Graphical Abstract

NOTE and T-NOTE: Efficient Transformer-Based Encoders for IoT Task Offloading in Edge-Fog-Cloud Computing

Arthur Garon, Sonia Yassa, Lylia Alouache

Highlights

NOTE and T-NOTE: Efficient Transformer-Based Encoders for IoT Task Offloading in Edge-Fog-Cloud Computing

Arthur Garon, Sonia Yassa, Lylia Alouache

- Novel Transformer-based architectures (NOTE and T-NOTE) for IoT task offloading using Deep Q-Learning in edge-fog-cloud environments

- T-NOTE achieves zero task failures with 16.6% latency reduction and 6.4% energy savings compared to greedy baseline on real-world IoT dataset

- Comprehensive evaluation framework with 30,000 real tasks from Pakistan, publicly available with open-source implementation

- Comparative analysis of DQL and multi-objective genetic algorithms (NPGA, NSGA-II) reveals trade-offs in convergence speed versus solution diversity

NOTE and T-NOTE: Efficient Transformer-Based Encoders for IoT Task Offloading in Edge-Fog-Cloud Computing

Arthur Garon, Sonia Yassa, Lylia Alouache[1,1,1]

[a]*CY Cergy Paris Université, ENSEA, ETIS, , Cergy, 95000, Île-de-France, France*

Abstract

The proliferation of Internet of Things (IoT) devices has created unprecedented challenges in resource management and task offloading across edge-fog-cloud computing environments. This paper addresses these challenges by proposing two novel Transformer-based architectures for IoT task offloading: NOTE (Node-Oriented Transformer Encoder) and T-NOTE (Task and Node-Oriented Transformer Encoder). Both models leverage Deep Q-Learning (DQL) to learn optimal offloading policies in dynamic, heterogeneous computing environments. NOTE focuses on node-level features such as CPU frequency, buffer capacity, and bandwidth, while T-NOTE extends this approach by incorporating task-specific attributes including size, deadline constraints, and computational requirements. Additionally, multi-objective Genetic Algorithms (NPGA and NSGA-II) were adapted for training MLP policies, providing insights into Pareto-optimal trade-offs between competing objectives. Comparative analysis demonstrates that T-NOTE achieves superior performance across all key metrics, reducing task throw rate from 0.19% to 0%, decreasing latency by 16.6%, and improving energy efficiency by 6.4% compared to the greedy baseline. One of the solution of GA-based + MLP methods successfully reached the solution identified by the DQL + MLP. This is confirming the consistency of both approaches. An open-source offloading framework, including proposed offloading strategies, based on a realistic topology and a real-world dataset of IoT tasks is proposed and made publicly available.

Keywords: Task offloading, Resource allocation, Edge-Fog-Cloud computing, Transformer architecture, Deep Q-Learning, Multi-objective optimization, Genetic algorithms, Energy, Latency, Task Throw Rate

1. Introduction

1.1. Background and motivation

The Internet of Things (IoT) has become a foundational element of modern technology, facilitating the development of innovative applications in domains such as smart cities, healthcare, autonomous systems, and other fields. The proliferation of IoT devices has been rapid and significant, with Cisco reporting a global total exceeding 30 billion (Benaboura et al.). These devices generate a substantial volume of data, approximately 2 exabytes on a daily basis. Achieving maximum potential from these systems necessitates the implementation of efficient processing and analysis methodologies. This requirement presents a formidable challenge in the domains of data management, resource allocation, and system scalability.

The inherent limitations of IoT devices, such as their small batteries, limited processing power, and minimal storage capacity, render them ill-suited to manage the substantial volumes of data they generate. It is evident that tasks such as real-time processing of sensor data or computation-intensive applications frequently exceed the capabilities of local devices. Conventional approaches entail the delegation of these tasks to centralized cloud servers. However, network limitations and latency sensitivity frequently render this approach inefficient, particularly for real-time applications (Benaboura et al.).

In order to address these challenges, fog computing has emerged as a distributed computing paradigm that extends the capabilities of cloud computing to the edge of the network. By facilitating the execution of storage, computation, and data management operations in close proximity to the data source, fog computing contributes to the reduction of latency, power consumption, and network traffic. It is evident that contemporary applications, including but not limited to smart homes, autonomous vehicles, smart agriculture, and

healthcare, are contingent on this paradigm in order to satisfy their real-time and location-aware processing requirements as evoked by Das and Inuwa (2023). Fog computing, first introduced in 2012, provides a hierarchical architecture that serves to bridge the gap between cloud servers and IoT devices, thereby allowing for seamless data flow and operational efficiency (Fahimullah et al., 2022).

Notwithstanding the advantages inherent in task offloading in fog computing, this process is encumbered by numerous challenges. Optimizing resource utilization, reducing energy consumption, and maintaining Quality of Service (QoS) require robust strategies due to the heterogeneous and dynamic nature of fog networks. Factors such as fluctuating workloads, mobility, and task diversity have been identified as contributing to an increase in the complexity of the situation. In order to address these challenges, innovative resource allocation techniques are required, including Machine Learning-based (ML) methods, auction models, and heuristic optimization as Fahimullah et al. (2022) exhibits.

Conventional resource management techniques frequently employ static heuristic approaches, which prove ineffective when confronted with the diverse and dynamic workloads that are characteristic of fog environments. They exhibit a deficiency in scalability and flexibility, which are prerequisites for real-time task offloading and resource optimization. Consequently, a substantial decline in performance is experienced as system demands increase (Iftikhar et al., 2023).

To address this problem, metaheuristics such as GA have emerged in the field of offloading as powerful optimization techniques capable of handling the multi-objective nature of fog computing task offloading. GAs excel in exploring complex solution spaces and finding near-optimal trade-offs between conflicting objectives such as latency minimization, energy efficiency, and resource utilization. Unlike traditional optimization methods that often focus on single objectives, multi-objective genetic algorithms can simultaneously optimize multiple performance criteria, making them particularly suitable for fog computing environments where various QoS parameters must be balanced. The evolutionary nature of GAs allows them to adapt to dynamic network conditions and heterogeneous resource availability, providing robust solutions for real-time task offloading decisions.

Recent advancements in the field of Artificial Intelligence (AI), particularly deep reinforcement learning (DRL), have demonstrated considerable potential in addressing the intricacies of task offloading in fog environments. Research has demonstrated the efficacy of AI-driven approaches in reducing latency, energy consumption, and operational costs. As Fahimullah et al. (2022) demonstrates in their review, techniques such as centralized Dueling Deep Q-Networks (DDQNs), decentralized learning models, and multi-agent reinforcement learning have been employed to optimize offloading policies and resource allocation. Furthermore, hybrid strategies that integrate artificial intelligence with conventional methods have demonstrated efficacy in heterogeneous fog environments (Mishra et al., 2023).

In order to maintain currency with the latest advancements in this domain, the present study investigates innovative methodologies that integrate the strengths of evolutionary computation and deep reinforcement learning for the purpose of optimizing IoT task offloading. The integration of these approaches addresses the limitations of individual techniques while leveraging their complementary advantages.

1.2. Contributions

To address these gaps, this work makes the following key contributions:

- The proposal of a two-transformers-based architecture, utilizing DQL for the purpose of task offloading within a cloud-fog-edge environment, is hereby presented. The selection of the node (edge, fog or cloud) to execute each incoming task is determined by these models, representing n different ctions. The NOTE system is oriented towards node-level features, such as CPU, buffer, and bandwidth. In contrast, the T-NOTE system incorporates additional task attributes, including size, deadline, and CPU-cycle requirements. This capability facilitates a more precise depiction of the interplay between node resources and task demands within a fog environment.

- This work also puts forth a model for offloading in hybrid environments that considers a substantial number of parameters for the QoS. To further expand upon the existing body of knowledge, an adaptation of a scenario with a real dataset was implemented within the framework of RayClousSim.

We let open-source the scenario and the dataset for the community[1]. All algorithms used, including NOTE and T-NOTE, are also implemented in this framework.

- Additionally, a comparative analysis was conducted between the GAs and a DQL approach. The implementation of these algorithms in the training of a MLP constitutes a pivotal element of the study. The GA approach is employed in a dynamic setting, thereby enabling the MLP to be optimized for multi-objective offloading in real time.

