Runtime Management of Artificial Intelligence Applications for Smart Eyewears

Abednego Wamuhindo Kambale
abednegowamuhindo.kambale@polimi.it
Politecnico di Milano
Milano, Italy

Hamta Sedghani
hamta.sedghani@polimi.it
Politecnico di Milano
Milano, Italy

Federica Filippini
federica.filippini@polimi.it
Politecnico di Milano
Milano, Italy

Giacomo Verticale
giacomo.verticale@polimi.it
Politecnico di Milano
Milano, Italy

Danilo Ardagna
danilo.ardagna@polimi.it
Politecnico di Milano
Milano, Italy

Abstract

Artificial Intelligence (AI) applications are gaining popularity as they seamlessly integrate into end-user devices, enhancing the quality of life. In recent years, there has been a growing focus on designing Smart Eye-Wear (SEW) that can optimize user experiences based on specific usage domains. However, SEWs face limitations in computational capacity and battery life. This paper investigates SEW and proposes an algorithm to minimize energy consumption and 5G connection costs while ensuring high Quality-of-Experience. To achieve this, a management software, based on Q-learning, offloads some Deep Neural Network (DNN) computations to the user's smartphone and/or the cloud, leveraging the possibility to partition the DNNs. Performance evaluation considers variability in 5G and WiFi bandwidth as well as in the cloud latency. Results indicate execution time violations below 14%, demonstrating that the approach is promising for efficient resource allocation and user satisfaction.

Keywords: Smart glasses, Smart Eye-Wear, Reinforcement Learning, Edge Computing, Task offloading

ACM Reference Format:
Abednego Wamuhindo Kambale, Hamta Sedghani, Federica Filippini, Giacomo Verticale, and Danilo Ardagna. 2023. Runtime Management of Artificial Intelligence Applications for Smart Eyewears. In *2023 IEEE/ACM 16th International Conference on Utility and Cloud Computing (UCC '23), December 4–7, 2023, Taormina (Messina), Italy.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3603166.3632562

UCC '23, December 4–7, 2023, Taormina (Messina), Italy

ACM ISBN 979-8-4007-0234-1/23/12.
https://doi.org/10.1145/3603166.3632562

1 Introduction

Nowadays Artificial Intelligence (AI) applications are ubiquitous. They are used in different fields (e.g., healthcare, agronomy, entertainment, etc.) and have produced promising results. For instance, in healthcare, AI algorithms help in reducing the time taken for diagnosis and improving patient outcomes [1, 2]. By leveraging AI, farmers can make informed decisions to enhance crop performance and minimize resource wastage [3]. Several companies are working to enable different user devices (e.g., smartphones, smart glasses) and embedded systems to run AI models to provide novel functionalities [4]. These AI applications are often image-based and frequently process Deep Neural Networks (DNNs) with many parameters. The ultimate goal in most cases is to provide applications with real-time processing capabilities.

Despite the effort that has been devoted towards increasing the memory and computational power of IoT and edge systems, there are still some limitations regarding battery capacity and computational power when running complex AI models on device [5, 6]. In some situations, it is possible to run all the DNN layers on a device, while in other cases this is unfeasible due to computational and memory constraints. In both scenarios, the battery capacity might be a limitation that negatively impacts the user experience.

Some studies [7–9] have proposed DNN partitioning as an approach to tackle the aforementioned problems. This technique consists of cutting the DNN at different layers, so that different partitions can be executed on different devices. The DNN partitioning provides different configurations among which the system can switch to provide good performance. Smart Eye-Wears (smart glasses) and virtual/augmented reality applications are a novel use case. The Smart Eye-Wear (SEW) acquires information from its environment, runs a part of the DNN, and offloads the remaining computation to the mobile phone and the cloud. In this context, the problem

is to determine which configuration to run in each time-interval to guarantee a good user experience in response to networks bandwidth or cloud latency variations.

This paper proposes a Reinforcement Learning (RL)-based runtime optimization framework for distributing the computation of AI applications among different computing devices, minimizing the energy and 5G network costs from the user's perspective and the end-to-end latency to guarantee a good user experience. The choice of the RL approach is motivated by its self-adaptive nature and suitability for scenarios involving multiple versions of SEW AI applications. We performed an evaluation of the proposed solution by considering network bandwidth and cloud latency variability. Results indicate execution time violations below 14%, demonstrating that the approach is promising for efficient resource allocation and user satisfaction.

This paper is organized as follows. Section 2 provides an overview of related work on DNN partitioning, resource management, and computation offloading, including studies that employ RL-based approaches. Section 3 describes the scenario considered in this research. Section 4 defines the problem of enhancing AI performance on SEW as a Markov Decision Process (MDP), providing also its RL-based formulation. In Section 5, the experimental setup is presented, and the results of the conducted experiments are discussed. Finally, Section 6 concludes the paper and presents future perspectives.

