IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 13 March 2024, accepted 17 May 2024, date of publication 22 May 2024, date of current version 30 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3404272

RESEARCH ARTICLE

Energy-Aware Selective Inference Task Offloading for Real-Time Edge Computing Applications

ABDELKARIM BEN SADA[1], AMAR KHELLOUFI[1], ABDENACER NAOURI[1], HUANSHENG NING[1], (Senior Member, IEEE), AND SAHRAOUI DHELIM[2]
[1]School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China
[2]School of Computer Science, University College Dublin, Dublin 4, D04 V1W8 Ireland

Corresponding author: Sahraoui Dhelim (sahraoui.dhelim@ucd.ie)

ABSTRACT IoT has recently witnessed a boom in AI deployment at the edge as a result of the newly developed small size Machine Learning (ML) models and integrated hardware accelerators. Although it brings huge benefits such as privacy-preserving and low-latency applications, it still suffers from typical resource limitations of edge devices. A new approach aims to deploy multiple inference models varying in size and accuracy onboard the edge device which could alleviate some of these limitations. This dynamic system can be leveraged to provide real-time energy efficient application by smartly allocating inference tasks to inference local models or offload to edge servers based on current constraints. In this work, we tackle the problem of efficiently allocating inference models for a given set of inference tasks between local inference models and edge server models in parallel under given time and energy constraints. This problem is considered strongly NP-hard and therefore we propose LITOSS, a 2-stage framework in which we use a lightweight Genetic Algorithm-based schemer for task scheduling along with a Reinforcement Learning (RL) agent for improving edge server selection. We perform experiments using a raspberry pi with a set of edge servers. Results show that our framework performed relatively faster compared to other meta-heuristic schemes such as LGSTO, Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) while providing higher average accuracy. We also show that using an RL agent to select the best subset of available edge servers increased, or maintained in worst cases, the average accuracy while reducing the average scheduling times.

INDEX TERMS Selective sensing, edge computing, machine learning, task offloading, genetic algorithms, reinforcement learning.

I. INTRODUCTION

AI has gained an increasing significance in various IoT applications. Specially with the emergence of compact AI models suitable for deployment on edge devices with limited resources which has ushered in a new era of low-latency applications in IoT. These small AI models, designed to run locally on edge devices rather than relying on remote servers, facilitate real-time processing and decision-making directly at the point of data collection or interaction [1].

One key application area for these local AI models is in smartphones, where they enable a range of real-time functionalities, including image and video recognition as well as Augmented Reality (AR). With local AI models, smartphones can perform tasks such as object detection, facial recognition, and overlaying digital information onto the physical world in real-time without needing to transmit data to distant servers. This not only reduces latency significantly but also enhances power efficiency by minimizing the need for data transmission and processing in the cloud [2], [3].

The emergence of advanced small-size AI models and Large Language Models (LLMs) such as Llama 2 by Meta [4], which represents a significant advancement in on-device AI capabilities. One of the key advantages of these small-scale models is their ability to perform on-device inference with reasonable runtimes and accuracy levels. This capability enables the provision of low-latency services, allowing for swift responses to user queries or inputs. Additionally,

The associate editor coordinating the review of this manuscript and approving it for publication was M. Anwar Hossain.

executing inference tasks locally on the device contributes to more efficient energy consumption compared to offloading these tasks to remote cloud services. By reducing the need for data transmission and processing over the network, on-device AI inference helps conserve battery life and optimizes the device's energy usage [5].

However, it is important to recognize that not all inference tasks can be effectively performed locally on edge devices while still providing real-time responses and maintaining energy efficiency. Some tasks may require more computational resources or specialized hardware than what is available on the device. In such cases, offloading these tasks to nearby edge servers, which have more powerful hardware and resources, may be necessary to achieve optimal performance.

Therefore, there exists a need to strike a balance between performing inference tasks locally on edge devices and offloading them to nearby edge servers. This balance ensures that tasks are executed in the most efficient manner possible, taking into account factors such as latency, energy consumption, computational resources, and real-time requirements [3]. By leveraging a combination of on-device and edge server processing, organizations can optimize the performance of their AI-driven applications while maximizing resource utilization and minimizing operational costs [2].

Hardware manufacturers are driving these trends by integrating AI accelerators into modern processors. This in turn enables edge devices to perform AI tasks locally, without relying heavily on cloud-based processing [1]. To accommodate the limitations of edge devices, such as limited computational power and storage capacity, developers are deploying multiple AI models with varying sizes and accuracy to allow for dynamic switching between them depending on the device's resource conditions. The size of an AI model dictates its inference time and accuracy. Larger models, which typically have more parameters and complexity, tend to produce more accurate inference results but require longer processing times. Conversely, smaller models, which have fewer parameters and are less complex, can process data more quickly but may sacrifice accuracy to some extent. One approach to address this trade-off between accuracy and processing time is to adjust the hyperparameters of the AI models. By tweaking these hyperparameters, the accuracy of models can be fine-tuned according to the specific requirements of the edge application. This flexibility allows edge nodes to deploy multiple local inference models, each optimized for different accuracy levels [6].

Considering an edge computing system empowered by AI models in which edge devices receive sensed data from the environment or from user interactions (see Fig 1). These devices are equipped with pretrained inference models which can be used to perform inference on data locally. Inference models can be of similar types with different internal structures or of different types. Each edge device is connected to a set of edge servers available for task offloading. Edge servers are equipped with more powerful and faster inference models. Edge nodes can choose to process data locally or offload tasks

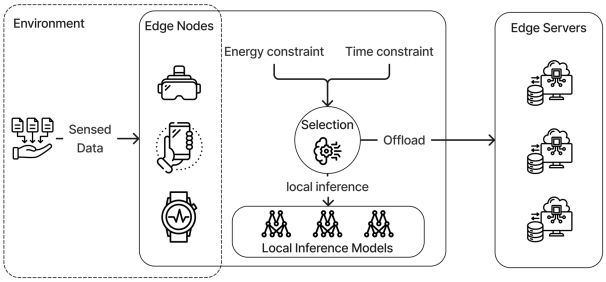

FIGURE 1. Edge computing system with inference task scheduling and offloading.

to multiple edge servers to achieve the best performance and energy efficiency under any given time and energy constraints.

The problem of assigning inference models to inference tasks is similar in nature to a more complex variant of the knapsack problem, where we attempt to fill a volume and weight limited knapsack with given pieces to maximize profit. In our case we try to fill a time and energy limited schedule with inference models to maximize accuracy. This type of problem is considered NP-Hard and therefore it has no polynomial time solutions. Since an edge device can be connected to large number of edge servers at a time, this poses another challenge where only the best subset of edge servers needs to be selected. Therefore, In this research, we introduce an edge computing framework for AI-powered real-time application leveraging meta-heuristic and reinforcement learning methods for parallel inference task scheduling under time and energy constraints while maximizing inference accuracy [7].

The main contributions of this paper can be summarized as follows:

- We provide a formulation to the novel problem of inference model scheduling between local inference and edge server offloading in parallel under time and energy constraints.
- We propose LITOSS a lightweight framework for inference task scheduling and offloading for edge computing consisting of two main modules namely LITS that is based on genetic algorithms for parallel scheduling of tasks, in addition to a reinforcement learning agent using the SARSA algorithm for learning the best edge server selection policy.
- We perform experiments using a Raspberry Pi as an edge device connected to multiple edge servers and compare the performance of the framework against other metaheuristic schemes such as LGSTO, Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO). We find that LITOSS offers higher average accuracy schedules and better constraint conformity compared to other schemes while resulting in lower scheduling times thanks to the edge server selection agent.

The rest of this paper is organized as follows. Section II presents the related works and points out the research gap.

In Section III we describe the system model. In Section IV we propose LITOSS and explain the framework components. In Section V we present the experiment setup and results in addition to analysis of the obtained results. Finally, we conclude this work in Section VI.

II. RELATED WORKS

In this section we organise the related works into two categories. First, we present works tackling the problem of inference model scheduling in edge computing. Second, we show edge server selection solutions proposed in the literature. Finally, we compare and analyse related works limitations and point out the research gap.