1.3. Paper organization

The reminder of this paper is structured as follows: Section 2 reviews related work; Section 3 describes the problem modeling, including the scenario and QoS modeling; Section 4 details the proposed strategies, from genetics algorithm to deep reinforcement learning approaches, including Transformers-based methods; Section 5 presents the experimental setup and results, as well as a comparative analysis of the proposed methods and that offloading analysis; finally, Section 6 concludes the paper and outlines future research directions.

2. Related Work

Task offloading in Fog/Cloud environments is a relatively recent research area compared to well-studied domains like image classification or time series forecasting. However, the decision-making process for task offloading is inherently complex due to its combinatorial nature.

2.1. Deterministic Approaches

Deterministic algorithms provide a direct method for solving the task offloading problem. For instance, the optimal task assignment algorithm proposed by Yan *et al.* Yan et al. (2020) employs a three-step approach: (i) assuming the offloading decisions are pre-determined, (ii) deriving closed-form expressions for optimal offloading, and (iii) implementing bisection search and a one-climb policy. This approach effectively reduces energy consumption and execution time for IoT tasks in Mobile Edge Computing (MEC) systems.

Nevertheless, the task offloading problem is considered NP-hard, as demonstrated in Guo et al. (2024); Jin et al. (2024); Sarkar and Kumar (2022). Consequently, deterministic methods face scalability limitations, prompting the exploration of heuristic, metaheuristic, and AI-based approaches.

2.2. Heuristic and Metaheuristic Methods

Heuristic methods are well-suited for scaling in complex edge/cloud environments, as they avoid the exponential computational growth of deterministic algorithms Zhang et al. (2024). For example, the Deadline and Priority-aware Task Offloading (DPTO) algorithm Adhikari et al. (2020) schedules tasks by prioritizing delays and task priorities, selecting optimal devices to minimize overall offloading time.

Metaheuristic algorithms, known for their adaptability, have also shown strong performance. (Bernard et al., 2024) introduced the Drafting Niched Pareto Genetic Algorithm (D-NPGA), which optimizes task offloading decisions and improves makespan and cost efficiency for IoT tasks in fog/cloud systems.

More sophisticated methods use multiple approaches. For example, Energy-Efficient and Deadline-Aware Task Scheduling in Fog Computing (ETFC) Pakmehr et al. (2024) employs a Support Vector Machine (SVM) to predict traffic on fog nodes and classify them as low- or high-traffic. Then, they use reinforcement learning (RL) on the low-traffic group and a non-dominated sorting genetic algorithm III (NSGA-III) on the high-traffic group to make the offloading decision. This allows both algorithms to perform better on adequate tasks.

Metaheuristic approaches have gained attention for their ability to address the NP-hard nature of the task offloading problem with flexibility and adaptability. A recent and comprehensive survey by Rahmani et al. (2025) reviewed a wide range of metaheuristic algorithms—including Genetic Algorithms (GA), Particle

---

[1] https://github.com/tutur90/Task-Offloading-Fog

Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Grey Wolf Optimizer (GWO)—applied to task offloading in IoT environments. The review identified key strengths of these methods in balancing latency, energy consumption, and cost efficiency across heterogeneous fog-edge-cloud infrastructures. Their taxonomy also distinguishes between evolutionary and swarm-based algorithms, showing that hybrid models are increasingly used to adapt to dynamic network conditions. Despite their strengths, the authors noted limitations in scalability and real-time adaptability—areas where AI and deep learning techniques, such as MLP or transformers under DQL paradigms, may provide an edge.

2.3. AI Approaches

Machine learning (ML) has recently demonstrated its efficacy in task offloading. For instance, decision tree classifiersSuryadevara (2021) determine whether tasks should be offloaded to the fog or cloud, resulting in reduced latency and energy consumption. Logistic regression modelsBukhari et al. (2022) have also been applied, estimating the probability of successful offloading by leveraging maximum likelihood estimation.

Deep learning (DL), a subset of ML, deserves special attention due to its notable advances across various fields.

Basic deep neural networks (DNNs) have been applied to IoT offloading tasks, such as Sarkar and Kumar (2022), where parallel DNNs optimize cost, energy consumption, and latency. However, DRL is often preferred for its decision-making capabilities in dynamic environments.

For example, Jiang et al.Jiang et al. (2021) utilized DQL, which is included in DRL techniques, to identify optimal offloading policies and resource allocation strategies for user equipment in fog/cloud systems. Their approach combines a dueling deep Q-network for model pre-processing and a distributed deep Q-network for efficient task allocation.

Moreover, multi-agent DRL methodsRen et al. (2021) have been employed to offload IoT tasks. Each IoT device trains its DRL model to select fog access points, followed by a greedy algorithm to determine cloud offloading. This approach demonstrates competitive performance in energy efficiency compared to exhaustive search and genetic algorithms.

Long Short-Term Memory (LSTM) networks, a type of recurrent neural network, are particularly suitable for the temporal dimensions of task offloading. Tu et al.Tu et al. (2022) combined LSTM with DQL to predict task dynamics in real-time, leveraging observed edge network conditions and server load. This hybrid model significantly improved latency, offloading cost, and task throughput.

Transformers, known for their revolutionary impact on natural language processing (NLP), have also been adapted for task offloading. Gholipour et al.Gholipour et al. (2023) proposed TPTO, a transformer-based framework with Proximal Policy Optimization (PPO). Their model encodes task dependencies using a transformer encoder and employs an actor-critic framework trained with PPO to generate probability distributions for offloading actions, achieving state-of-the-art results in edge computing environments.

While many existing offloading solutions rely on synthetic datasets and focus on Multi-access or Vehicular Edge Computing (MEC/VEC) scenarios Fahimullah et al. (2022); Tu et al. (2022); Gholipour et al. (2023), research specifically targeting fog computing remains limited. In particular, models such as Deep Neural Networks (DNNs) Sarkar and Kumar (2022) and Deep Q-Learning (DQL) Jiang et al. (2021) have rarely been evaluated in realistic fog environments using real-world data.

Genetic Algorithms (GAs) are often static in nature, such as Bernard et al. (2024); Pakmehr et al. (2024), meaning that they cannot be employed in real-time applications and can only converge through iterations on the same simulation environment where tasks and their order remain static. This characteristic makes them particularly complex to deploy in real-world applications where dynamic adaptation is required.

Transformer-based deep learning techniques remain poorly explored in this domain, despite being state-of-the-art in many fields. Some exploration efforts show promise; however, existing approaches often implement basic action spaces, such as TPTO Gholipour et al. (2023), which is designed with only two actions: offloading the task to the cloud or processing it on the edge.

Algorithms are referenced in the Table 1 with the method, the dataset type, the QoS evaluated, and the job type. Used QoS are referenced in the Table 2 with their description.

Paper	Method	Algorithm	Dataset	QoS	Type of Job	Environment	Tool
Yan et al. (2020)	Deterministic	Optimal Task Assignment	Real	E, ET	IoT tasks	MEC	NA
Adhikari et al. (2020)	Heuristic	DPTO	Synthetic	QWT, D	IoT tasks	Cloud & Fog	NA
Bernard et al. (2024)	Metaheuristic	D-NPGA	Synthetic	C, M	IoT tasks	Cloud & Fog	Python
This work	Metaheuristic	NPGA/NSGA-II+MLP	Real	C, L, TTR	IoT tasks	Cloud & Fog	Python, PyTorch
Pakmehr et al. (2024)	AI, Metaheuristics	ETFC	C, E, L, DLV	Synthetic	IoT tasks	Fog	NA
Bukhari et al. (2022)	AI	Logistic Regression (LR)	Real	C, E, L	IoT tasks	Cloud & Fog	Python, Matlab
Suryadevara (2021)	AI	Decision Tree (DT)	Synthetic	E, L	IoT tasks	Cloud & Fog	iFogSim
Sarkar and Kumar (2022)	AI	Deep Neural Network (DNN)	Synthetic	C, E, L	Mobile	Cloud & Fog	iFogSim
Jiang et al. (2021)	AI	DRL	Synthetic	E, L	Mobile	Cloud & Fog	Python, Adam optimizer
Ren et al. (2021)	AI	Multi-agent DRL	Synthetic	E	IoT tasks	Cloud & Fog	NA
Tu et al. (2022)	AI	DRL+LSTM	Real	C, L, TTR	Mobile	MEC	NA
Gholipour et al. (2023)	AI	Transformers PPO	Synthetic	L	IoT tasks	Edge	Python, TensorFlow
This work	AI	DQL+MLP	Real	C, L, TTR	IoT tasks	Cloud & Fog	Python, PyTorch
This work	AI	DQL+Transformers	Real	C, L, TTR	IoT tasks	Cloud & Fog	Python, PyTorch