2 Related Work

With the usage of smart devices, AI applications are being integrated in almost every aspect of life, but big challenges arise when they are deployed on resource-constrained systems with limited computational and energy capacity. To solve this issue, researchers have investigated the possibility of partitioning DNNs and offloading resource-hungry computation to the edge or the cloud. Accordingly, the authors of [10] introduce Neurosurgeon, a collaborative edge-cloud computing approach based on layer-granularity model partitioning, which deploys neural networks to mobile devices and cloud servers and adapts the model partitioning based on network bandwidth or energy consumption. Similarly, the authors of [11] propose an algorithm to find the optimal partitioning point at layer-granularity level in multi-devices and multi-edge servers setups, considering the minimum cost for each edge server corresponding to each mobile device. They used a partition point retain algorithm to reduce the search spaces and run a procedure to find the optimal partition point. However, the authors of both [10] and [11] consider only a two-tier partitioning where computation can only be split between two entities namely the edge device and the edge/cloud server.

Furthermore, many researchers have intensively investigated resource management and task offloading to enable a fair distribution among different entities and avoid paying high costs to guarantee reasonable performance. The study in [12] presents a multi-user and multi-server setup and provides an adaptive multi-objective algorithm to minimize the application power consumption, response time, and cost by relying on a two-tier matching game to solve the underlying resource management optimization problem. However, the authors consider only a two-tier architecture and resource management for generic tasks with no consideration of AI applications. Differently from previous studies, the authors of [13] investigated DNN model partitioning and offloading schemes for multiple users and developed an evolutionary pricing policy and a slot model to plan the offloading of DNN subtasks efficiently.

Recently, RL has been widely used for runtime resource management, intending to optimize a specific objective function that depends on the problem definition and the application requirements. For instance, researchers in [14] developed a task-offloading algorithm based on deep reinforcement learning (DRL) considering logical and data dependency between user application subtasks. The algorithm considers a multi-user and multi-edge setup and adapts well to larger-scale problems, but it ignores the server workload variation, even though a single edge server serves multiple edge devices. Additionally, the offloading mechanism in use is binary, allowing tasks to be executed either entirely locally or exclusively remotely. Authors in [15] formulated the fog-cloud task-offloading problem as an MDP, intending to optimize the average resource utility considering different metrics: the task execution time, the total service latency, the transmission electric power, and the task priority. They used a DRL-based algorithm to solve the defined MDP. However, the work considers the network components as static, which does not usually hold for real-world scenarios.

RL-based resource management has also been expanded to Metaverse applications: authors in [16] proposed an approach to decompose the Metaverse applications in subtasks that can run separately at different parts of the system infrastructure. They also designed a DRL framework to capture the real-time properties of Metaverse applications and learn an optimal policy to manage the execution of those applications at different tiers to maximize the performance of the entire Metaverse system. Nevertheless, this approach prioritizes optimizing the provider's revenue, without considering the user's perspective.

In this work, we propose a three-tier DNN partition model at layer granularity and provide a Q-learning-based runtime offloading scheme considering the variability in the 5G, WiFi throughput, and cloud latency. The novelty of this work is that it solves the runtime management of the AI tasks from the user's perspective to enhance the SEW battery lifetime and reduce the 5G cost, considering the variability at three different system levels while keeping the end-to-end execution time within a reasonable range to ensure a good

user experience. To the best of our knowledge, this is the first paper addressing this problem.

3 Reference Scenario

In this section, we present the reference scenario considered throughout our work to evaluate the approach and the results that we will describe in the next sections. We consider a simple setting where the SEW device runs an AI application comprising a DNN that can perform AI tasks (object classification, object detection, object tracking, etc.).

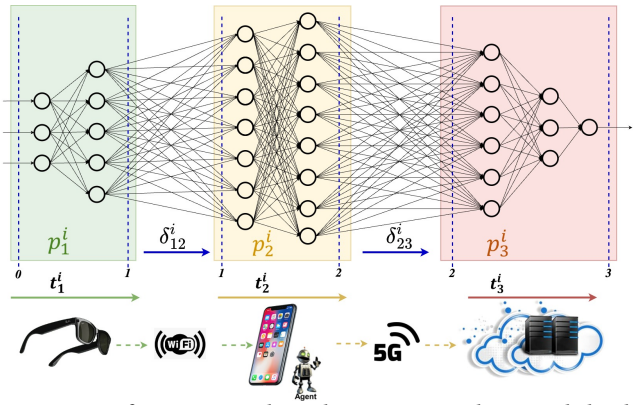


Figure 1. Performing AI task on the SEW, smartphone, and cloud.

The ultimate goal is to guarantee a good user experience by introducing AI application execution time constraints. For instance, when considering a tracking application the application frame rate should not be lower than 30 frames per second (fps) to guarantee enjoyability, thus the end-to-end execution time should be lower than 33 ms [17]. For Simultaneous Localisation And Mapping (SLAM) applications, a frame rate of 15-30 fps is acceptable [18] implying the end-to-end execution time to lie between 33 ms and 66 ms.