A. INFERENCE TASK SCHEDULING AND OFFLOADING

Task offloading and scheduling in the context of inference models where accuracy plays an important role has seen little attention in the literature except for a few recent works mentioned here.

In [8], the authors propose a confidence metric data selection scheme in edge nodes used to select the data samples that could lead to poor accuracy inferences in which case they are offloaded to the edge server. Their results show an improved overall accuracy under an energy constraint. Their work however, does not leverage multiple inference models and does not consider any time constraints which makes it unfit for real-time applications.

The authors in [6] proposed AMR^2, a scheduling scheme based on LP-Relaxation and rounding which considers all possible cases of scheduling two inference tasks between the edge device and the edge server. They relax the problem's constraint to take fractional values and then perform rounding to get the result. Inference tasks are scheduled at the edge device using dynamic programming. Although their scheme performed better than the greedy scheme under time constraint, their system does not take into consideration the energy constraint of the edge node.

The work in [5] proposes LGSTO a lightweight genetic algorithm for inference task scheduling in which their work shows promising results where LGSTO performed 70% faster than other metaheuristic schemes such as PSO, ACO, and Non-dominated Sorting Genetic Algorithm 2 (NSGA-II) while producing schedules with higher average accuracy. However, two limitations have not been considered. First, LGSTO only generates sequential schedules in which the edge node stays idle while the edge server is processing the offloaded inference task. Although this conserves energy, it wastes precious CPU time specially in the case of real-time applications. Secondly, they only considered a single edge server while in practice an edge node can have access to multiple edge servers at once which opens the door for more parallel processing.

B. SERVER SELECTION IN EDGE COMPUTING

In this section we present recent works solving the server selection in edge computing using multiple parameters such as network state, energy of edge devices, server capacity and load.

A collaborative task offloading solution between Mobile Edge Computing (MEC) servers and the cloud is proposed in [9]. Considering dynamic environments such as channel latency variations, energy of mobile devices and edge server computing capacity they propose a Deep Q-Networks (DQN) solution to find the optimal stationary control policy based on recursive decomposition of action space available to each state. Their proposed solution outperformed the block successive upper-bound minimization method (BSUM) and Q-learning methods in simulations.

To solve the server selection problem in dynamic MEC with frequent user mobility, the authors in [10] modeled the problem of continuous server selection as a Markov Decision Process, and proposed a Deep Reinforcement Learning-based algorithm to learn the selection policy based on the observed performance of past server selections. Instead of simply adding user information into the states which results in a complex state space, they modeled user arrivals, departures and movement as different events. Using a Long Short-Term Memory (LSTM), they encoded historical information of past events and then use them as states. By feeding these states into a neural network their solution selects the optimal action. Their solution showed lowest overall cost compared to other methods.

In [11] the authors propose a joint load balancing and offloading solution (JSCO) in multi-server Vehicle Edge Computing (VEC) systems under latency constraints. Their system uses server load as a selection criteria in which they select and distribute offloaded tasks on available servers in order to balance the load and maximize system utility. They formulate the problem as a mixed integer nonlinear programming problem where by decoupling it as two sub-problems they developed a lower complexity solution. Their results showed fast convergence and performance compared to benchmark solutions.

The limitations of works presented in Section II are summarized in table 1 where we show a research gap which this work is designed to cover.

TABLE 1. Related works comparison.

	Inference Model Scheduling			Edge Server Selection		
	Energy Constraint	Time Constraint	Multi-Model	Network Latency	Edge device energy	Server Capacity
[8]	✓	x	x	x	x	x
[6]	x	✓	✓	x	x	x
[5]	✓	✓	✓	x	x	x
[9]	x	x	x	✓	✓	✓
[10]	x	x	x	✓	x	✓
[11]	x	x	x	✓	x	✓
LITOSS	✓	✓	✓	✓	✓	✓

III. SYSTEM MODEL

We consider system where an edge node is equipped with a set of local inference models $M_{local} = \{1, \ldots, m_l\}$ as depicted in Fig 2. Additionally, each edge node has access to a set

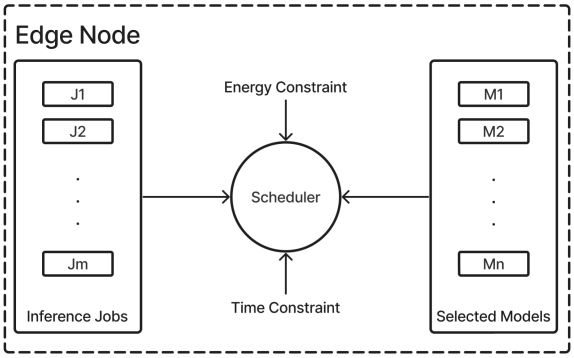

FIGURE 2. Inference model scheduling for given inference tasks using the selected models.

of edge servers where each server is equipped with a single inference model denoted as $M_{server} = \{1, \ldots, m_s\}$. Using M_{local} and M_{server}, every edge node can construct a set of selected inference models $M = \{1, \ldots, m\}$. Finally, for each time slot t, an edge node receives a set of inference tasks denoted as $J = \{1, \ldots, n\}$.

A. INFERENCE ACCURACY

Local inference models deployed in edge nodes can take several forms. One option is to use a single tunable model, where adjusting hyperparameters can change both the accuracy and inference times. Another option is to deploy multiple instances of a similar type of inference models with different internal structures, such as varying layer sizes in the case of Deep Neural Networks (DNNs). Alternatively, a variety of models with different sizes and top-1 average accuracies can be used to cater to diverse inference tasks.

The actual top-1 accuracy of each model for a specific inference task is unknown before performing inference, therefore, we rely on the average accuracy estimated from historical measurements. The average top-1 accuracy of model i is denoted as a_i. This approach provides a practical estimate for model performance across different tasks. It is important to note that the average top-1 accuracy of inference models deployed on edge servers is considered to be significantly higher than that of local models on edge devices. This ensures that offloading tasks to edge servers yields a substantial benefit in terms of accuracy, justifying the additional energy and time costs associated with data transmission and processing on the servers.

$$a_j > a_i \quad \forall i \in M_{local}, j \in M_{server}$$

B. INFERENCE TIME

We estimate the average inference time for each inference model i denoted as τ_i^{inf} where $i \in M$ by averaging historical inference times. Additionally, data preprocessing delay is considered to be included in τ_i^{inf}.

Similarly, we define the average communication latency of edge server i by τ_i^{lat} which is estimated from previous latency measurements. Let τ_{ij}^{off} to be the time to offload task j where $i \in J$ to edge server i. τ_{ij}^{off} can be calculated using the communication channel bandwidth and the size of task j denoted as s_j. τ_{ij}^{off} is given by:

$$\tau_{ij}^{\text{off}} = \frac{s_j}{b} + \tau_i^{\text{lat}} \tag{1}$$

where b denotes the bandwidth of the communication channel between the edge node and the edge server.

Let τ_{ij}^{task} be the total processing time of task j using model i including inference and offloading given by:

$$\tau_{ij}^{\text{task}} = \begin{cases} \tau_i^{\text{inf}} & \forall i \in M_{local} \quad j \in J \\ \tau_i^{\text{inf}} + \tau_{ij}^{\text{off}} + \tau_i^{\text{resp}} & \forall i \in M_{server} \quad j \in J \end{cases} \tag{2}$$

τ_i^{resp} represents the average response time from edge server i wich is defined as:

$$\tau_i^{\text{resp}} = \frac{s_r}{b} + \tau_i^{\text{lat}} \tag{3}$$

Let $x_{ij} = \{0, 1\}$ be a binary variable representing the decision in which the inference model i is assigned to inference task j or not. Let τ_k^{slot} as the time to process all tasks for time slot k. Since local inference and offloading are performed in parallel, we define τ_k^{slot} as the maximum of the total local inference time, denoted as ω_i, and the total server time, denoted as ϕ_i, which includes the offloading, inference, and response times for all offloaded tasks. τ_k^{slot} is given by:

$$\tau_k^{\text{slot}} = \max(\tau_k^{\text{local}}, \tau_k^{\text{server}}) \quad \forall k \in K \tag{4}$$

where

$$\tau_k^{\text{local}} = \sum_{i=1}^{m_l} \sum_{j=1}^{n} x_{ij} \tau_{ij}^{\text{inf}}$$

and

$$\tau_k^{\text{server}} = max(x_{ij}(\gamma + \tau_{ij}^{\text{inf}} + \tau_i^{\text{resp}}), \phi_k) \;\; \forall i \in M_{server}, \; j \in J \tag{5}$$

