading decisions not only to save power but also to lower the App completion time.

In summary, the agent learns how to make correct offloading decisions in order to minimize either the power consumption of the client device, the sum completion time of the Apps, or both, based the parameter α . We have shown also that the RL model can be adapted to optimize not only the completion time of Apps but also the power consumption of the client device. While here we made the assumption that the power consumed is higher during CO than during local computation, this is not entirely true and further research needs to be made.

6. Conclusion and Outlook

The present work about computation offloading in industrial edge computing scenarios addresses the challenge of making effective offloading decisions to maximize overall system’s performance. In general, offloading computation tasks from resource-constrained devices to edge servers regardless the system state does not guarantee improved performance, reduced computation time, or lower power consumption. The main goal has been to evaluate the effect of the decision choices and the possibility to learn from observed network behavior in actual scenarios with real hardware. The core contribution lays in implementing a RL-based model to dynamically determine whether to offload tasks, adapting to varying application task complexities, payload sizes, and network conditions within a dynamic actual 5G environment.

The results show that the RL agent outperforms the closed-form solution in dynamic CO scenarios by learning network dynamics and optimizing offloading decisions to maximize overall gain. While both the RL agent and “Theoretical Decision Making” approach exhibit errors near the theoretical border, i.e., where CO gain is close to 1, due to minor variations in system utilization and network behavior, the RL-based agent demonstrated more effective error handling behavior. Specifically, while occasionally making more incorrect decisions than the closed-form solution, the RL agent’s decisions resulted in a smaller overall time penalty, minimizing the total completion time for all applications. Furthermore, the RL-based approach consistently

achieved lower sum completion time (“Total Time”) and reduced power consumption compared to total local processing scheme, a feat not consistently achieved by the closed-form solution. This work provides compelling evidence that an RL-based approach is beneficial for CO decision making, enabling adaptation to changing conditions and optimization of key performance indicators like power consumption and application performance.

This is an initial step toward fully dynamic CO emulation in industrial environments. Future work will focus on addressing current limitations and expanding the scope of the investigation. A key area for improvement is incorporating even more realistic network dynamics, capturing the continuous fluctuations in bandwidth experienced in real-world deployments. Testing the model in a real industrial setting such as AGV, accounting for client device mobility and implementing real-time training and testing, is a crucial next step. Furthermore, we propose integrating additional parameters into the offloading decision-making process, such as client and server system load, actual client device power consumption, and location data. Finally, exploring alternative ML algorithms, such as DRL, could address the limitations of the Q-Learning approach. DRL’s ability to handle continuous state spaces and complex scenarios makes it a promising avenue for further enhancing CO performance and adaptability in dynamic industrial environments.

Acknowledgment

We acknowledge the use of Large Language Models – GPT-5o Mini and Gemini 2.5 Pro – for editing and grammar enhancement in selected sections. The authors alone are responsible for the content of the paper.

References

- [1] Artemenko A., Swamy K., Volk E., Jeroschewski S., and Dommel J. Validation and evaluation of computation offloading in campus networks using a real 5g setup. In *2024 IEEE 99th Vehicular Technology Conference (VTC2024-Spring)*, pages 1–5, 06 2024.
- [2] Chu S., Gao C., Xu M., Ye K., Xiao Z., and Xu C. Efficient Multi-Task Computation Offloading Game for Mobile Edge Computing . *IEEE Transactions on Services Computing*, 17(01):30–46, 2024.
- [3] Dai X., Xiao Z., Jiang H., Lei M., Min G., Liu J., and Dustdar S. Offloading dependent tasks in edge computing with unknown system-side information. *IEEE Transactions on Services Computing*, 16(6):4345–4359, 2023.
- [4] Eames A. Power consumption benchmarks. <https://raspi.tv/2019/how-much-power-does-the-pi4b-use-power-measurements>. Accessed: 2025-05-08.

-
- [5] Galkin P., Golovkina L., and Klyuchnyk I. Analysis of single-board computers for iot and iiot solutions in embedded control systems. In *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*, pages 297–302. IEEE, 2018.
 - [6] Geerling J. Jeff geerling raspberry pi power consumption measurements. <https://www.pidramble.com/wiki/benchmarks/power-consumption>. Accessed: 2025-05-08.
 - [7] Goudarzi M., Rodriguez M. A., Sarvi M., and Buyya R. μ -ddrl: A qos-aware distributed deep reinforcement learning technique for service offloading in fog computing environments. *IEEE Transactions on Services Computing*, 17(01):47–59, 2024.
 - [8] Hawkins M. Matt raspberry pi power consumption measurements. <https://www.raspberrypi-spy.co.uk/2018/11/raspberry-pi-power-consumption-data/>. Accessed: 2025-05-08.
 - [9] He H., Xu G., Pang S., and Zhao Z. Amts: Adaptive multi-objective task scheduling strategy in cloud computing. *China Communications*, 13(4):162 – 171, 2016.
 - [10] Hortelano D., de Miguel I., Barroso R. J. D., Aguado J. C., Merayo N., Ruiz L., Asensio A., Masip-Bruin X., Fernández P., Lorenzo R. M., and Abril E. J. A comprehensive survey on reinforcement-learning-based computation offloading techniques in edge computing systems. *Journal of Network and Computer Applications*, 216:103669, 2023.
 - [11] Hu Y. C., Patel M., Sabella D., Sprecher N., and Young V. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.
 - [12] Huang L., Feng X., Zhang C., Qian L., and Wu Y. Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. *Digital Communications and Networks*, 5(1):10–17, 2019.
 - [13] Isikdag U. Internet of things: Single-board computers. In *Enhanced Building Information Models*, pages 43–53. Springer, 2015.
 - [14] Kumar K., Liu J., Lu Y.-H., and Bhargava B. A survey of computation offloading for mobile systems. *Mobile networks and Applications*, 18(1):129–140, 2013.
 - [15] Lepore M., Serra D., and Maccioni R. Leveraging artificial intelligence and optimization for agile agv scheduling in an edge-to-cloud manufacturing framework. *Soft Computing*, 29(13):5041–5069, Jul 2025.
 - [16] Li J., Gao H., Lv T., and Lu Y. Deep reinforcement learning based computation offloading and resource allocation for mec. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2018.