Figure 1 presents the three main entities of the system, namely the SEW device, the mobile phone, and the cloud server. We assume the SEW is based on the next-generation Qualcomm System-on-Chip (SoC) and is connected to the phone using WiFi 7, which also has a Qualcomm SoC (e.g., according to the characteristics of the next Qualcomm SoCs, the smart glass will be based on the Snapdragon AR2 chip while the mobile phone on Snapdragon 8 Gen 2[1]). The SEW device has a CPU, an AI neural engine to perform some computation, and a battery with limited capacity power. The same applies to the mobile phone (which includes a CPU and a GPU).

The SEW acquires data from its surrounding environment and sends them to the mobile phone. Periodically, the acquired data are sent to the mobile phone with or without undergoing some initial processing, depending on the system state (e.g., battery level, network quality). When the mobile phone receives data from the SEW, it processes them through the DNN. The application execution might end at this stage if the mobile phone performs all the remaining computations. However, similarly to the SEW, the mobile phone might need to offload some computations to the cloud depending on the system state, to satisfy the system constraints and to enable a good user experience. In this case, the cloud will perform all remaining computations and return the results.

The system components communicate through different network domains. The phone uses a 5G network to send data to the cloud server. This communication setup can considerably reduce the transfer delay between two connected entities [19, 20]. It is well-known that 5G mmWave systems provide large bandwidth but are characterized by huge variability [21]. WiFi interference may also occur when multiple devices exploiting it are operating on the same frequency, leading to disruptions and variations in network throughput impacting the overall performance of WiFi-based systems and applications [22]. Cloud servers also exhibit some performance variability due to their workload fluctuations, which impacts servers queuing time and hence the AI application end-to-end latency [23]. Our management system will take local decisions to cope with such variability, but the cloud resource allocation is considered out of our control.

The AI-based applications consisting of DNNs are resource-intensive and energy-consuming on local devices; DNN partitioning allows to offload some computational tasks, offering significant benefits. The DNN is characterized by different candidate partitioning points, and this enables the SEW to execute all or part of its layers. Indeed, the SEW can either execute the whole DNN locally or offload a part of the computation according to the decision of a Reinforcement Learning (RL) agent running in the phone, which considers the actual battery state and the available computing power.

In this work, the RL agent on the mobile phone decides which DNN configuration to run. We adopted the concept of configurations, since the study in [10] shows that only some partitioning points make sense from the point of view of energy consumption and data transfers between the different system components. The RL agent takes one action per minute or, if the time constraint is violated, one every 5 s. The aim is to choose the configuration that minimizes energy consumption and 5G connection cost while satisfying execution time constraints to guarantee a good user experience.

Let $\mathcal{K}$ denote the set of all candidate configurations. For simplicity, we assume that every configuration $\kappa^i \in \mathcal{K}$ has three partitions, and we introduce a new partition $p_\emptyset^i$ to denote that nothing is executed. This establishes a clear link between partitions and devices, so that, when the agent chooses a configuration, we know directly which partition to execute on each layer. In particular, p_1^i is always executed on the SEW, p_2^i on the mobile phone, and p_3^i on the cloud

[1] https://www.qualcomm.com/products/mobile/snapdragon/xr-vr-ar/snapdragon-ar2-gen-1-platform

server. p_j^i is $p_\emptyset^i$ if nothing is executed on the device j. δ_{dk}^i represents the amount of data per request (in byte) sent between partitions, where δ_{12}^i indicates the data sent from the SEW to the smartphone and δ_{23}^i from the smartphone to the cloud. If the execution occurs fully on the SEW, $\delta_{12}^i = \delta_{23}^i = 0$, and if everything is offloaded to the cloud, $\delta_{12}^i = \delta_{23}^i = \delta_0$, where δ_0 is the size of the input tensor. With the knowledge of partition-device assignments, the latencies of executing the partitions of configuration κ^i on the SEW, mobile phone, and cloud can be denoted as t_1^i, t_2^i, t_3^i, respectively, and can be determined by profiling the DNN [10].

The end-to-end execution time comprises the local execution time, the data-transfer latency from the SEW to the smartphone, the smartphone execution time, the data-transfer latency from the smartphone to the cloud, and the cloud execution time. In this work, we will neglect the time needed to send back the result, as it is negligible compared to the execution time and data-transfer latency. For instance, considering an object detection task, the data that are sent back to the SEW are the labels of the objects, their bounding boxes, and their class probabilities, whose sizes are negligible compared to the size of the intermediate tensor that is sent.