Table 1: Summary of Task Offloading Methods in Fog and Edge Computing

Metric	Description
C: Cost	Refers to the expenses incurred in task offloading, including computation, storage, and data transfer costs.
L: Latency	The time delay between the initiation of a request and the reception of the response. Crucial for real-time applications.
E: Energy	The total energy consumed during task offloading, including device and network-level energy usage.
ET: Execution Time	The total time taken to execute a task from start to finish, including computation and communication time.
D: Delay	The time difference between task submission and the start of its processing. Includes network and processing delays.
M: Makespan	The total time required to complete all tasks in a batch or workflow. Indicates overall system efficiency.
QWT: Queue Waiting Time	The duration a task spends waiting in the queue before being processed. Impacts response time and throughput.
TTR: Task Throw Rate	The rate at which tasks are successfully processed and completed in the system, indicating system throughput.
DLV: Dead Line Violation	The percentage or rate of tasks that succeed to complete within their specified deadlines, indicating system reliability and quality of service.

Table 2: Explanation of QoS Metrics in Edge-Fog-Cloud Computing

3. Problem Modeling

3.1. Scenario Modeling

This scenario is modeled according to foundational concepts presented in Aazam et al. (2022); Bukhari et al. (2022); Jazayeri et al. (2021), following a three-tier offloading approach that involves *edge*, *fog*, and *cloud* nodes. In Figure 1, a high-level cloud-enabled architecture is shown, where a Global Gateway (GG) collects tasks from various IoT devices before determining whether to process them locally or offload them to fog or cloud resources.

A real-world dataset of IoT-generated tasks, is employed to study and evaluate these offloading decisions. The tasks in this dataset vary in size, deadline constraints, and computational complexity, thereby reflecting the heterogeneous nature of practical IoT workloads.

3.1.1. Architecture

In Figure 1, a schematic overview of the proposed three-tier architecture is illustrated. The system is composed of IoT devices (sensors, mobile nodes, and other edge devices) that generate tasks. These tasks subsequently reach the Global Gateway, which processes them locally if resources are available, or decides to offload them to the fog or the cloud based on resource availability and energy considerations.

Global Gateway. The GG layer is located at the edge and aggregates tasks from local IoT devices. In addition to routing capabilities, the GG has limited computational power, enabling it to handle smaller tasks locally and thus reduce network traffic. This approach is particularly advantageous for time-sensitive tasks that can be processed quickly on-site.

Upon receiving a task, several factors are evaluated:

- Node state: The current load on the GG, along with the availability of fog and cloud nodes.

- Network conditions: Bandwidth, potential congestion, and latency to fog or cloud nodes.

- Energy usage: A predictive assessment of energy costs if the task is executed locally versus offloaded.

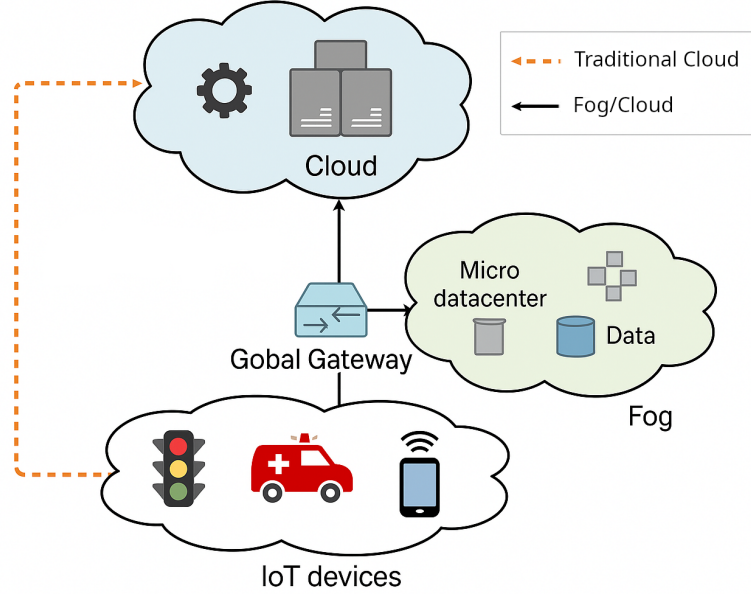- Task requirements: Size, computational complexity, and deadline or QoS constraints.

Figure 1: System Architecture

A subsequent decision is made to determine the optimal execution strategy for the task. The system evaluates three primary options: processing the task locally if it is computationally feasible and resource-efficient, offloading the task to a nearby Fog Node to leverage edge computing capabilities, or offloading the task directly to a Cloud Node for intensive computational requirements that exceed local and fog node capacities.

Fog Nodes. Fog nodes provide moderate computational resources that exceed those of edge devices but remain below the capacity of cloud data centers. These intermediate computing nodes offer lower latency than distant cloud nodes due to their closer physical proximity to the network edge. Fog nodes serve as an essential intermediate tier for tasks that cannot be processed locally at the Gateway (GG) yet do not require the extensive resources of the cloud infrastructure. This positioning makes them ideal for applications requiring balanced performance between computational capability and response time.

Cloud Nodes. Cloud nodes represent large-scale data centers with substantial computational and storage capabilities. Although they typically have higher baseline latencies because of greater network distances, they are well suited for computationally intensive applications. Cloud nodes excel in handling large tasks that demand significant CPU and memory resources, making them optimal for complex analytical workloads. They are particularly effective for batch processing scenarios with relaxed real-time constraints, where throughput is prioritized over immediate response. Furthermore, cloud data centers can accommodate high concurrency situations through dynamic resource scaling, allowing them to adapt to varying computational demands efficiently.

CPU capacity: Cloud servers can support a high volume of CPU cycles, facilitating efficient offloading for tasks with elevated computational demands.

3.2. QoS Modeling

Task offloading decisions are evaluated within the RayCloudSim framework developed by Zhang et al. (2022), a comprehensive simulation platform for modeling and assessing cloud-edge-IoT computing environments based on LEAF (Wiesner and Thamsen, 2021).

3.2.1. Task Throw Rate

In distributed environments, task execution failures can occur due to issues such as *network disconnections*, *node isolation*, or *buffer overflows*.

The *task throw rate* τ is defined as:

$$\tau = \frac{\text{Number of failed tasks}}{\text{Total tasks generated}}. \tag{1}$$

A lower throw rate is indicative of a more robust and efficient offloading strategy.

3.2.2. Latency

The latency metric for task offloading is exclusively defined for tasks that achieve successful offloading, as the underlying assumption requires that tasks both arrive at the destination and undergo computational processing. For tasks meeting this criterion, the total latency encompasses three distinct components and is formally expressed as:

$$L_{\text{total}} = L_{\text{transmission}} + L_{\text{processing}} + L_{\text{queuing}} \tag{2}$$

Conversely, for tasks that fail to achieve successful offloading, whether due to transmission failures, processing errors, or other system constraints, the total latency is assigned a null value:

$$L_{\text{total}} = 0 \tag{3}$$

Transmission Delay. Transmission delay is composed of transfer delay and propagation delay:

$$L_{\text{transmission}} = L_{\text{transfer}} + L_{\text{propagation}}. \tag{4}$$

1. Transfer Delay: The duration required to send all bits of a packet onto the transmission medium:

$$L_{\text{transfer}} = \frac{S}{B}, \tag{5}$$

where S is the data size (in bits) and B is the allocated bandwidth for the task (in bits per second).