- [17] Li K., Wang X., He Q., Yang M., Huang M., and Dustdar S. Task computation offloading for multi-access edge computing via attention communication deep reinforcement learning. *IEEE Transactions on Services Computing*, 16(4):2985–2999, 2023.
- [18] Lokavarapu V. S., Rao K. N., Gokuruboyina S., and Reddy S. S. An in-depth analysis of computation offloading approaches in multi-access edge computing systems. *Peer-to-Peer Networking and Applications*, 18(4):219, Jun 2025.
- [19] Mach P. and Becvar Z. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.
- [20] Magadán L., Suárez F., Granda J., and García D. Low-cost real-time monitoring of electric motors for the industry 4.0. *Procedia Manufacturing*, 42:393–398, 2020.
- [21] Magnific0 J. Wondershaper. <https://github.com/magnific0/wondershaper>. Accessed: 2025-05-08.
- [22] Mao Y., You C., Zhang J., Huang K., and Letaief K. B. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358, 2017.
- [23] Mehrez I. Tradeoff-aware heuristic service deployment in a mixed edge and cloud computing environment. Master’s thesis, University of Stuttgart, Germany, 2020.
- [24] Okumuş F. and Kocamaz A. F. Cloud based indoor navigation for ros-enabled automated guided vehicles. In *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)*, pages 1–4. IEEE, 2019.
- [25] Park S., Kwon D., Kim J., Lee Y. K., and Cho S. Adaptive real-time offloading decision-making for mobile edges: Deep reinforcement learning framework and simulation results. *Applied Sciences*, 10(5):1663, 2020.
- [26] Peng P., Lin W., Wu W., Zhang H., Peng S., Wu Q., and Li K. A survey on computation offloading in edge systems: From the perspective of deep reinforcement learning approaches. *Computer Science Review*, 53:100656, 2024.
- [27] Pi R. Rpi documentation. <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>. Accessed: 2025-05-08.
- [28] Rohadi E., Rahmad C., Sukmana S. E., Sartimbul A., Ping K. A. H., Joseph A. A., Mat D. A. A., Rahmat M. F., Zakki A. A., and Ramadhan D. N. A review: Performance analysis of single board computer technology as an iot system. In *2025 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*, pages 106–110, 2025.
- [29] Rozlomii I., Yarmilko A., and Naumenko S. Data security of iot devices with limited resources: challenges and potential solutions. In *doors*, 2024.

-
- [30] Shannon C. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.
 - [31] Sutton R. S. and Barto A. G. *Reinforcement learning: An introduction*. MIT press, 2018.
 - [32] Tran-Dang H. and Kim D.-S. Digital twin-empowered intelligent computation offloading for edge computing in the era of 5g and beyond: A state-of-the-art survey. *ICT Express*, 11(1):167–180, 2025.
 - [33] Trinh B. and Muntean G.-M. A deep reinforcement learning-based offloading scheme for multi-access edge computing-supported extended reality systems. *IEEE Transactions on Vehicular Technology*, 72(1):1254–1264, 2023.
 - [34] Weicker R. P. Dhrystone: a synthetic systems programming benchmark. *Communications of the ACM*, 27(10):1013–1030, 1984.
 - [35] Wu D., Wang Z., Pan H., Yao H., Mai T., and Guo S. In-network computing empowered mobile edge offloading architecture for internet of things. *IEEE Transactions on Services Computing*, 17(6):3817–3829, 2024.
 - [36] Wu H. Multi-objective decision-making for mobile cloud offloading: A survey. *IEEE Access*, 6:3962–3976, 2018.
 - [37] Wu (Dissertation) H. Analysis of offloading decision making in mobile cloud computing. *Freie Universität Berlin, Germany*, 2015.
 - [38] Xiao F., Fan W., Han L., Qiu T., and Cheng X. Joint service deployment and task offloading for datacenters with edge heterogeneous servers. *IEEE Transactions on Services Computing*, 18(2):839–853, 2025.
 - [39] Yang Y., Du X., Ye Y., Ding J., Wang T., Chen M., and Li K. Multi-Objective Deep Reinforcement Learning for Function Offloading in Serverless Edge Computing. *IEEE Transactions on Services Computing*, 18(01):288–301, 2025.
 - [40] Yi S., Li C., and Li Q. A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 workshop on mobile big data*, pages 37–42, 2015.
 - [41] Zabihi Z., Moghadam A. M. E., and Rezvani M. H. Reinforcement learning methods for computation offloading: A systematic review. *Association for Computing Machinery*, 56:1–41, 2023.
 - [42] Zhang H., Wu W., Wang C., Li M., and Yang R. Deep reinforcement learning-based offloading decision optimization in mobile edge computing. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–7. IEEE, 2019.