4 Problem Definition

When the SEW runs an AI task, the decision agent determines whether this should be fully executed on the SEW or if the execution of some DNN layers should be offloaded to the smartphone or the cloud server. This is done by choosing a DNN configuration. We can formulate this decision problem as the minimization of the total processing cost:

$$\min \ (\alpha e + c_{5G})\, \tau \tag{P1a}$$

subject to:

$$l_{total} < L_{max}, \tag{P1b}$$

where e is the energy consumed by the SEW, computed as in [24], being the sum of the energy to run p_1^i on the SEW ($e_{local} = \sum_{i=1}^{|\mathcal{K}|} z_{SEW}\, \mu_1^i\, x^i$) and the energy needed to transfer the tensor δ_{12} to the mobile phone ($e_{tr} = \sum_{i=1}^{|\mathcal{K}|} \theta_{SEW}\, \delta_{12}^i\, x^i / r_{WIFI7}$), z_{SEW} is the electrical energy needed by the SEW to perform one FLOP, and θ_{SEW} the electrical power required by its network interface while sending data. μ_1^i is the workload (in FLOPs) of the partition p_1^i. x^i equals 1 if the configuration κ^i is chosen and 0 otherwise. The cost c_{5G} is given by $c_{5G} = \sum_{i=1}^{|\mathcal{K}|} g\, \delta_{23}^i\, x^i$, where g is the cost for sending 1 byte of data from the mobile phone to the cloud using the 5G connection. l_{total} is the total execution time, given by $l_{total} = l_{SEW} + l_{phone} + l_{sp} + l_{pc} + l_{cloud}$ where $l_{SEW} = \sum_{i=1}^{|\mathcal{K}|} t_1^i\, x^i$ and $l_{phone} = \sum_{i=1}^{|\mathcal{K}|} t_2^i\, x^i$ represent the execution time of partition p_1^i on the SEW and p_2^i on the mobile phone, respectively, $l_{sp} = \sum_{i=1}^{|\mathcal{K}|} \delta_{12}^i\, x^i / r_{WIFI7}$ and $l_{pc} = \sum_{i=1}^{|\mathcal{K}|} \delta_{23}^i\, x^i / r_{5G}$ are the time to transfer δ_{12} from SEW to the phone and δ_{23} from the phone to the cloud. l_{cloud} is the cloud execution time, which depends on t_3^i and the current workload on the cloud server.

The goal is to minimize the energy consumption e of the SEW and the 5G connection cost c_{5G}. In the objective function (P1a), α denotes the energy unit price cost (measured in \$/J), and τ is the control time period. This work does not deal with the cost of the cloud service since we are trying to optimize the parameters from the user's perspective. The optimization of the overall system will be the objective of future work. In the constraint (P1b), L_{max} is the maximum time latency that guarantees a good user experience.

We formulate the offloading problem as a discrete-time infinite-horizon MDP. The easiest way to solve the MDP problem is to use dynamic programming, but in some cases this is not possible due to the state space dimension or to the fact that the system dynamics are unknown. Thus, we present an RL-based approach to tackle the MDP.

An MDP can be defined by a 5-tuple $\langle \mathbf{S}, \mathbf{A}, P, c, \gamma \rangle$, where $\mathbf{S}$ represents the (possibly infinite) set of all the possible states; $\mathbf{A}(s)$ denotes the finite set of all possible actions in state s; $P(s'|s,a)$ is the transition probability from a given state s to a state s' given an action $a \in \mathbf{A}(s)$; $c(s,a,s')$ is the immediate cost when an action a is executed in state s and the system transits to state s'; and $\gamma \in [0,1]$ is a discount factor that adjusts the importance of future costs.

We define the agent state as $s = (r_{WIFI7},\ r_{5G},\ l_{SEW},\ l_{phone},\ l_{cloud})$, where the data rates r_{WIFI7} and r_{5G}, and the execution time in the cloud l_{cloud} are exogenous parameters that are not under the agent's control. Therefore, their variability is independent of the chosen actions and is simply observed from the environment. Moreover, we assume that the phone is in light load and there are no other tasks running on the phone and consuming resources. On the other hand, the SEW and phone latencies, i.e., l_{SEW} and l_{phone}, change according to the actions, but their value is known a priori: indeed, we can estimate the execution times of the partitions executed on the SEW and the smartphone for all the configurations.

If we discretize all the continuous variables with 10 values each, the state space has a dimension in the order of 10^5.

The action consists in selecting a configuration. Therefore, the number of actions equals the number of configurations, which we denote by $|\mathcal{K}|$. Note that, in some states, the agent may choose not to change the configuration selected in the previous time window; we model this scenario by introducing an action η that represents the *do nothing* choice. The set of all the possible actions $\mathbf{A}(s)$ is hence given by $\mathbf{A}(s) = \{a^1, a^2, \ldots, a^{|\mathcal{K}|}\} \cup \{\eta\}$.

The system dynamic is stochastic; hence, the transition probability matrix is unknown and cannot be determined for the moment. We associate a cost $c(s,a,s')$ to each triple state-action-next state. This embeds the energy cost $c_e(s,a) = \alpha e$, the 5G connection cost $c_{5G}(s,a)$, and the penalty $c_{ex}(s,a,s')$ we pay for violating the execution time requirement (Constraint (P1b)). In a given time slot, if the agent chooses an action different from η, the system must be reconfigured, which

induces some time overhead. Therefore, we introduce a configuration penalty c_{cfg} defined as $c_{cfg} = \mathbb{1}_{\{a \neq \eta\}}$, meaning that we pay a penalty equal to 1 every time the chosen action is different from η, 0 otherwise. Similarly, the cost $c_{ex}(s, a, s')$ incurred when the system violates the execution time constraint can be simply defined as $c_{ex}(s, a, s') = \mathbb{1}_{\{l_{total} > L_{max}\}}$.