2. Propagation Delay: The time taken for a signal to traverse the medium:

$$L_{\text{propagation}} = \frac{d}{v}, \tag{6}$$

where d is the total transmission distance (in meters) and $v \approx 2 \times 10^8 \, \text{m/s}$ in optical fiber.

Processing Delay. Processing delay is the time spent by a computing node on executing a task:

$$L_{\text{processing}} = \frac{C \cdot S}{f}, \tag{7}$$

where C is the number of CPU cycles required to process the task, and f is the CPU frequency (in cycles per second).

Queuing Delay. Queuing delay captures the waiting time a task experiences when the node is busy executing other tasks:

$$L_{\text{queuing}} = \sum_{i=1}^{N} L_{\text{processing},i}, \tag{8}$$

where N is the number of tasks that arrived before the current task, and $L_{\text{processing},i}$ is the processing time of each task in the queue.

3.2.3. Energy Consumption Model

Similar to the latency metric, energy consumption is exclusively defined for tasks that achieve successful offloading. The simulation framework categorizes energy consumption into three distinct components:

- **Idle Energy** (E_{idle}^i): Energy drawn by node i when it remains idle, estimated through an idle power coefficient P_{idle}.

- **Execution Energy** ($E_{\text{exe}}^{i,k}$): Energy consumed by node i during the execution of task k, based on execution power P_{exe}.

- **Transmission Energy** ($E_{\text{trans}}^{i,k}$): Energy utilized to transmit task k from source node j to destination node i, determined by a per-bit cost $C^{m,n}$ on the communication link.

All energy terms are expressed in Joules (J), while power (P) is measured in Watts (W). Let N denote the number of nodes and T represent the total number of tasks successfully offloaded. The overall energy consumption in the system is calculated as:

$$E_{\text{total}} = \sum_{i=1}^{N} \left(E_{\text{idle}}^i + \sum_{k=1}^{T} \left(E_{\text{exe}}^{i,k} + E_{\text{trans}}^{i,k} \right) \right) \tag{9}$$

This formulation ensures that energy consumption measurements reflect only the computational and communication activities associated with successful task offloading, maintaining consistency with the latency metric definition and providing a meaningful performance evaluation.

Node Power Consumption Model. Based on the work of Ismail and Materwala (2021), the power consumption of a computing node can be approximated by:

$$P(u_{\text{cpu}}) = \alpha + \beta \cdot u_{\text{cpu}}, \tag{10}$$

where:

- α represents the idle power (P_{idle}).

- β is the incremental power coefficient for executing tasks, defined by $\beta = (P_{\text{exe}} - P_{\text{idle}})$.

- $u_{\text{cpu}} \in [0, 1]$ is the CPU utilization ratio.

This model has a Standard Error of Estimation (SEE) of 12.9%, indicating a reasonably accurate fit to empirical data.

Idle Energy. The idle energy for node i over the entire simulation duration T can be calculated as:

$$E_{\text{idle}}^i = \int_0^T P_{\text{idle}}^i \, dt = P_{\text{idle}}^i \cdot T. \tag{11}$$

Execution Energy. Execution energy can be expressed by integrating the CPU utilization over time:

$$E_{\text{exe}}^i = \int_0^T \left(P_{\text{exe}}^i - P_{\text{idle}}^i \right) \cdot u_{\text{cpu}}(t) \, dt. \tag{12}$$

In practical simulations, full CPU utilization ($u_{\text{cpu}} = 1$) is often assumed while tasks are running, yielding:

$$E_{\text{exe}}^{i,k} = T_{\text{exe}}^{i,k} \cdot \left(P_{\text{exe}}^i - P_{\text{idle}}^i \right), \tag{13}$$

where $T_{\text{exe}}^{i,k}$ represents the execution time of task k on node i.

Transmission Energy. The transmission energy required to move task k from node j (source) to node i (destination) depends on data size and per-bit link costs:

$$E_{\text{trans}}^{i,k} = s^k \sum_{(m,n) \in I^{j,i}} C^{m,n}, \tag{14}$$

where:

- s^k is the size of task k in bits.
- $C^{m,n}$ is the energy cost per bit (J/bit) on the link from node m to node n.
- $I^{j,i}$ is the set of links used along the path from node j to node i.

This model offers a structured, interpretable framework for quantifying energy consumption in offloading scenarios, thereby facilitating meaningful comparisons of energy efficiency under different scheduling and resource-allocation strategies.

4. Proposed Offloading Strategies

A range of offloading strategies was proposed to explore different decision-making paradigms, including NPGA, NSGA, and DQL. Each approach balances complexity and global optimality to varying degrees.

4.1. Metaheuristics

Metaheuristic algorithms, such as evolutionary methods, excel at exploring large, dynamic search spaces in IoT offloading. Although they have been applied in more static contexts by Bernard et al. (2024), this work extends them to a *dynamic* genome representation namely, an MLP mapping real-time states to offloading actions, following ideas proposed by Such et al. (2018).

4.1.1. GA

Genome Representation (MLP Encoding). A candidate solution (or individual) is encoded as the set of weight matrices and bias vectors in a multi-layer perceptron (MLP). Each layer ℓ has trainable parameters $\mathbf{W}_\ell$ (weights) and $\mathbf{b}_\ell$ (bias). Thus, the full network is defined as

$$\left\{ (\mathbf{W}_1, \mathbf{b}_1), (\mathbf{W}_2, \mathbf{b}_2), \ldots, (\mathbf{W}_L, \mathbf{b}_L) \right\}.$$

These parameters determine how input features (e.g., CPU, buffer, bandwidth) are progressively transformed into node-specific scores for offloading. Evolutionary operators act directly on both the weight matrices and the bias vectors over multiple generations.

MLP Forward Mapping. For an input feature vector $\mathbf{X}$, the MLP computes the output $\mathbf{Y}$ layer by layer:

$$\mathbf{h}_1 = \phi(\mathbf{W}_1 \mathbf{X} + \mathbf{b}_1), \quad \mathbf{h}_2 = \phi(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2), \quad \ldots \quad \mathbf{Y} = f(\mathbf{W}_L \mathbf{h}_{L-1} + \mathbf{b}_L),$$

where $\phi(\cdot)$ is a non-linear activation function (e.g., ReLU, sigmoid), and $f(\cdot)$ is the output activation (e.g., softmax for classification, linear for regression).

Crossover and Mutation. **Crossover:** Two parent genomes $\{(\mathbf{W}_\ell^{(p1)}, \mathbf{b}_\ell^{(p1)})\}$ and $\{(\mathbf{W}_\ell^{(p2)}, \mathbf{b}_\ell^{(p2)})\}$ are combined by arithmetic crossover:

$$\mathbf{W}_\ell^{(\text{child})} = \alpha \, \mathbf{W}_\ell^{(p1)} + (1-\alpha) \, \mathbf{W}_\ell^{(p2)}, \qquad \mathbf{b}_\ell^{(\text{child})} = \alpha \, \mathbf{b}_\ell^{(p1)} + (1-\alpha) \, \mathbf{b}_\ell^{(p2)},$$

where $\alpha \in [0,1]$ is chosen at random, ensuring offspring inherit traits from both parents.

Mutation: With probability p_{mutation}, each element of a weight matrix or bias vector is perturbed by Gaussian noise:

$$\mathbf{W}[i,j] \leftarrow \mathbf{W}[i,j] + \epsilon_{ij}, \qquad \mathbf{b}[k] \leftarrow \mathbf{b}[k] + \delta_k,$$

where $\epsilon_{ij}, \delta_k \sim \mathcal{N}(0, \sigma^2)$. Clipping may be applied to maintain valid parameter ranges.

Overall Workflow of Genetic Algorithms. The genetic algorithm process begins with population initialization, where an initial population of candidate solutions (chromosomes) is randomly generated using an appropriate representation format. Each individual in the population is then evaluated using multiple objective functions to determine fitness values across all optimization criteria. The selection phase follows, where parent individuals are chosen for reproduction based on their performance scores. The core evolutionary operations of crossover and mutation are then applied to generate offspring by recombining selected parents through crossover operations and introducing random variations through mutation. Subsequently, the replacement phase evaluates new offspring and forms the next generation by combining or replacing individuals from the parent and offspring populations using multi-objective replacement strategies. The algorithm terminates after reaching a maximum number of generations, achieving convergence criteria, or meeting other predefined stopping conditions. The final Pareto front represents the set of non-dominated solutions, effectively demonstrating the trade-offs among competing objectives. The pseudo code of the GA is referenced in Algorithm 4.1.1.