γ is given by:

$$\gamma = \sum_{i=1}^{m_s} \sum_{j=1}^{n} x_{ij} \tau_{ij}^{\text{off}} \tag{6}$$

In Fig 3 we show an example of a schedule for 6 inference tasks where 3 tasks are processed using local inference models while the remaining 3 are offloaded to 3 different edge servers. At t_0 we have:

$$\gamma = \tau_{1,4}^{\text{off}} + \tau_{2,5}^{\text{off}}$$

and

$$\tau^{\text{server}} = \gamma + \tau_{2,5}^{\text{inf}} + \tau_{2,5}^{\text{resp}}$$

then at t_1 we have:

$$\gamma = \gamma + \tau_{3,6}^{\text{off}}$$
$$\tau^{\text{server}} = max(\gamma + \tau_{3,6}^{\text{inf}} + \tau_{3,6}^{\text{resp}}, \tau^{\text{server}})$$

in which τ^{server} keeps the old value.

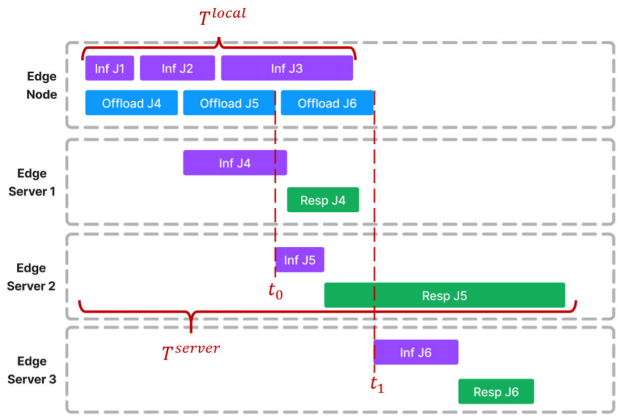

FIGURE 3. An example of a schedule for 3 edge servers and 6 inference tasks.

C. INFERENCE ENERGY

Let $e^{\text{off}}ij$ represent the energy cost of offloading task j to edge server i. This cost depends on the offload time, $\tau^{\text{off}}ij$, and ξ_i, the average energy cost of transmitting data to edge server i per unit of time. The value of ξ_i is influenced by several factors, including the communication medium (such as Wi-Fi, Cellular, Bluetooth, or Zigbee), each of which has distinct power requirements, data rates, and transmission ranges that affect energy consumption.

Other factors influencing ξ_i include the transmission power level of the wireless device, with higher power levels generally consuming more energy to maintain communication over longer distances or in environments with obstacles or interference. Additionally, the power consumed by the device in idle or standby mode, as well as signal strength and quality, impact energy consumption. Stronger signals and better quality reduce energy needs, while weak or noisy signals require higher power levels for reliable communication.

Environmental factors such as interference, obstacles, and electromagnetic noise also affect energy consumption by influencing signal propagation and reception quality. Various optimization techniques, such as data compression, packet aggregation, adaptive modulation, and power control algorithms, can help reduce energy consumption by improving spectral efficiency and minimizing transmission overhead.

We assume that ξ_i can be calculated internally by monitoring battery usage and the network adapter's configurations, such as transmission power. By averaging these measured power usage metrics, we can estimate ξ_i. Therefore, the energy cost of offloading task j to edge server i, e_{ij}^{off}, is given by:

$$e_{ij}^{\text{off}} = \frac{\tau_{ij}^{\text{off}}}{\xi_i} \tag{7}$$

Using a similar approach, we can calculate the energy cost of receiving the inference response denoted as e_i^{resp} as follows:

$$e_i^{\text{resp}} = \frac{\tau_i^{\text{resp}}}{\xi_i} \tag{8}$$

Let $e^{\text{inf}}i$ represent the average energy cost of performing an inference task using model i. This inference energy cost is relatively negligible compared to the offloading energy cost. Consequently, it is considered a constant, which can be estimated based on the inference time and the maximum power consumption of the edge device's CPU under full load conditions. Let $e^{\text{task}}ij$ denote the total energy cost of processing task j using model i, calculated as follows:

$$e_{ij}^{\text{task}} = \begin{cases} e_i^{\text{inf}} & \forall i \in M_{local} \quad j \in J \\ e_i^{\text{inf}} + e_{ij}^{\text{off}} + e_i^{\text{resp}} & \forall i \in M_{server} \quad j \in J \end{cases} \tag{9}$$

We define the total energy consumption of processing all tasks of slot k by:

$$e_k^{\text{slot}} = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} e_{ij}^{\text{task}} \leq E \tag{10}$$

D. PROBLEM FORMULATION

In this section we identify two optimization problems. First, the problem of assigning inference models to inference tasks while respecting the given time and energy constraints and maximizing the overall accuracy. Secondly, the problem of selecting an optimal subset of available edge servers which maximizes the average accuracy of produced schedules while reducing the scheduling time.

1) INFERENCE TASK SCHEDULING PROBLEM

The problem can be formulated as follows:

$$Maximize\ a_k^{\text{slot}} = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} a_i \tag{11}$$

where a_k^{slot} is the total accuracy for a time slot k. Given E and T as the energy and time constraints respectively, Equation 11 is subject to:

$$\tau_k^{\text{slot}} \leq T \quad \forall k \in K \tag{12}$$

$$e_k^{\text{slot}} \leq E \quad \forall k \in K \tag{13}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j \in J \tag{14}$$

Using Equations 12 and 13, we ensure that the parallel processing time and energy consumption for each time slot adheres to the specified time and energy constraints. Lastly, Equation 14 guarantees that each inference task is assigned an appropriate inference model, thereby producing a complete solution.

This problem can be conceptualized as a variation of the classic Knapsack Problem (KP). Here, we aim to fill our schedule (the knapsack) with inference models (the items) to maximize accuracy (profit), while adhering to time and energy constraints (knapsack weight and volume capacities). Specifically, this problem is a multi-dimensional knapsack problem (MDKP) because both the items and the knapsack have two dimensions. Additionally, since inference models (items) can be reused to construct a schedule, the problem

becomes an instance of the unbounded multi-dimensional knapsack problem (UMDKP).

However, considering that inference tasks can be processed both locally and on edge servers in parallel, this situation deviates from the UMDKP. In UMDKP terms, this parallelism allows items to overlap in the time dimension (weight) but not in the energy dimension (volume), which complicates the direct application of UMDKP.

Alternatively, if we treat each edge server as a separate knapsack, the problem aligns with the multi-knapsack problem (MKP). This approach, however, introduces significant complexity, especially when trying to enforce a global time constraint across all knapsacks.

2) EDGE SERVER SELECTION PROBLEM

Let y_j be a binary variable representing whether the inference model from edge server j is selected to be part of M.

Objective:

$$\textit{Maximize} \quad a_k^{\text{slot}} = \sum_{i=1}^{m_l} a_i + \sum_{j=1}^{m_s} a_j y_j \tag{15}$$

$$\text{Subject to :} \quad \sum_{j=1}^{m_s} y_j \leq \pi \tag{16}$$

$$y_j \in \{1, 0\}, \quad \forall j \in \mathcal{S} \tag{17}$$

where:

- a_i is the average accuracy of local inference model i.
- a_j is the accuracy of the inference model from edge server j.
- π is the maximum allowed cardinality of set M, i.e., the maximum number of edge servers that can be selected.

This formulation ensures that each selected edge server contributes its single inference model to the selected subset M. The binary variable y_j determines the selection of edge servers, and the objective function maximizes the total accuracy obtained from the selected models. Solving this optimization problem provides the optimal or near-optimal solution for selecting the subset M for each time slot k.

This type of problem is a combinatorial optimization problem known as a subset selection problem, where the goal is to select a subset of elements from a given set while optimizing a certain objective function subject to certain constraints.