We combine the different costs using a simple additive weighting approach [25], defining $c(s, a, s')$ as:

$$c(s, a, s') = \omega_e \frac{c_e}{e_{max}} + \omega_{con} \frac{c_{5G}}{c_{5G,max}} + \omega_{ex} c_{ex} + \omega_{cfg}\, c_{cfg}$$

where ω_e, ω_{con}, ω_{ex} and ω_{cfg} are non-negative weights summing up to 1. e_{max} and $C_{5G,max}$ are the normalization parameters for the energy consumption and the cost of the 5G connection, respectively. The effect of different factors such as the battery level and the computational load of the device is implicitly embedded in the definition of the weight ω_e and the energy consumption.

As the system dynamic is unknown, it is impossible to solve this MDP using traditional dynamic programming. Hence, we propose a model-free Q-learning approach to tackle this problem.

Algorithm 1 Epsilon-greedy Q-learning algorithm

Require:
States S
Actions A
Cost function C
Learning rate $lr \in [0, 1]$
Discount factor $\gamma \in [0, 1]$
Epsilon $\varepsilon \in [0, 1]$
procedure QLearning(S, A, C, lr, γ, ε)
  Initialize $Q : S \times A \rightarrow \mathbb{R}$ arbitrarily
  Start in state $s \in S$
  while Q has not converged **do**
    Choose action using ε-greedy policy
    $a \leftarrow \varepsilon$-greedyAction(S, A, ε)
    $s' \leftarrow$ Observe state after a ▷ Observe the new state
    $c \leftarrow C(s, a, s')$ ▷ Incur the cost
    $Q(s', a) \leftarrow (1 - lr)Q(s, a) + lr(c + \gamma \min_{a'} Q(s', a'))$
    $s \leftarrow s'$
  end while
  return Q
end procedure

The agent stores the Q-function $Q(s, a)$, which is an estimate of the expected long-term costs that result from the execution of an action a in a state s. The agent uses these estimates to decide which action to take at the next step. When taking an action, the agent observes the current costs and updates these estimates over time, thus improving its policy.

Q-learning simply uses sample averages to estimate the optimal Q-function. We consider an ε-greedy action selection approach that, at each step τ, chooses the greedy action (i.e, $a^\tau = \text{argmin}_{a \in A(s^\tau)} Q(s^\tau, a)$) with probability $1-\varepsilon$, exploiting its knowledge of the system (exploitation), whereas it picks a random action with probability ε to enhance its knowledge of the application (exploration). $Q(s^\tau, a^\tau)$ is updated at the end of time step τ as shown in Algorithm 1.

The ε-greedy policy guarantees the exploration of suboptimal actions with low probability, while choosing the optimal action for a given state most of the times.

5 Experimental Analysis

The performance evaluation of the RL agent is conducted under a scenario featuring network bandwidth variability and cloud performance fluctuation. Section 5.1 presents the experimental setup, while Section 5.2 provides the experimental results obtained under these settings.

5.1 Experimental setup

We selected YOLOv5 as a representative model of DNN for SEW applications. It employs a Convolutional Neural Network (CNN) as its backbone and includes additional operations in its neck and head to enable object detection. Furthermore, we accommodated the constraints of SLAM applications, which require an end-to-end execution time spanning from 33 ms to 66 ms [18].

We partitioned and profiled YOLOv5 to get the execution of different partitions on the smart glasses, the mobile phone, and the cloud. To obtain reasonable values, the profiling of the DNN was conducted using real user-side hardware, specifically the Jetson TX2[2] device. The execution times on the edge side were obtained by running the docker containers on a desktop PC. Profiling highlighted that the end-to-end execution ranges between 500 ms and 1000 ms for most of the configurations, which is far away from the SEW application requirements. This is a limitation of the currently available hardware; since we assume that faster CPUs and GPUs will be available in the short future, we normalized the processing times of the computational layers to comply with the application execution time requirement and took six configurations to perform the experiments.

Table 1 presents the parameters of those configurations. The first three configurations represent the case where the whole DNN runs on the SEW, the phone, or the cloud, respectively. In the fourth and the fifth configurations, the workload is distributed on the three devices and every device executes a part of the DNN. The last configuration represents the scenario where the workload is distributed between the phone and the cloud.

Specifically, Table 1 includes information such as the sizes of transferred data δ_{12} and δ_{23} (in MB) from the SEW to the phone and from the phone to the cloud, respectively. Latency (measured in ms) and workload (measured in MFLOPs) for each partition on their respective devices are also presented.