Multi-objective GA. As the GA algorithm is based on multiple individuals, it can achieve multiple objectives. During the selection process, the selected individuals were able to select multiple scores, rather than a single score, thereby creating the Pareto front. The present study focuses on two scalable and robust approaches.

NPGA (Horn et al., 1994) employs a combination of tournament selection and a niching mechanism to maintain population diversity and prevent premature convergence to a single region of the Pareto front. The selection process works by conducting tournaments between randomly selected candidate solutions, where the winner is determined based on Pareto dominance relationships within a randomly chosen comparison set. Specifically, two candidates compete by counting how many individuals they dominate from a subset of the population, and the candidate with higher dominance count is selected. This approach naturally maintains diversity by distributing selection pressure across different regions of the Pareto front without requiring explicit front classification.

In contrast, *NSGA-II* (Deb et al., 2002) uses non-dominated sorting to classify solutions into hierarchical fronts and employs crowding distance calculations to preserve solution diversity within each front and guide selection for the next generation. The selection mechanism first performs non-dominated sorting to organize the population into ranked fronts $(\mathcal{F}_1, \mathcal{F}_2, \ldots)$, where $\mathcal{F}_1$ contains the best non-dominated solutions. Within each front, crowding distance is calculated to measure the density of solutions in the objective space, favoring individuals in less crowded regions. Selection prioritizes individuals from better fronts first, and within the same front, those with larger crowding distances are preferred to maintain diversity along the Pareto front.

Algorithm 1 Multi-Objective Genetic Algorithm for IoT Task Offloading

---

Require: Generations $G_{\max}$, population size N, crossover rate p_c, mutation rate p_m
Ensure: Pareto-optimal offloading policies $\mathcal{P}^*$
1: Initialize population $\mathcal{P}_0$ with N random MLP genomes
2: **for** $g = 0$ to $G_{\max} - 1$ **do**
3:    **Evaluate:** For each individual $\chi_i \in \mathcal{P}_g$
4:       Simulate offloading with MLP weights from χ_i
5:       Compute objectives: f_L (latency), f_E (energy), f_{TTR} (timeout rate)
6:    **Select:** Create mating pool $\mathcal{M}_g$ using multi-objective selection
7:       (NSGA-II non-dominated sorting + crowding distance)
8:       (NPGA Pareto tournament selection)
9:    **Reproduce:** Generate offspring $\mathcal{Q}_g$
10:    **for** $i = 1$ to N **do**
11:       Select parents χ_p, χ_q from $\mathcal{M}_g$
12:       Apply crossover (prob. p_c) and mutation (prob. p_m)
13:       Add offspring to $\mathcal{Q}_g$
14:    **end for**
15:    **Replace:** Form next generation $\mathcal{P}_{g+1}$ from $\mathcal{P}_g$ and $\mathcal{Q}_g$
16: **end for**
17: **return** Non-dominated solutions from final population

4.2. Deep Q-Learning

DQL is well-suited for IoT offloading because (i) the environment state (CPU, buffer, bandwidth, task attributes) evolves over time, (ii) each offloading decision alters subsequent states and tasks, and (iii) a reward can be assigned at each step (e.g., penalizing task failures or excessive latency).

Core Algorithm. A neural network approximates the Q-function, estimating the long-term value of choosing an action (offloading to a particular node) in a given state. Training leverages mini-batches from an experience replay buffer of past transitions (s, a, r, s'). Iteratively, these Q-values converge, leading to improved decisions over time.

Workflow. The DQL workflow, as detailed in Algorithm 2, begins with initialization where an environment is built reflecting a specific scenario with fog/cloud nodes and tasks, an IoT task dataset is split into training/testing sets, and the neural network is configured with parameters such as hidden layers and learning rate. At each step, the agent constructs an observation by creating a flattened vector of node resources including free CPU, buffer, and link bandwidth, which the neural network uses to estimate Q-values for each node. Action selection follows an ε-greedy policy where with probability ε, a random node is chosen for exploration, while otherwise the node with the highest Q-value is selected for exploitation. The environment is then updated as the chosen node processes the task, with the simulation advancing until the task completes or another event occurs such as queue filling or deadline miss, potentially causing task failure. Upon task completion or failure, a reward r is calculated and the transition (s, a, r, s') is stored in the replay buffer, where the reward function is defined as:

$$r = \begin{cases} \lambda_0, & \text{if task fails;} \\ \lambda_1 \dfrac{L_{\text{task}}}{\max(L_{\text{epoch}})} + \lambda_2 \dfrac{E_{\text{task}}}{\max(E_{\text{epoch}})}, & \text{otherwise.} \end{cases}$$

Here, λ_0 (often negative/zero) penalizes failure, $L_{\text{task}}, E_{\text{task}}$ are normalized by epoch maxima, and λ_1, λ_2 tune the importance of latency and energy, respectively. Finally, after a certain number of tasks, transitions are sampled from the replay buffer for training updates, where the Q-learning target is computed as $Q_{\text{target}} = r + (1 - \mathbf{1}_{\text{done}}) \gamma \max_{a'} Q(s', a')$, and the network is trained using methods such as MSE loss to align predicted Q-values with Q_{target}, with Q-value estimates converging over multiple epochs.

11

Key Observations. Several key observations emerge from the DQL approach. Failure penalties implemented through a negative λ_0 can strongly discourage decisions that often cause task failures, providing a direct mechanism to avoid poor offloading choices. Metric normalization achieved by dividing latency and energy by epoch-wide maxima ensures bounded values for stable learning, preventing any single metric from dominating the reward signal. Metric emphasis can be adjusted by tuning $\lambda_{1,2}$ to shift the balance between minimizing latency and saving energy, allowing the system to be configured for different optimization priorities. Finally, architectural flexibility is maintained since although an MLP is standard, more sophisticated models such as Transformers can replace or extend the MLP while retaining the same DQL pipeline.

4.2.1. MLP-Based DQL

MLP-based DQL mirrors the MLP structure used in genetic algorithms, but instead of evolving weights, it trains them via Q-learning. The architecture consists of a feed-forward MLP that processes the current system state including CPU, buffer, bandwidth, and other relevant metrics, and outputs Q-values per node to guide the offloading decision. The policy employs an ε-greedy strategy that balances exploration and exploitation, ensuring the agent can discover new promising strategies while leveraging learned knowledge. The reward definition encodes latency, energy, and failures into a comprehensive reward function that captures the multi-objective nature of the offloading problem. Batch updates are performed by sampling from a replay buffer, which helps stabilize training and reduce correlation among transitions, leading to more robust learning dynamics.

4.2.2. Transformer-based DQL

Transformer architectures (Vaswani et al., 2023) leverage multi-head self-attention to model complex relationships across various domains, despite often having high parameter counts. Task offloading is no exception, as shown by Gholipour et al. (2023), where a PPO-based policy was employed for Transformers. In the present work, two Transformer-based DQL variants are introduced to enable task offloading across multiple nodes:

NOTE. NOTE encodes each node's available resources (CPU, buffer, and bidirectional link bandwidth) into an embedding vector. A positional encoding is then added to identify each node's location in the topology. A Transformer encoder stack processes these node embeddings in parallel, enabling the model to learn both pairwise and global context. Finally, a feed-forward projection estimates the Q-value for each node, guiding the offloading decision. Because NOTE primarily targets node-level features, tasks are treated in an aggregated manner.

- **Node Embeddings:** CPU frequency, buffer size, and upstream/downstream bandwidth are first passed through a fully connected layer to produce an embedding of dimension d_{model}, where d_{model} is the hidden dimension of the model. Concatenating all node embeddings yields a tensor of size $n_{\text{nodes}} \times d_{\text{model}}$, where n_{nodes} is the total number of nodes in the environment.

- **Positional Encoding:** Trainable parameters that characterize each node's position in the network topology. These embeddings also form a $n_{\text{nodes}} \times d_{\text{model}}$ matrix.