In this specific case, we are tasked with selecting a subset M of inference models from both local models and models hosted on edge servers to perform inference tasks. The objective is to maximize the total accuracy obtained from the selected models while minimizing the cardinality of M (i.e., selecting the fewest number of models necessary to achieve a faster scheduling times and higher accuracy).

Such type of optimization problem can be solved using various methods such as greedy algorithms where we select models based on certain criteria (e.g., highest accuracy or best cost-benefit ratio) until the cardinality constraint is met. While greedy algorithms do not guarantee optimal solutions, they can provide fast and efficient solutions in many cases. Moreover, dynamic programming can also be used if the problem exhibits overlapping subproblems and optimal substructure properties. This approach is particularly useful for problems with small problem sizes and a limited number of feasible solutions. Metaheuristic algorithms such as genetic algorithms, simulated annealing, or particle swarm optimization can be used to explore the solution space and find near-optimal solutions. These algorithms are suitable for complex optimization problems with large solution spaces and non-linear objective functions.

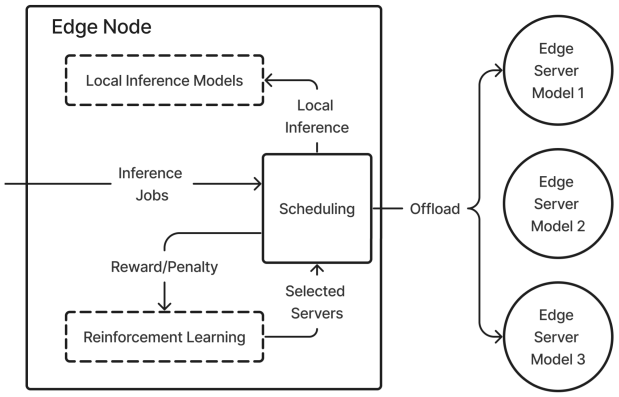

FIGURE 4. LITOSS architecture.

IV. A LIGHTWEIGHT INFERENCE TASK OFFLOADING AND SERVER SELECTION FRAMEWORK

In this section we propose LITOSS a framework for efficiently scheduling inference tasks between local inference and edge server offloading. As depicted in Fig 4 LITOSS consists of two main components. First, is the reinforcement learning-based server selection module responsible for providing the optimal subset of edge servers to second module denoted as the scheduler. It assigns inference models to inference tasks while adhering to the given time and energy constraints. The scheduler provides feedback to the server selection module to improve the selection process based on the obtained total accuracy and constraint compliance parameters.

A. INFERENCE TASK SCHEDULING

In this section, we present a Lightweight Inference Task Scheduling (LITS) scheme based on genetic algorithms. The main steps of LITS are shown in Algorithm 1.

1) POPULATION INITIALIZATION

Let h^k be a schedule containing the set of assigned inference models to inference tasks at a time slot k. $h^k = \{m_1, m_2, \ldots, m_J\}$. Let $\mathcal{H}$ be the set of schedules representing the population of solutions given by $\mathcal{H} = \{h_1^k, h_2^k, \ldots, h_p^k\}$ where p is the population size.

Algorithm 1 Main Steps of LITS

1: Initialize population
2: Evaluate fitness and rank population
3: If termination criteria satisfied go to Step 7
4: Explore neighborhood of the best solution
5: Produce new generation using Tournament Selection, Crossover and Mutation
6: Go to Step 2
7: Return the best solution

The initial population is generated randomly where each schedule is assigned random inference models from the set $\mathcal{M}$ as shown in Algorithm 2.

Algorithm 2 Population Initialization

$\mathcal{H} \leftarrow \{\}$
for $i = 1$ to p:
    $h_i^k \leftarrow \{\}$
    for $j = 1$ to J:
        $h_i^k \leftarrow h_i^k + \{Random(\mathcal{M})\}$
    $\mathcal{H} \leftarrow \mathcal{H} + \{h_i^k\}$

2) FITNESS EVALUATION AND RANKING

We define the fitness function f_k for time slot k as the total accuracy A_k multiplied by coefficients α_k and β_k added to penalize overshooting the target time and energy constraints respectively.

$$f_k = a_k^{\text{slot}} \alpha_k \beta_k \tag{18}$$

where α_k and β_k are given by:

$$\alpha_k = \min(1, \frac{T}{\tau_k^{\text{slot}}}) \tag{19}$$

$$\beta_k = \min(1, \frac{E}{e_k^{\text{slot}}}) \tag{20}$$

T and E are the time and energy constraints respectively. Using the above formulas results in $0 \leq \alpha_k \leq 1$ and $0 \leq \beta_k \leq 1$ which ensures that we do not reward undershooting the constraints.

After evaluating the fitness values for the current generation of solutions, the population is sorted in descending order according the fitness values.

3) NEIGHBORHOOD EXPLORATION

We define the steps of neighborhood exploration in Algorithm 3. The algorithm takes a schedule h_k and performs a walk distance w for each assigned inference model i starting from $h_k(i) - w$ to $h_k(i) + w$ respecting index limits of inference models. By comparing the fitness values of each neighbor f_k^{neighbor} with f_k and then storing them in the set B if they have a better fitness value. Otherwise the inference model index $f_k^{\text{neighbor}}(i)$ is returned to original value.

Algorithm 3 Neighborhood Exploration

GetBestNeighbors(h_k, f_k, w)
$B \leftarrow \{\}$
$h_k^{\text{neighbor}} = h_k$
for $i = 0$ to $\mathcal{J}$:
    for $j = \max(0, h_k(i) - w)$ to $\min(h_k(i) + w, \mathcal{M} - 1) - 1$:
        if $(j = h_k(i))$
            continue
        $h^{\text{neighbor}}(i) \leftarrow j$
        if $f_k^{\text{neighbor}} > f_k$:
            $B \leftarrow B + h^{\text{neighbor}}$
        else
            $h^{\text{neighbor}}(i) \leftarrow h_k(i)$

Algorithm 4 Reproduction Process

$G \leftarrow \{\}$
for $i = 0$ to p:
    $p_1 = TournamentSelection(\mathcal{H})$
    $p_2 = TournamentSelection(\mathcal{H})$
    $(o_1, o_2) = Crossover(p_1, p_2)$
    if $(\mu > 0)$:
        $Mutate(o_1)$
        $Mutate(o_2)$
    if $(fitness(o_1) > 0)$:
        $G = G + \{o_1\}$
    if $(fitness(o_2) > 0)$:
        $G = G + \{o_2\}$
$\mathcal{H} \leftarrow G$
$\mu = \mu \times \sigma$

4) REPRODUCTION PROCESS

The next generation is create by performing tournaments to select parents. A tournament of a predetermined size is selected randomly from the sorted population. Then the elements of the tournament are compared against each other using their fitness values to pick the best schedule. This process is repeated twice to obtain two parents which are then crossed over using a discrete uniform approach where genes of the offspring are chosen randomly from each parent with equal probability.

The newly generated offspring are subjected to probabilistic mutations to maintain population diversity and prevent convergence to local optima. A mutation probability parameter μ controls whether an offspring undergoes mutation. Higher mutation probabilities increase random changes, aiding exploration but risking disruption of promising solutions. Lower probabilities slow exploration but help preserve promising solutions.

A fading parameter σ is introduced to adjust the mutation probability dynamically throughout the evolution process. This involves reducing the mutation rate as the algorithm progresses through generations. The strategy aims to promote exploration initially with a higher mutation rate, gradually

decreasing it to allow for exploitation and refinement of promising solutions. The fading mutation probability strategy balances exploration and exploitation. Higher mutation rates early in optimization aid in exploring diverse solution spaces, while reducing rates as the algorithm converges allows for fine-tuning around more promising regions. The steps of the reproduction process are presented in Algorithm 4.

5) TERMINATION CRITERIA

The convergence of the evolutionary process can be monitored by comparing the last n top-1 solutions from previous generations. If reproduction fails to yield any better top-1 ranking solutions, the process is terminated.