[2]https://developer.nvidia.com/embedded/jetson-tx2

Table 1. DNN configuration parameters

Configuration parameters								
Config	δ_{12}(MB)	δ_{23}(MB)	Partition 1		Partition 2		Partition 3	
			latency(ms)	μ(MFLOPs)	latency(ms)	μ(MFLOPs)	latency(ms)	μ(MFLOPs)
1	0	0	40	17382	0	0	0	0
2	1.2	0	0	0	30	17382	0	0
3	1.2	1.2	0	0	0	0	6	17382
4	6.25	2.4	15	2987	15	7048	4	7347
5	2.4	2.4	20	5988	8	6432	2	4962
6	1.2	4	0	0	15	8042	4	9340

As already mentioned, Partition 1, 2, and 3 are always executed on the SEW, phone, and cloud, respectively, unless they coincide with $p^i_\emptyset$ (in which case, the corresponding latency and μ value in the table will take the value 0). The first three configurations depict exclusive DNN operation on the SEW, phone, or cloud. In the fourth and fifth setups, workload distribution across all three devices is demonstrated, where each handles a section of the DNN. The last configuration showcases workload division between the phone and the cloud. The interesting characteristics of the SEW in the experiments are the electrical energy z_{SEW} it needs to perform one FLOP and the electrical power θ_{SEW} required by the network interface while sending data. According to results in [24], z_{SEW} can be roughly estimated to 8 pJ/FLOP for the Snapdragon SoC 855 and θ_{SEW} is set to 700 mW which is the average value of the power consumption of the network interface during data-intensive tasks like video streaming and upload or web browsing.

The RL agent implements the Q-learning algorithm with an ε-greedy policy. We tried different values for ε, and we finally fixed $\varepsilon = 0.02$ for all the experiments. We investigated the agent behavior with a learning rate that is initialized to 1 and decays exponentially with the number of training steps. The action space of the agent comprises the 6 actions that correspond to the selection of each of the 6 configurations, plus η (DoNothing). The discount factor γ was set to 0.99 in all experiments, and we considered an MDP with an infinite horizon. For each experiment, we run the agent for a total of 250000 steps. We set L_{max} to 50 ms, while the weights we used are $\omega_{ex} = 0.9$, $\omega_e = 0.06$, $\omega_{con} = 0.02$ and $\omega_{rcfg} = 0.02$, privileging the execution time metric.

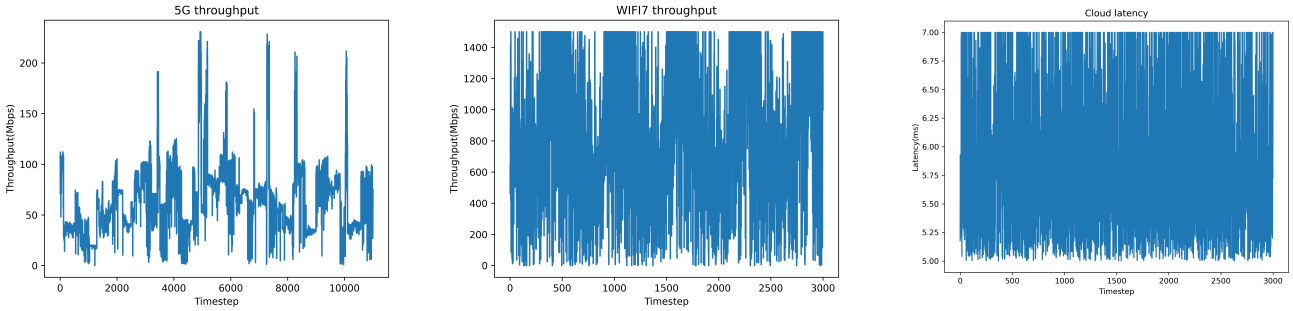
Finally, we considered real traces [21, 26] for WiFi6 and 5G. The 5G dataset includes 11,024 rows, and the uplink throughput values range between 0 and 230.75 Mbps. As an experiment takes 250,000 steps to complete, we replayed the trace after 11,024 steps and, to avoid the periodicity in the trace, we performed some operations as random shifts, insertion of random noise (± 10% of the value at time step t) and the inversion of the trace in its middle. Figure 2a provides a snapshot of the 5G trace for the first 11,000 time steps.

The WiFi6 trace was collected from a simulator considering a scenario with nine access points placed at a center of a region of 10×10 meters, each serving up to five devices. For each device, the WIFI6 dataset includes overall 3000 rows. We scaled up all the values between 0 and 1500 Mbps to approximate the preconized WiFi7 throughput, and replayed the trace similarly to the 5G data. Figure 2b provides a snapshot of the WiFi7 trace for 3000 steps.

To emulate the behavior of real-world applications, we treated the cloud server as an M/M/1 queue, where incoming requests follow a Poisson distribution according to a widely used assumption in cloud and edge literature [27]. The cloud latency trace was obtained by drawing the cloud execution time from exponential distributions whose mean values depend on the chosen DNN configuration, according to the values reported in Table 1. A snapshot of the cloud latency trace corresponding to running the third configuration for the first 3000 time steps is given in Figure 2c. Note that, in all experiments, we compute the reward as the negative of the cost.