- **Transformer Encoder:** Multi-head self-attention highlights potential bottlenecks and optimal resource matches among the nodes. Residual connections and layer normalization are used, followed by a feed-forward sublayer with a GELU activation (Hendrycks and Gimpel, 2023). This encoder block is repeated N times.

- **Output Layer:** A fully connected layer projects the encoded representations to Q-values, producing one scalar per node. The node with the highest Q-value is then chosen for offloading.
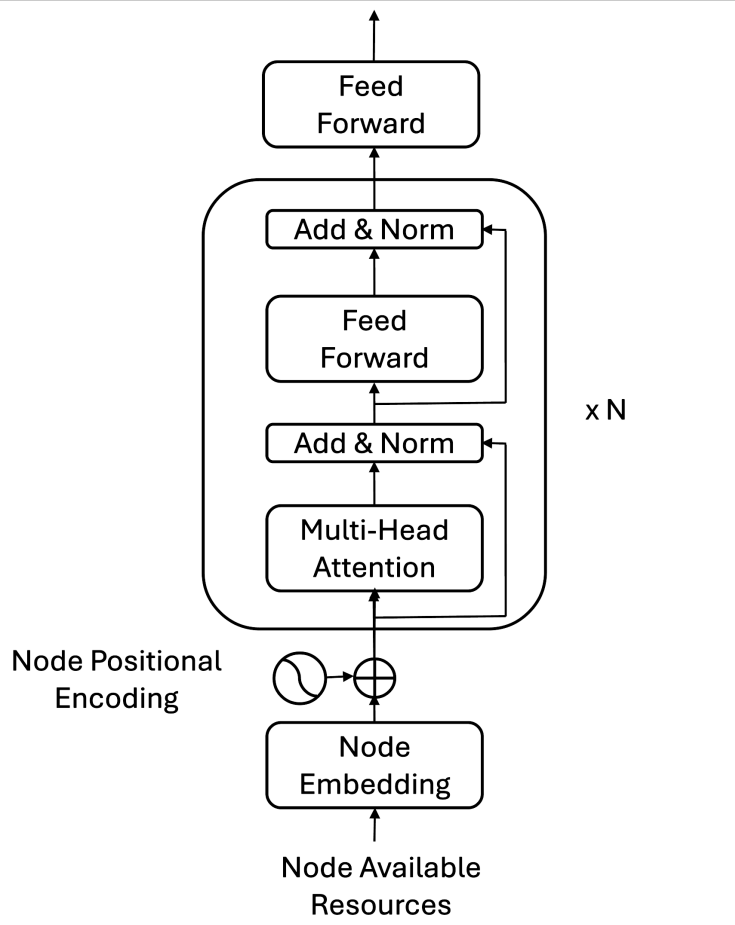
Figure 2: NOTE Architecture

T-NOTE. T-NOTE extends NOTE by incorporating task attributes (e.g., size, deadline, and CPU-cycle requirements). These task features are passed through a fully connected layer to produce a vector of size d_{model}. The resulting vector is then replicated across the n_{nodes} dimension and added to the node embeddings and node positional encodings before entering the Transformer encoder. Task-specific positional encoding can be combined or managed separately, allowing the model to capture task characteristics through shared attention. While including task information enables the Transformer encoder to more accurately learn interactions between node resources and task demands, it also increases the complexity of each DQL state. This added complexity sometimes leads to greater model instability, as rapid changes in the task dimension can cause oscillations in policy behavior. Consequently, NOTE (which omits explicit task embeddings) may remain advantageous in scenarios where a simpler, more stable representation is desired.
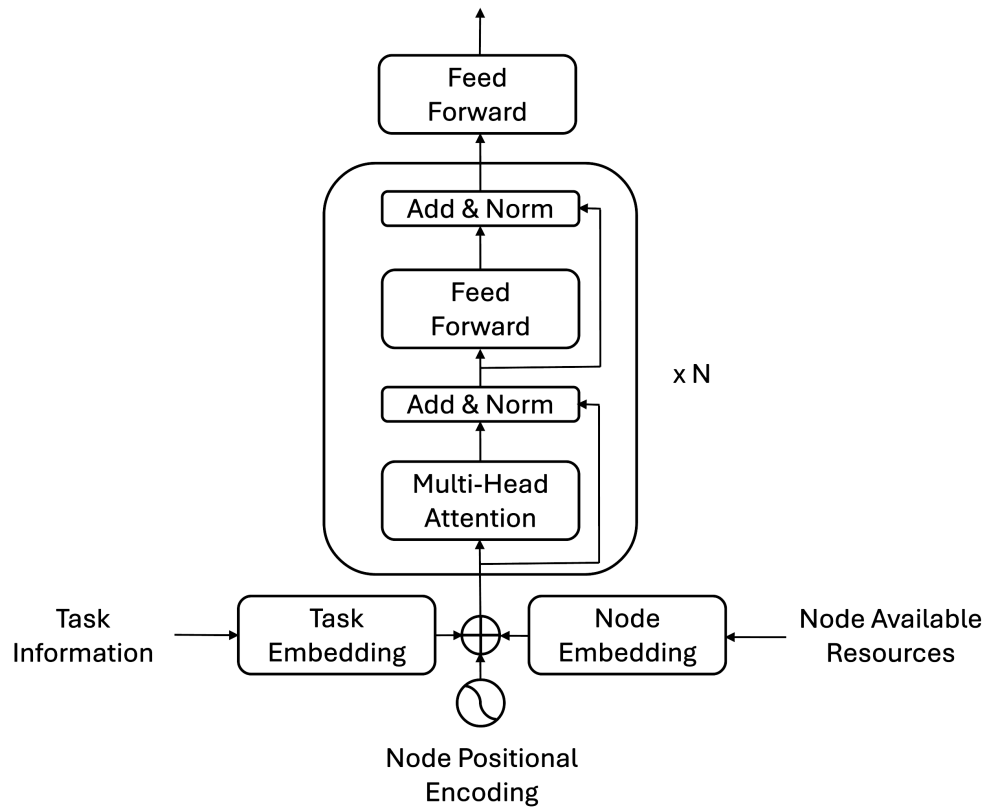
Figure 3: T-NOTE Architecture

Algorithm 2 Deep Q-Reinforcement Learning for IoT Task Offloading

Require: Environment $\mathcal{E}$, task dataset $\mathcal{D}$, hyperparameters $\{\alpha, \gamma, \epsilon_0, \epsilon_{\text{decay}}, \lambda_0, \lambda_1, \lambda_2\}$
Ensure: Trained Q-network policy π^*
 1: Initialize Q-network Q_θ with random parameters θ
 2: Initialize experience replay buffer $\mathcal{B} \leftarrow \emptyset$
 3: Set exploration rate $\epsilon \leftarrow \epsilon_0$
 4: **for** epoch $e = 1$ to $E_{\max}$ **do**
 5:    Initialize epoch metrics: $L_{\max}^{(e)} \leftarrow 0$, $E_{\max}^{(e)} \leftarrow 0$
 6:    **for** each task $\tau_i \in \mathcal{D}$ ordered by generation time **do**
 7:       Wait until $t_{\text{current}} \geq \tau_i.\text{arrival\_time}$
 8:       Construct state vector s_i from system resources/task features
 9:       **Action Selection:**
10:       **if** rand() $< \epsilon$ **then**
11:          $a_i \leftarrow$ random edge node
12:       **else**
13:          $a_i \leftarrow \arg\max_a Q_\theta(s_i, a)$
14:       **end if**
15:       Execute offloading action a_i and simulate task execution
16:       Observe next state s_{i+1} and compute reward:
17:

$$r_i = \begin{cases} \lambda_0 & \text{if task execution fails} \\ -\lambda_1 \cdot \frac{L_i}{L_{\max}^{(e)}} - \lambda_2 \cdot \frac{E_i}{E_{\max}^{(e)}} & \text{otherwise} \end{cases}$$