B. EDGE SERVER SELECTION

The server selection module uses reinforcement learning for optimizing the subset of selected servers using rewards and penalties from the scheduling module. In this section we start by modeling the selection process as a Markov Decision Process (MDP) and employ State–action–reward–state–action (SARSA) as method for learning the policy [12]. SARSA is preferable over more advanced algorithms such as DQN, for constrained edge devices running time-sensitive real-time applications primarily due to its lightweight nature and faster execution speed. Unlike DQN, which relies on deep neural networks requiring significant computational resources and time for training and inference, SARSA uses simpler tabular methods or linear function approximations that are computationally less intensive. This makes SARSA much faster, reducing latency, which is crucial for real-time applications. Additionally, SARSA's on-policy learning approach allows for more stable and predictable performance in dynamic environments, further enhancing its suitability for edge devices with limited processing power and strict time and energy constraints.

1) MARKOV DECISION PROCESS (MDP)

The decision-making in an MDP involves an agent interacting with an external environment over discrete steps. At each step, the agent evaluates the current environment state and chooses an action. The environment then transitions to a new state in response to the agent's action, while the agent receives a reward from the environment. This process repeats iteratively until task completion or a termination condition. The agent's main goal in the MDP is to optimize its decision-making to maximize cumulative rewards over time. By selecting actions leading to favorable outcomes and accruing rewards, the agent aims for the highest total reward [13].

In the context of the problem at hand, the agent, representing the server selection module, maintains a state that encapsulates relevant information about the environment. Each time the agent selects a subset of servers as an action, it triggers a transition to a new state, which reflects the updated status of the environment after the selection has been made. Additionally, upon taking an action, the agent receives a reward from the environment, in our case represented by the scheduler, which serves as feedback based on the effectiveness of the server selection.

In the following sections, we describe the state space, action space, and reward function of the MDP model.

2) STATE SPACE

Each state s is represented as a binary vector $s = [s_1, s_2, \ldots, s_n]$, where s_i denotes whether edge server i is selected (1) or not selected (0) for offloading tasks.

3) ACTION SPACE

The actions a correspond to selecting subsets of edge servers for offloading. Each action involves choosing which subset of edge servers to select. Actions are represented as a binary vector $a = [a_1, a_2, \ldots, a_n]$, similar to states, where each element a_i indicates whether edge server i is selected (1) or not selected (0).

4) REWARD FUNCTION

The reward function combines the objectives of maximizing the average accuracy and minimizing latency. Let A_k be the average accuracy of the selected schedule of time slot k. Let d_k be the number of selected but unused inference models for time slot k. We define the reward function R_k as:

$$R_k = a_k^{\text{slot}} \alpha_k \beta_k \varphi_k \tag{21}$$

where α_k and β_k are the penalties of surpassing the time and energy constraints respectively, while φ_k is a reward-only coefficient for selecting the right set of edge servers that were all used by the scheduler.

$$\varphi_k = \max(1 + \frac{1-d}{10}, 1) \tag{22}$$

5) POLICY

A policy is defined as a strategy that controls the agent's decision-making process. Specifically, it defines how the agent selects actions in different states to maximize its cumulative reward over time.

To learn an optimal policy we use SARSA (State-Action-Reward-State-Action) which is considered an on-policy learning method meaning it learns the value of state-action pairs while following a specific policy. It aims to learn Q-Values representing state-action pairs, denoted as $Q(s, a)$, where s is the current state and a is the action taken in that state. These Q-values represent the expected cumulative reward an agent can obtain by taking action a in state s and then following a specific policy to choose subsequent actions. SARSA iteratively evaluates and improves its policy by updating Q-values based on observed transitions. It follows an ϵ-greedy policy, where with probability ϵ, the agent chooses a random action (exploration), and with probability $1 - \epsilon$, it chooses the action with the highest Q-value (exploitation). Initially, Q-values are initialized arbitrarily or using heuristics.

At each time slot k, the agent observes the current state s_k, selects an action based on the ϵ-greedy policy, and executes

the action. After taking action, the agent observes the resulting reward R_k and transitions to the next state s_{k+1}. The agent then selects the next action a_{k+1} based on the ϵ-greedy policy applied to the new state s_{k+1}. SARSA updates the Q-value of the previous state-action pair (s_k, a_k) using the observed reward and the Q-value of the next state-action pair (s_{k+1}, a_{k+1}). [12] The Q-value update follows the SARSA update rule given by:

$$Q(s_k, q_k) \leftarrow Q(s_k, a_k) + \alpha[R_k + \gamma Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k)] \quad (23)$$

where α is the learning rate and γ is the discount factor.

SARSA continues to interact with the environment, updating Q-values and improving its policy, until a termination condition is met (e.g., a maximum number of episodes or convergence). It gradually reduces the exploration rate ϵ over time with a decay coefficient ρ to shift from exploration to exploitation, allowing the agent to exploit the learned policy more as training progresses. With sufficient exploration and learning, SARSA converges to an optimal policy that maximizes cumulative rewards over time, taking into account the trade-off between exploration and exploitation.

6) EXPERIENCE REPLAY

Instead of updating Q-values immediately after each transition, we use experience replay in which the agent stores experiences in the replay memory and updates Q-values using sampled mini-batches from the replay memory. Experience replay helps stabilize training by breaking correlations between consecutive experiences and preventing the agent from being biased towards recent experiences. It improves learning efficiency by reusing past experiences and facilitating learning from a diverse set of experiences [14]. Let D be a replay memory buffer to store experiences encountered by the agent during interactions with the environment. Each experience $e_k = (s_k, a_k, R_k, s_{k+1}, a_{k+1})$ consisting of the current state s_k, the action taken a_k in the current state, the reward R_k received after taking action a_k, the next state s_{k+1} after taking action a_k, and finally the action a_{k+1} taken in the next state. During training, we sample mini-batches of experiences B from the replay memory D to update the Q-values.

The steps for learning an optimal policy using SARSA are shown in Fig 5.

C. DESIGN OF LITOSS

At each time slot k, LITOSS receives a set of inference tasks J_k. Initially, the server selection agent selects a random subset from the available set of edge servers M_{server}. Using the selected edge servers and the local inference models, a set M of selected inference models is constructed. The scheduler assigns inference models to the given inference tasks and calculates the reward R_k using the average accuracy a_k^{slot} and the total time τ^{slot_k}. This reward is used to update the edge server selection agent. The main steps of LITOSS are shown in Algorithm 5.

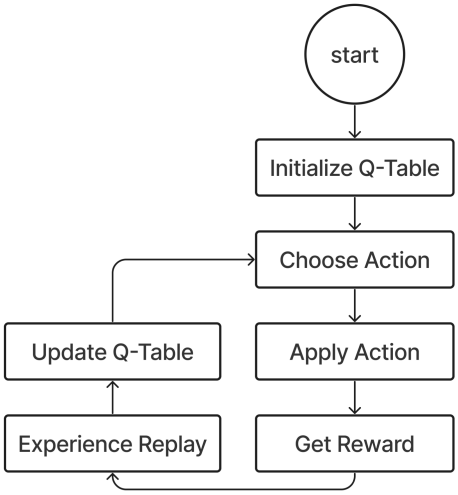

FIGURE 5. Steps of learning an optimal policy using reinforcement learning.

Algorithm 5 Main Steps of LITOSS

```
Initialize Server Selection
for each time slot k:
    M_k ← M_local
    M_k ← M_k + ServerSelection(M_server)
    h_k ← ScheduleTasks(J_k, M_k)
    Update(Reward(h_k))
    if M_server changes then ResetServerSelection()
```

V. EXPERIMENTS AND RESULTS

In this section we present our experiment setup followed by the obtained results.

A. SETUP CONFIGURATION

Experiments are conducted using a Raspberry 4 representing an edge device connected using a WiFi access point to a set of 10 edge servers consisting of a combination of laptops and desktop computers.

The experiment scenario is an object classification application performed on a stream of images obtained from the ImageNet-mini dataset [15].

A set of lightweight object classification inference models are chosen as local models deployed in the raspberry pi consisting of ResNet-18 and ResNet-34 [16], in addition to ShuffleNet-V2 [17]. The edge servers are equipped with a more accurate and larger inference model, specifically the ResNeXt-101 [18]. During the deployment phase we perform tests on these models to estimate the average inference time and accuracy of each inference model on each machine (see Table 2). Note that the average inference time of edge servers (we experimented with 10 servers) varies from machine to machine depending on their hardware capabilities and therefore is omitted from the table.