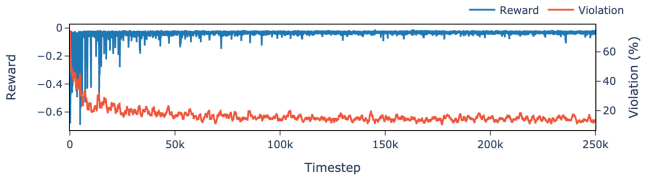
5.2 Experimental results

The plots presented in Figure 3 illustrate the outcomes obtained after training the agent over 250,000 steps. Figure 3a displays the agent reward over time and the moving average of violations computed with a window of 1000 steps. These plots are averages of 10 experiments conducted in parallel. During the initial steps, the agent reward is not optimal, as it lacks knowledge of the system dynamics. This observation is further reinforced by Figure 3b, which depicts the execution time violations for different intervals. During the initial 2000 steps, a higher number of violations is evident, as the agent actively explores the configurations space to acquire system knowledge. During this phase, it also selects configurations associated with significant costs. Additionally, the reward improves with time, and after 50,000 steps we can observe that the agent has successfully learned an effective policy

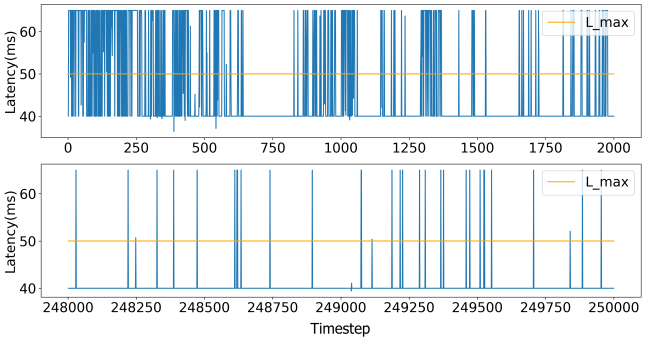


(a) Snapshot of the 5G trace from 0 to 11,000 steps **(b)** Snapshot of the WIFI7 trace for 3,000 steps **(c)** Snapshot of the cloud latency for 3,000 steps

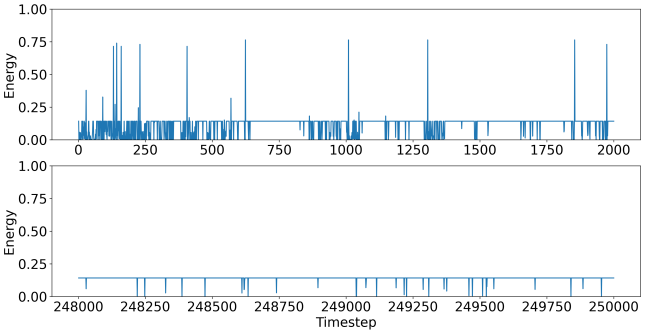
Figure 2. Traces used for the experiments as sources of variability to the system



(a) Reward and cumulative violation.



(b) Execution time and violation at different time steps interval.



(c) Normalized energy consumption at different time steps.

Figure 3. Experimental results with variation in the 5G, WIFI throughputs & Cloud latency and decaying learning rate. The energy is normalized considering the maximum energy consumption of the SEW.

to maximize its reward. On average, the violations incurred during the 250,000 steps amount to 13.8%.

The plots in Figure 3c depict the energy consumption of the SEW, normalized between 0 and 1, in different time intervals. The agent is observed to effectively help the SEW save energy over the course of training, which is beneficial to enhance the SEW battery lifespan. Based on the conducted experiments, the agent demonstrates adaptability to system configurations. Moreover, the agent exhibits the ability to save energy intermittently, which is advantageous for the SEW performance.

6 Conclusion

In this paper, we designed an agent using the Q-learning method to manage at runtime the execution of an AI application running on a SEW, a local smartphone, and possibly on cloud. Taking advantage of the DNN partitioning mechanism, the RL agent chooses the appropriate DNN configuration to enhance the SEW battery lifetime while guaranteeing better quality-of-experience and a maximum tolerable latency. We investigated the behavior of the RL agent with different sources of variability, such as 5G and WiFi throughput, and cloud latency. The results show that our RL agent manages to learn the optimal policy: the overall execution time violation below 14% highlights that we can rely on it for the runtime management of SEW-based AI applications guaranteeing the user satisfaction.

In future works, HyperOpt, a Bayesian tool for hyperparameter optimization, could be used to determine the optimal hyperparameter configuration; various RL algorithms could be implemented and compared with the current findings; and the trained agent might be evaluated in a real environment where parameters change dynamically to assess real-world performance.

Acknowledgments

This work has been sponsored by the EssilorLuxottica Smart Eyewear Lab.

References

[1] F. D'Ascenzo, O. De Filippo, G. Gallone, and et al. Machine learning-based prediction of adverse events following an acute coronary syndrome (praise): a modelling study of pooled datasets. *The Lancet*, 397:199–207, 01 2021.