18:       Store transition (s_i, a_i, r_i, s_{i+1}) in $\mathcal{B}$
19:       **if** $|\mathcal{B}| \geq$ batch_size **or** end of epoch **then**
20:          Sample mini-batch $\mathcal{M}$ from $\mathcal{B}$
21:          **for** each $(s, a, r, s') \in \mathcal{M}$ **do**
22:             Compute target: $y = r + \gamma \cdot \max_{a'} Q_\theta(s', a')$
23:          **end for**
24:          Update Q-network: $\theta \leftarrow \theta - \alpha\nabla_\theta \mathcal{L}(\theta)$
25:          where $\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{M}}\left[(Q_\theta(s, a) - y)^2\right]$
26:          Decay exploration: $\epsilon \leftarrow \epsilon \cdot \epsilon_{\text{decay}}$
27:       **end if**
28:    **end for**
29: **end for**
30: **return** Optimal policy $\pi^*(s) = \arg\max_a Q_\theta(s, a)$

4.3. Computational Complexity

The computational complexity of neural network models during inference is primarily determined by their architecture and is independent of the training method. However, training complexity varies significantly based on the optimization algorithm employed. Below, we compare the training-time complexities for Multi-Layer Perceptrons (MLPs) trained with Deep Q-Learning (DQL) and Genetic Algorithms (GAs), as well as Transformers trained with DQL.These approaches are particularly relevant in scenarios where standard supervised learning is infeasible, such as reinforcement learning environments or non-differentiable optimization landscapes. We focus on the key factors influencing training costs, assuming similar network architectures where applicable for fair comparison.

4.3.1. MLP Trained with DQL

For an MLP with L hidden layers, where d_l denotes the number of neurons in layer l (including input dimension d_0 and output dimension d_{L+1}), the cost of a single forward and backward pass, C, is

$$C = \mathcal{O}\left(\sum_{l=1}^{L+1} d_l \cdot d_{l-1}\right).$$

DQL, a reinforcement learning method, involves sequential interactions with an environment, where the model learns to maximize cumulative rewards through trial-and-error. This requires E episodes, each with T time steps, during which the network performs forward passes to select actions, receives rewards, and updates via backpropagation (often using experience replay and target networks for stability). The overall training complexity is thus approximated as

$$\mathcal{O}(E \cdot T \cdot C).$$

This is substantially higher than supervised learning due to the need for exploration, temporal credit assignment, and handling sparse or delayed rewards, leading to slower convergence and greater computational demands.

4.3.2. MLP Trained with GA

Using the same MLP architecture, the per-forward-pass cost remains

$$C = \mathcal{O}\left(\sum_{l=1}^{L+1} d_l \cdot d_{l-1}\right).$$

However, GAs employ a population-based evolutionary search, evaluating a population of P candidate weight sets over G generations. Fitness assessment for each individual typically involves forward passes across a dataset of N samples (or simulated environments). Evolutionary operators (selection, crossover, mutation) add minor overhead but are dominated by evaluations. The training complexity is therefore

$$\mathcal{O}(G \cdot P \cdot N \cdot C).$$

Compared to DQL, GA training can be even more computationally intensive in large populations or datasets, as it lacks gradient guidance and relies on black-box optimization. This often results in lower sample efficiency but can excel in rugged, non-differentiable landscapes where gradients are unavailable.

4.3.3. Transformer Trained with DQL

Transformers introduce a different architectural complexity. For a model with L layers, hidden size d, and sequence length n, the cost of a single forward and backward pass, C, is

$$C = \mathcal{O}\left(L \cdot (n^2 d + n d^2)\right),$$

driven by quadratic self-attention and linear feed-forward operations. When trained with DQL, the process mirrors MLP-DQL E episodes of T time steps each, involving action selection, reward feedback, and updates. The training complexity is

$$\mathcal{O}(E \cdot T \cdot C).$$

Relative to MLP-based DQL, Transformer's higher per-pass cost (C)—due to sequence-length scaling—amplifies the overall expense, making it particularly demanding for long-sequence tasks. Like MLP-DQL, it suffers from reinforcement learning's inherent inefficiencies but benefits from Transformer's strong representational power in sequential decision-making.

5. Performance Evaluation

5.1. Experimentation Setup

All experiments were conducted on two distinct hardware configurations.

DQL Experiments.. The DQL algorithms (both MLP-based and Transformer-based) were trained on a workstation equipped with:

- CPU 16 cores

- RAM 64 GB

- GPU NVIDIA Tesla P100 with 16 GB memory

This configuration provided sufficient computational power to accelerate neural network training on large batches.

GA Experiments.. The GA-based methods (NPGA, NSGA-II) were executed on a high-performance server with:

- CPU 192 cores

- RAM 256 GB

This setup enabled extensive parallelism, allowing multiple individuals to be evaluated simultaneously and thus accelerating convergence for evolutionary algorithms.

5.1.1. Dataset

A real IoT task trace collected in Islamabad, Pakistan, is utilized as described by Aazam et al. (2022). This dataset contains heterogeneous IoT jobs (commonly referred to as *tuples*).

The dataset covers diverse devices (sensors, dumb objects, mobiles, actuators, and location-based nodes). Originally, data were sampled every minute over a one-hour period. To avoid clustering all tasks within the same second of each minute, tasks were uniformly distributed across that minute, producing a more realistic workload. The statistics of these tasks are reported in Table 3

Statistic	Generation Time (s)	Task Size (MB)	Cycles/Bit	Trans. Bit Rate (MB/s)	DDL (s)
Count	30,000	30,000	30,000	30,000	30,000
Mean	1891	206	344	88	60
Std Dev	1094	75	351	39	23
Min	0	80	50	20	20
25%	941	170	100	80	39
50%	1884	220	200	90	60
75%	2856	270	700	100	79
Max	3780	300	1200	150	99

Table 3: Statistical Summary of Generated Tasks

This dataset was adapted for the RayCloudSim framework, retaining only the most pertinent variables and appending additional information (for example, the number of cycles per bit). The units—originally unspecified in the source—were standardized to align with typical IoT node specifications, thereby ensuring consistency and accuracy in subsequent simulations and experiments.

5.1.2. System Resources and Topology

To align with the dataset location, we designate the GG in Islamabad, Pakistan as the Edge node. The Fog nodes are configured as Micro-Data Centers (MDCs) distributed across various cities in Pakistan to provide regional computational resources. For the Cloud layer, we select data centers based on Google's global data center locations (Google, 2024), specifically choosing the geographically nearest data centers to Pakistan to minimize latency overhead and ensure realistic network conditions.

In Figure 4, a network is shown to consist of eight nodes (*Edge*, *Fog*, and *Cloud*) connected by multiple links with diverse bandwidth capacities. The *edge node* (e0) is equipped with a low CPU frequency and a limited task buffer, whereas the *fog nodes* (f0, f1, f2, f3, f4) has intermediate CPU frequencies and queue sizes. The *cloud nodes* (c0, c1) exhibits the highest CPU frequencies and large buffer capacities, although they may incur higher idle power consumption and larger network latency due to their remote location.
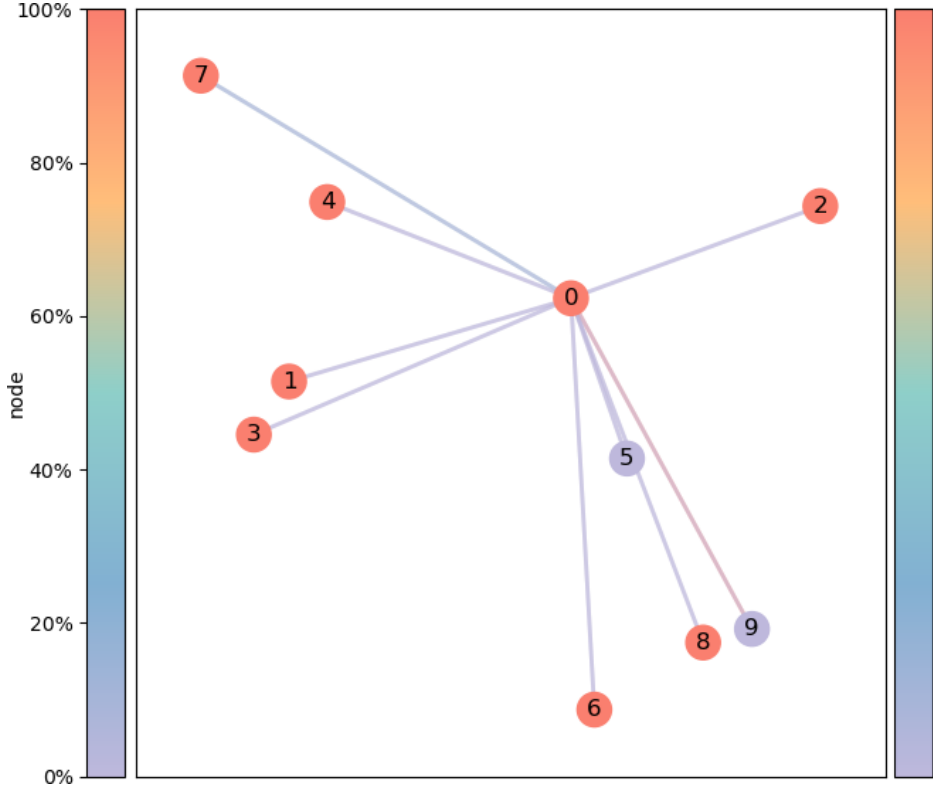
Figure 4: Used Architecture featuring Edge (Global Gateway), Fog (MDCs), and Cloud.