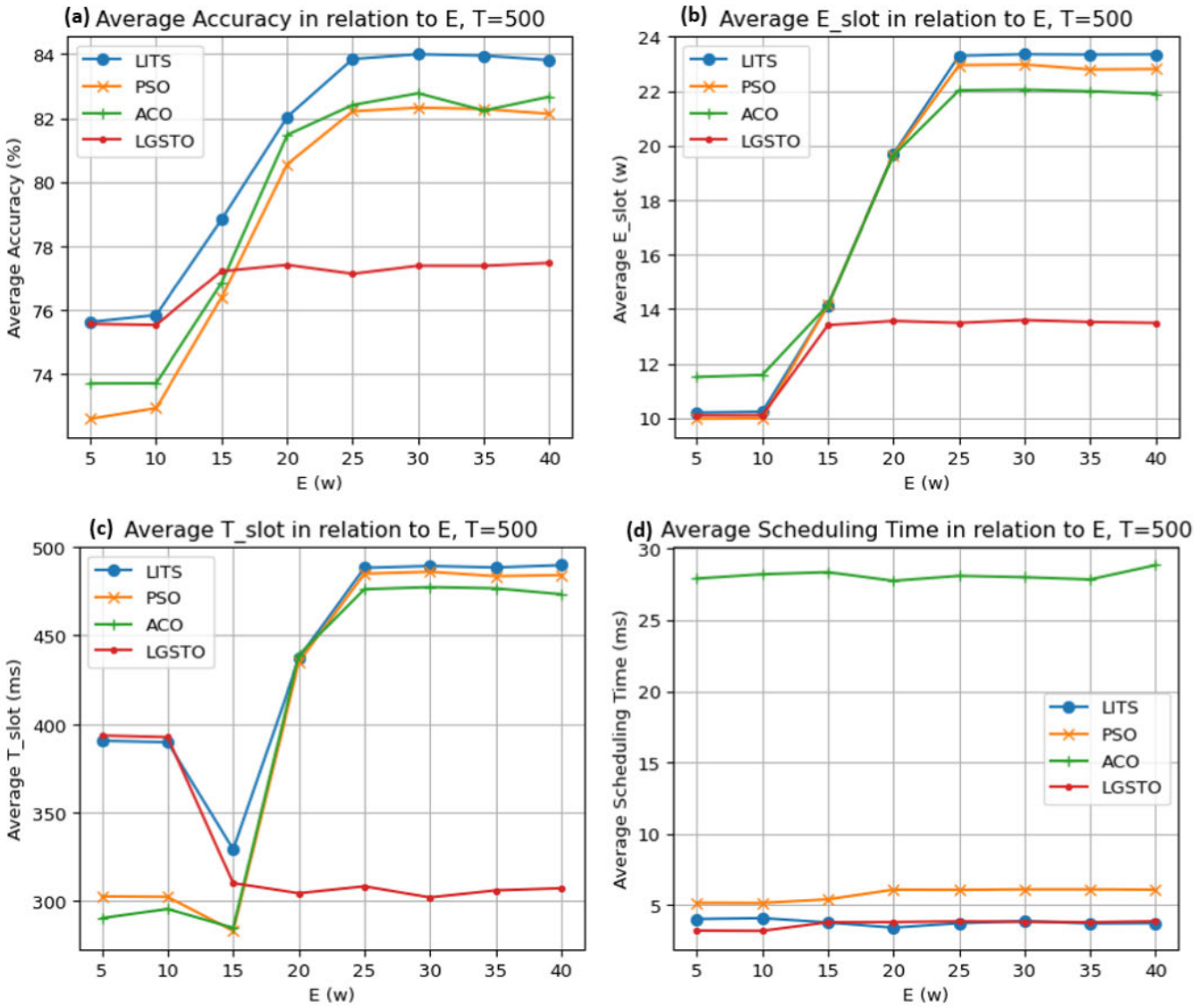

FIGURE 6. Comparison of LITS, PSO, ACO and LGSTO with varying energy constraint.

TABLE 2. Models average accuracies and inference times.

	Average Accuracy (%)	Average Inference Time (ms)
ShuffleNet-V2	66.15	19.44
ResNet-18	72.01	28.07
ResNet-34	76.79	42.45
ResNeXt-101 (Edge Servers)	87.05	-

TABLE 3. LITS parameters.

p	50
μ	0.35
σ	0.95
w	3
Termination Count	5
Generations Count	30
Tournament Size	10

B. EXPERIMENT PARAMETERS

In this section we provide implementation parameters of LITOSS. Table 3 and Table 4 contain parameters for inference task scheduling and edge server selection modules respectively. These parameters are fine-tuned using automated scripts.

C. IMPLEMENTATION OF COMPARABLE SCHEMES

We compare our results against LGSTO [5] and other meta-heuristic schemes such as Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO). The configuration parameters for each scheme are presented in 6. LGSTO and GA are configured similar to LITS.

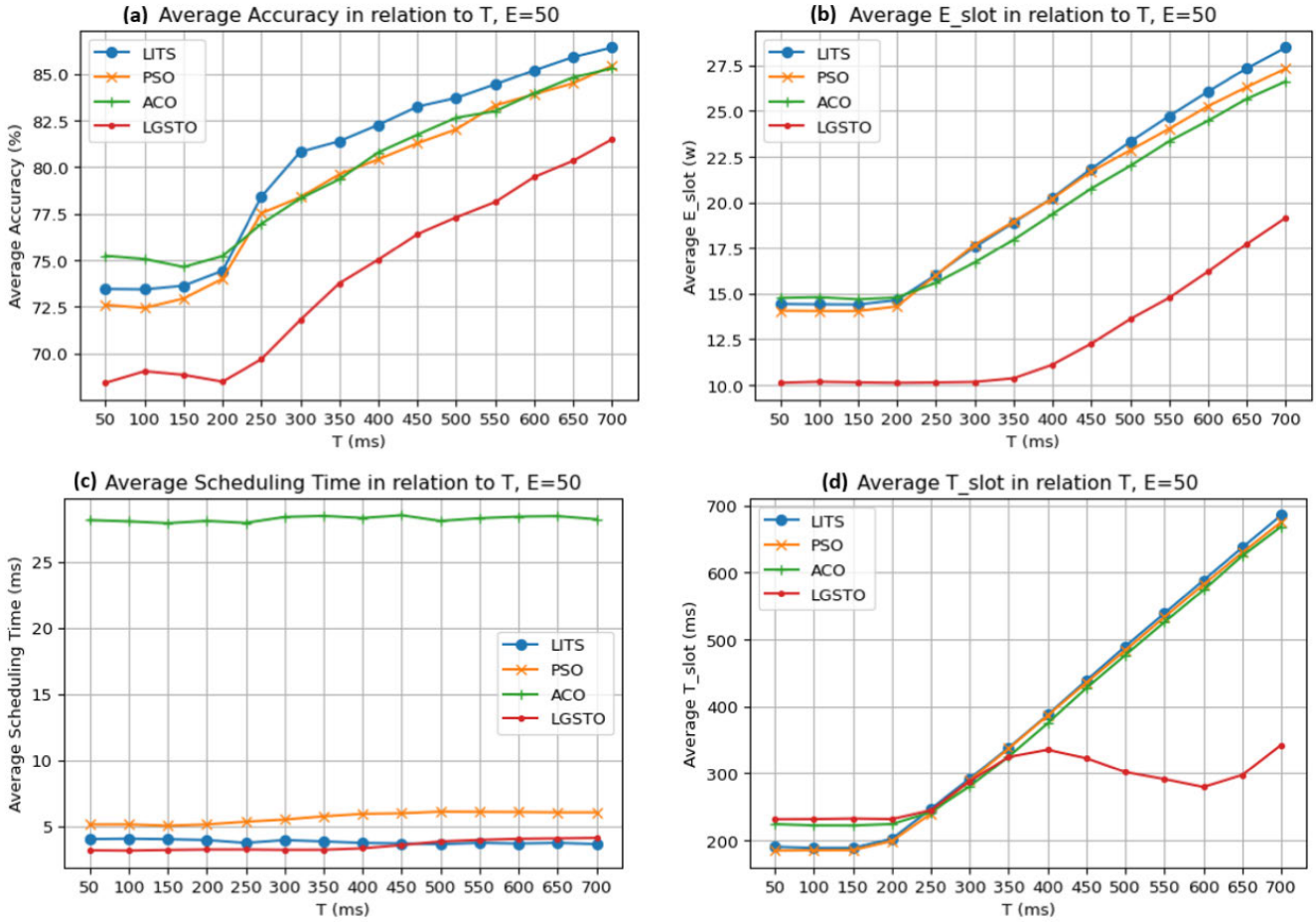

FIGURE 7. Comparison of LITS, PSO, ACO and LGSTO with varying time constraint.