[2] S. D. Raj and Karthiban. Applications of artificial intelligence in healthcare. In *International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–2, 2022.

[3] E. Elbasi, N. Mostafa, and Z. AlArnaout. Artificial intelligence technology in the agricultural sector: A systematic literature review. *IEEE Access*, 11:171–202, 2023.

[4] W. Yao. The application of artificial intelligence in the internet of things. In *International Conference on Information Technology and Computer Application (ITCA)*, pages 141–144, 2019.

[5] J. Coelho and L. Nogueira. Enabling processing power scalability with internet of things (iot) clusters. *Electronics 2022, Vol. 11, Page 81*, 11:81, 12 2021.

[6] O. Debauche, S. Mahmoudi, and A. Guttadauria. A new edge computing architecture for iot and multimedia data management. *Information 2022, Vol. 13, Page 89*, 13:89, 2 2022.

[7] J. Karjee, P. Naik S, and K. Anand. Split computing: Dnn inference partition with load balancing in iot-edge platform for beyond 5g. *Measurement: Sensors*, 23:100409, 10 2022.

[8] A. Parthasarathy and B. Krishnamachari. Partitioning and placement of deep neural networks on distributed edge devices to maximize inference throughput. In *ITNAC*, pages 239–246. IEEE, 2022.

[9] N. Y. Yen, C. Yang, and C. Tsung. Partitioning dnns for optimizing distributed inference performance on cooperative edge devices: A genetic algorithm approach. *Applied Sciences 2022, Vol. 12, Page 10619*, 12:10619, 10 2022.

[10] Y. Kang, J. Hauswald, C. Gao, and et al. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGPLAN Notices*, 52:615–629, 4 2017.

[11] Z. Liao, W. Hu, J. Huang, and et. al. Joint multi-user dnn partitioning and task offloading in mobile edge computing. *Ad Hoc Networks*, 144:103156, 2023.

[12] P. Wang, K. Li, B. Xiao, and K. Li. Multiobjective optimization for joint task offloading, power assignment, and resource allocation in mobile edge computing. *IEEE INTERNET OF THINGS JOURNAL*, 9, 2022.

[13] M. Gao, R. Shen, L. Shi, and et al. Task partitioning and offloading in dnn-task enabled mobile edge computing networks. *IEEE Transactions on Mobile Computing*, 22(4):2435–2445, 2023.

[14] B. Gong and X. Jiang. Dependent task-offloading strategy based on deep reinforcement learning in mobile edge computing. *Wireless Communications and Mobile Computing*, 2023:1–12, 01 2023.

[15] V. Jain and B. Kumar. Qos-aware task offloading in fog environment using multi-agent deep reinforcement learning. *Journal of Network and Systems Management*, 31(1):7, 2023.

[16] N. H. Chu, D. N. Nguyen, D. T. Hoang, and et al. Dynamic resource allocation for metaverse applications with deep reinforcement learning. In *2023 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2023.

[17] W. Luo, J. Xing, A. Milan, and et al. Multiple object tracking: A literature review. *Artificial Intelligence*, 293:103448, 2021.

[18] R. Gomez-Ojeda, F. A. Moreno, D. Zuñiga-Noël, and et al. Pl-slam: A stereo slam system through the combination of points and line segments. *IEEE Transactions on Robotics*, 35:734–746, 6 2019.

[19] R. Dangi, P. Lalwani, G. Choudhary, and et al. Study and investigation on 5g technology: A systematic review. *Sensors*, 22, 1 2022.

[20] C. Deng, X. Fang, X. Han, and et al. Ieee 802.11be-wi-fi 7: New challenges and opportunities. *IEEE Communications Surveys and Tutorials*, 22:2136–2166, 7 2020.

[21] A. Narayanan, X. Zhang, R. Zhu, and et al. A variegated look at 5g in the wild: Performance, power, and qoe implications. In *ACM SIGCOMM*, page 610–625, 2021.

[22] T. Pulkkinen, J. K. Nurminen, and P. Nurmi. Understanding wifi cross-technology interference detection in the real world. In *IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 954–964, 2020.

[23] D. Ardagna, G. Casale, M. Ciavotta, and et al. Quality-of-service in cloud computing: modeling techniques and their applications. *J. Internet Serv. Appl.*, 5(1):11:1–11:17, 2014.

[24] Z. Towfic, D. Ogbe, J. Sauvageau, and et al. Benchmarking and testing of qualcomm snapdragon system-on-chip for jpl space applications and missions. In *IEEE Aerospace Conference (AERO)*, pages 1–12, 2022.

[25] K. P. Yoon and C. Hwang. *Multiple attribute decision making: an introduction*. Sage publications, 1995.

[26] F. Wilhelmi. [ITU AI/ML Challenge 2021] Dataset IEEE 802.11ax Spatial Reuse, September 2021.

[27] U. Tadakamalla and D. A. Menascé. Autonomic resource management for fog computing. *IEEE Transactions on Cloud Computing*, 10(4):2334–2350, 2022.