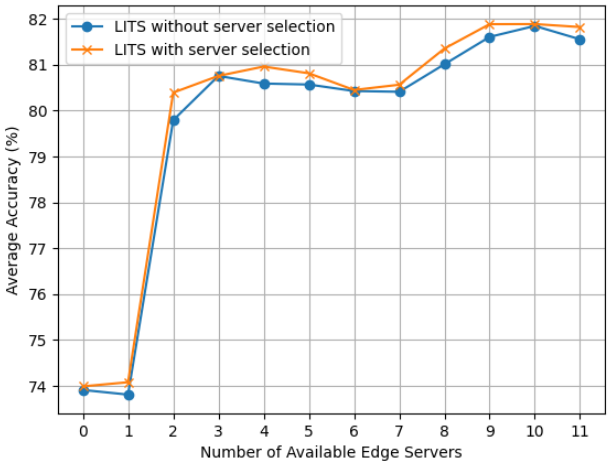

FIGURE 8. Accuracy comparison of LITS with and without edge server selection.

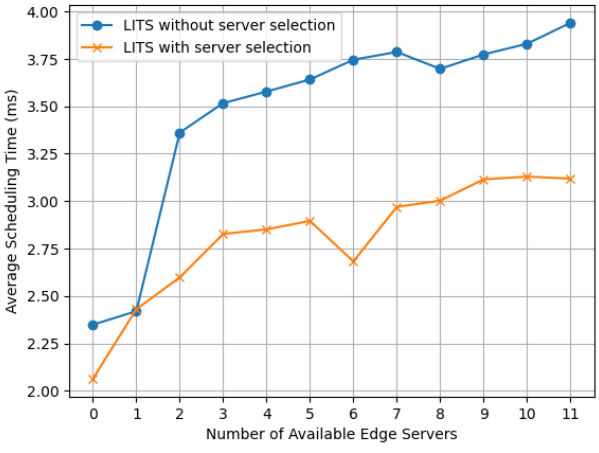

FIGURE 9. Scheduling time comparison of LITS with and without edge server selection.

D. RESULTS

In this section we present results of the experiments starting with evaluating LITS (i.e. the scheduler) on its own compared to other metaheuristic schemes. Followed by the evaluation of the edge server selection module. We consider the average of τ^{slot}, e^{slot}, a^{slot} and scheduling times calculated over all time slots of each test run as metrics.

To evaluate LITS we first perform experiments while varying energy constraint from 5 to 40 w while keeping the

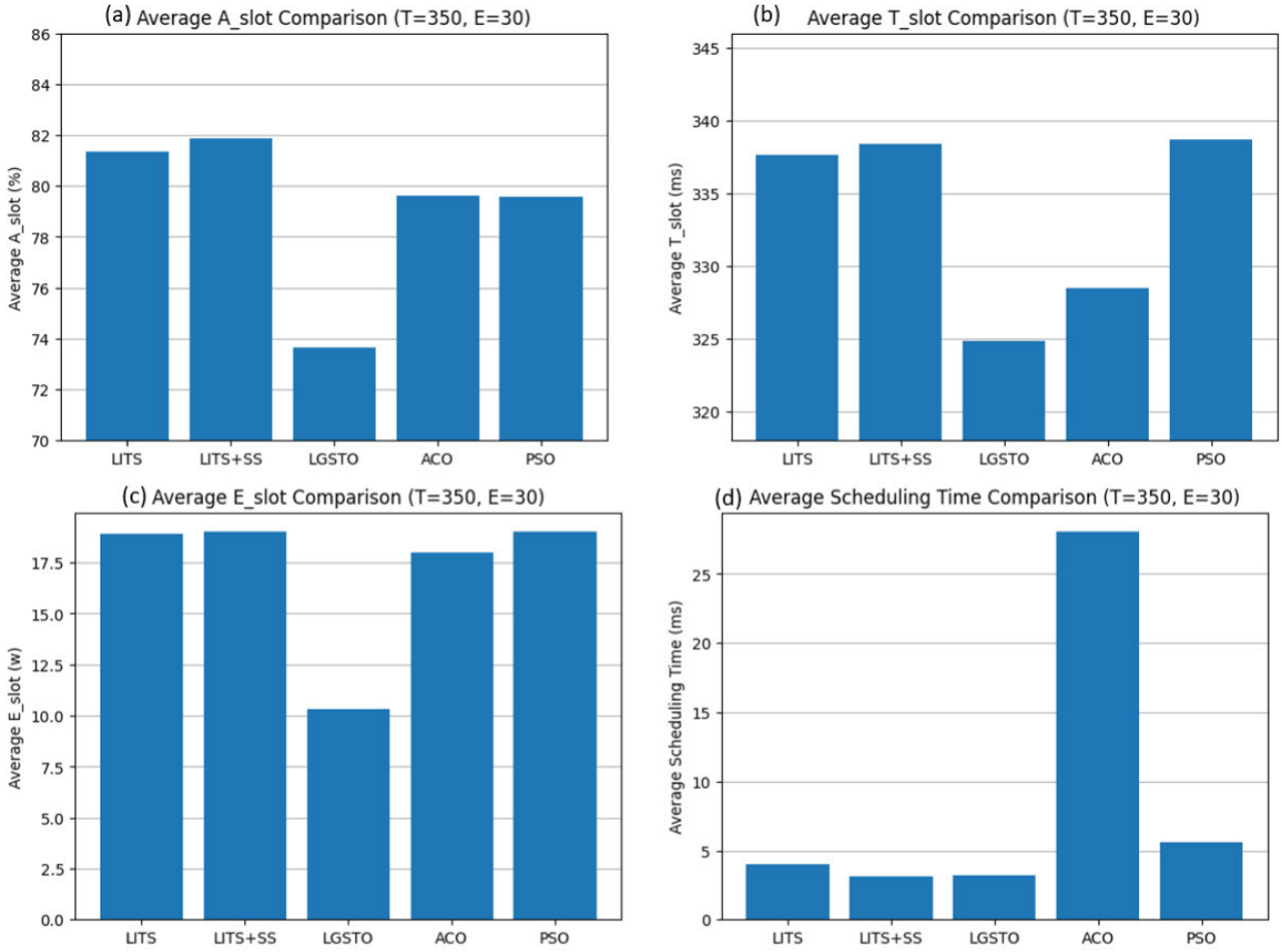

FIGURE 10. Performance comparison of all schemes.

TABLE 4. SARSA parameters.

α	0.1
γ	0.9
ϵ	1.0
ρ	0.99
ρ_{min}	0.01
D	1000
B	32

TABLE 5. Experiment parameters.

Inference Tasks Count	3923
Tasks Per Time Slot	10

TABLE 6. Parameters of comparable schemes.

PSO	Swarm Size	100
	Max Iterations	100
ACO	Ants	200
	Evaporation Rate	0.1

time constraint set to 500ms as depicted in Fig 6. Subplot (a) shows that LITS is producing schedules with higher average accuracy compared to ACO and PSO. LGSTO, however, showed very low average accuracy due to the fact that it only generates sequential schedules and does not take advantage of parallel offloading. This also affects LGSTO in the average e^{slot} and τ^{slot} depicted in subplot (b) and (c) respectively. Looking at subplot (b) we see that all schemes produce schedules averaging 10w while the constraint is set to 5w due to the fact that 10 is the lowest power a scheme can have and no solutions which satisfy that constraint exist. After that all schemes scale linearly with the increase of the power constraint until 25 in which all schemes stay at below 24w due to the limitation shifting from the power constraint to the time constraint. In subplot (c) we see at the start that LGSTO and LITS take advantage of the available time constraint to select slower but higher accuarcy models which results in higher average accuracy as shown in the start of subplot (a), while other schemes fail to do so. Subplot (d) shows the scheduling time where LGSTO and LITS have the lowest scheduling times at less than 5ms however LITS produces much higher accuracy schedules.

TABLE 7. Performance comparison of all schemes.

	Average Accuracy (%)	Average T^{slot} (ms)	Average E^{slot} (w)	Average Scheduling time (ms)
LITS	81.35	337.64	18.90	3.98
LITOSS	81.86	338.38	18.99	3.16
LGSTO	73.63	324.84	10.34	3.17
ACO	79.60	328.46	18.02	28.03
PSO	79.59	338.67	19.00	5.57

Similarly, we perform the same experiments but this time varying the time constraint to observe how the scheduling schemes scale with different constraint values. Looking at Fig 7 we see a similar trend where LITS is producing schedules at higher average accuracy. At the beginning we see that ACO is producing slightly higher accuracy than LITS this is as a result of ACO not respecting the time constraint as shown in plot (d). On the other hand, LGSTO is producing very low average accuracy as a result of not leveragin parallel offloading. Subplots (b) and (d) show that most scheme were unable to satisfy the time constraints below 200ms and only found solutions at around 200ms which the minimum average time for a schedule in this experiment. Above 200ms we see that both e^{slot} and τ^{slot} scale linearly with the increase of time constraint.

To evaluate the edge server selection module we perform the same experiments as before while varying the number of available edge servers from 0 to 10. In terms of accuracy as shown in Figure 8 we see no reduction in average accuracy while using server selection compared to not using it. Whereas in Figure 9 we see an improvement in scheduling times as a result of selecting a fewer number of edge servers for the scheduler to work with which in turn reduces the dimensions of the scheduling problem at hand and thus lower the runtime.

Finally, we compare all the schemes together as shown in Figure 10 and Table 7. Subplot (a) shows that LITS with server selection has the highest average accuracy compared to other schemes even LITS without server selection while producing these schedules at the lowest scheduling times as shown in subplot (d).

VI. CONCLUSION

In this work, we proposed an inference task scheduling and offloading framework for edge computing under time and energy constraints. Using a lightweight genetic algorithm based scheduling scheme we showed that the complex problem of scheduling inference tasks across local models and edge server models in parallel can be solved efficiently under given time and energy constraints. Additionally, we introduced an enforcement learning based server selection agent to help reduce the number of available edge servers and improve the speed and accuracy of the scheduling method. Experiments performed using the ImageNet-Mini dataset showed that our framework is lightweight enough to run on a raspberry pi and perform real-time inference task scheduling under the given time and energy constraints while producing higher average accuracy compared to other schemes.

Finally, we identify three limitations in this work. First, we did not develop a robust method for estimating the energy cost of data transmission over unreliable wireless communication channels. Second, in task offloading applications, server load is a critical parameter for server selection, yet we lack a distributed scheme for accurately estimating server load. Lastly, employing a deep reinforcement learning approach for server selection could incorporate a broader range of state parameters, enhancing decision-making. This approach is being considered for future research.

REFERENCES

[1] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Comput. Surveys*, vol. 54, no. 8, pp. 1–37, Nov. 2022.
[2] N. Abdenacer, N. N. Abdelkader, A. Qammar, F. Shi, H. Ning, and S. Dhelim, "Task offloading for smart glasses in healthcare: Enhancing detection of elevated body temperature," in *Proc. IEEE Int. Conf. Smart Internet Things (SmartIoT)*, Aug. 2023, pp. 243–250.
[3] A. Ben Sada, A. Naouri, A. Khelloufi, S. Dhelim, and H. Ning, "A context-aware edge computing framework for smart Internet of Things," *Future Internet*, vol. 15, no. 5, p. 154, Apr. 2023. [Online]. Available: https://www.mdpi.com/1999-5903/15/5/154
[4] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, and S. Bhosale, "Llama 2: Open foundation and fine-tuned chat models," 2023, *arXiv:2307.09288*.
[5] A. B. Sada, A. Khelloufi, A. Naouri, H. Ning, and S. Dhelim, "Selective task offloading for maximum inference accuracy and energy efficient real-time IoT sensing systems," 2024, *arXiv:2402.16904*.
[6] A. Fresa and J. Prakash Champati, "Offloading algorithms for maximizing inference accuracy on edge device under a time constraint," 2021, *arXiv:2112.11413*.
[7] V. Cacchiani, M. Iori, A. Locatelli, and S. Martello, "Knapsack problems—An overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problems," *Comput. Oper. Res.*, vol. 143, Jul. 2022, Art. no. 105693.
[8] I. Nikoloska and N. Zlatanov, "Data selection scheme for energy efficient supervised learning at IoT nodes," *IEEE Commun. Lett.*, vol. 25, no. 3, pp. 859–863, Mar. 2021.
[9] T. M. Ho and K.-K. Nguyen, "Joint server selection, cooperative offloading and handover in multi-access edge computing wireless network: A deep reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 21, no. 7, pp. 2421–2435, Jul. 2022.
[10] H. Liu and G. Cao, "Deep reinforcement learning-based server selection for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13351–13363, Dec. 2021.
[11] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4377–4387, Jun. 2019.
[12] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA," *IEEE Access*, vol. 8, pp. 54074–54084, 2020.
[13] G. Yang, L. Hou, X. He, D. He, S. Chan, and M. Guizani, "Offloading time optimization via Markov decision process in mobile-edge computing," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2483–2493, Feb. 2021.
[14] M. Tang and V. W. S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.
[15] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," 2017, *arXiv:1606.04080*.
[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.
[17] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet v2: Practical guidelines for efficient CNN architecture design," 2018, *arXiv:1807.11164*.
[18] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," 2017, *arXiv:1611.05431*.

ABDELKARIM BEN SADA received the B.Sc. degree in computer science from the University of Djelfa, Algeria, in 2014, and the M.Sc. degree in networking and distributed systems from the University of Laghouat, Algeria, in 2016. He is currently pursuing the Ph.D. degree with the University of Science and Technology Beijing, China. His research interests include computer vision, machine learning, and the Internet of Things.

AMAR KHELLOUFI received the B.S. degree (Hons.) in computer science from the Faculty of Sciences and Technology, Ziane Achour University of Djelfa, Djelfa, Algeria, in 2012, and the M.S. degree in distributed information systems from the Faculty of Sciences, University of Boumerdès, Boumerdes, Algeria, in 2014. He is currently pursuing the Ph.D. degree with the School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China. His current research interests include the Internet of Things, blockchain applications, edge computing, and distributed systems.

ABDENACER NAOURI received the B.S. degree in computer science from the University of Djelfa, Algeria, in 2011, and the M.Sc. degree in networking and distributed systems from the University of Laghouat, Laghouat, Algeria, in 2016. He is currently pursuing the Ph.D. degree with the University of Science and Technology Beijing, Beijing, China. His current research interests include cloud computing, smart communications, machine learning, the Internet of Vehicles, and the Internet of Things.

HUANSHENG NING (Senior Member, IEEE) received the B.S. degree from Anhui University, Hefei, China, in 1996, and the Ph.D. degree from Beihang University, Beijing, China, in 2001. He is currently a Professor with the School of Computer and Communication Engineering, University of Science and Technology Beijing, China, and the Founder and a Principal with Beijing Cyberspace International Science and Technology Cooperation Base. His current research interests include the IoT, general cyberspace and metaverse, smart education, cyber-syndrome, and cyber-health.

SAHRAOUI DHELIM received the master's degree in networking and distributed systems from the University of Laghouat, Algeria, in 2014, and the Ph.D. degree in computer science and technology from the University of Science and Technology Beijing, China, in 2020. He was a Visiting Researcher with Ulster University, U.K., from 2020 to 2021. He is currently a Senior Postdoctoral Researcher with University College Dublin, Ireland. His research interests include social computing, smart agriculture, deep-learning, recommendation systems, and intelligent transportation systems. He serves as a Guest Editor for several reputable journals, including *Electronics* journal and *Applied Science Journal*.