In Table 4, node specifications are listed, including Millions of Instructions Per Second (MIPS), the buffer size in GB, the up and down bandwidth from/to e0 (the global gateway) in GB and power coefficients in W. The network links range from 700 bps to 50,000 bps, forming a heterogeneous environment suitable for analyzing resource allocation and routing strategies under variable link constraints.

5.2. Hyperparameters

Since our modeling framework assigns zero latency and energy values to tasks that fail to be successfully offloaded, the primary optimization objective becomes the minimization of the *task rejection rate*. This constraint is essential to prevent model degeneration, where the system could exploit the null assignment by deliberately rejecting tasks to artificially improve energy and latency performance metrics.

To further stabilize optimization, we adopt a hierarchical prioritization scheme that emphasizes latency over energy efficiency, as latency more directly impacts user experience and system responsiveness. This

Node	Type	ID	MIPS	Buffer	Up BW	Down BW	Idle P	Exec P	Location
e0	Edge	0	10	4	–	–	3	10	Islamabad, PK
f0	Fog	1	110	16	2.5	1.7	30	150	Multan, PK
f1	Fog	2	50	6	1	700	15	50	Gilgit, PK
f2	Fog	3	95	12	2	1.5	25	120	Karachi, PK
f3	Fog	4	85	10	1.5	1.2	20	100	Lahore, PK
f4	Fog	5	75	8	1.2	1	20	100	Peshawar, PK
c0	Cloud	6	500	51	3	3	300	1100	Singapore
c1	Cloud	7	500	51	3	3	250	1000	Belgium

Table 4: Detailed Specifications of Nodes

prioritization is reflected in the reward weight hierarchy:

$$\lambda_0 \gg \lambda_1 > \lambda_2.$$

Based on empirical evaluation of different configurations using a Multi-Layer Perceptron (MLP) architecture trained with Deep Q-Reinforcement Learning (DQL), we selected the following reward weight distribution:

Reward weights $(\lambda) = [1, 0.1, 0.05]$.

Finally, to improve clarity and maintain consistency across training and evaluation, the energy consumption metric is reported in terms of power consumption throughout the experimental analysis.

5.2.1. Model Configurations

The complete hyperparameter specifications are detailed in Table 5. The MLP architectural parameters remain consistent across both Genetic Algorithm (GA)-based approaches and DQL implementations to ensure comparative validity.

Parameter	Transformer Model	MLP Model
Hidden dimension (d_{model})	64	256
Number of layers (n_{layers})	6	3
Number of heads (n_{heads})	4	-
Feed Forward Network (d_{ff})	256	-
Dropout	0.2	0
Activation function	GELU	ReLU
Input features	{cpu, bw, buffer}	{cpu, bw, buffer}

Table 5: Model Specifications

5.2.2. Heuristic Baseline Methods

We implement baseline heuristic approaches for comparative evaluation, namely, *Random*, *Round Robin (RR)* and *Greedy*.

Random. Random strategy, consist to randomly assign tasks to nodes uniformly at random, without consideration of current resource availability or node capabilities.

Round Robin. RR is a straightforward load balancing technique where tasks are cyclically distributed among nodes, with each node receiving tasks in turn before cycling back to the first node. This ensures basic load balancing but ignores node heterogeneity such as CPU frequencies and buffer sizes, potentially causing inefficiencies under varied workloads.

Greedy. Greedy methods assign tasks based on immediate resource availability or specific cost metrics, such as selecting the node with the highest remaining CPU capacity. While easily implemented and computationally efficient, greedy strategies often forgo long-term, globally optimal decisions in favor of immediate local optima.

5.3. Evaluation

Table 6 presents a detailed comparison of eight distinct offloading strategies, ranging from simple heuristic approaches to sophisticated machine learning-based methods. The results reveal significant performance variations across different algorithmic approaches.

Offloading Strategies	TTR (%)	Latency (s)	Energy (W)
Random	14.21	20.89	238.3
Round Robin	14.98	18.36	239.9
Greedy	0.1889	4.394	338.7
MLP	0.1111	4.332	349.0
NOTE	**0.0000**	4.150	337.6
T-NOTE	**0.0000**	**3.663**	**317.0**

Table 6: Comparison of Offloading Strategies with Task Throw Rate (TTR), Latency (L), and Energy (E) on Pakistan test set. Best results are in **bold**.

Random and Round Robin exhibit high Task Throw Rates (TTR) exceeding 14% and latency over 18 seconds. This is due to their simplistic task assignment mechanisms, which do not account for the varying resource capacities of nodes but rather distribute tasks evenly, favoring energy efficiency as more fog nodes are utilized 5.6.1. In contrast, the Greedy approach significantly reduces TTR to 0.1889% and latency to 4.3936 seconds by prioritizing nodes with the highest available CPU resources 5.6.2. However, this comes at the cost of increased energy consumption (338.6652 W) due to the overutilization of high-capacity nodes. Since this paper focuses primarily on minimizing TTR, the Greedy method's performance is taken as the baseline.

The DQL-based approaches demonstrate substantial improvements over the Greedy baseline, with progressively refined architectures yielding incrementally better performance. The MLP strategy achieves a TTR of 0.1111%, representing a 41.2% reduction compared to Greedy, while simultaneously reducing latency to 4.3315 seconds (a 1.4% improvement). However, this comes at the expense of higher energy consumption (348.9158 W, 3.0% increase over Greedy), suggesting that the MLP's learned offloading policy favors reliability and speed over energy efficiency. The transformer-based NOTE architecture further advances performance by eliminating task failures entirely (0.0000% TTR) and reducing latency to 4.1504 seconds (5.5% improvement over Greedy), while maintaining comparable energy consumption (337.5961 W). This improvement stems from the transformer encoder's ability to capture complex relationships between node observations through self-attention mechanisms. Building upon this foundation, T-NOTE extends the architecture by incorporating task-level observations alongside node characteristics, enabling the model to make more contextually aware offloading decisions. This extension yields the best overall performance across all metrics: maintaining zero task failures while achieving the lowest latency (3.6626 seconds, 16.6% improvement over Greedy) and the most efficient energy consumption (317.0230 W, 6.4% reduction compared to Greedy), demonstrating that task-aware attention mechanisms enable superior resource allocation strategies.

5.4. MLP-Based DQL

The MLP-based DQL approach utilizes a feed-forward neural network to estimate Q-values for each node based on the current state of the system, which includes CPU availability, buffer size, and link bandwidth.

Robustness and Performance. The MLP-based DQL approach exhibits sensitivity to hyperparameter configurations, particularly the learning rate and reward weights. While it can achieve performance improvements over heuristic baselines, its convergence behavior is less stable compared to Transformer-based methods. The MLP's limited capacity may contribute to this instability, as it relies on fixed-size input vectors.

Convergence Analysis. Figure 5 illustrates the convergence behavior of the MLP-based DQL model over training epochs. The best model is picked based on the highest average score, which combines TTR, L and E metrics with reward weights, at the 9th epoch.
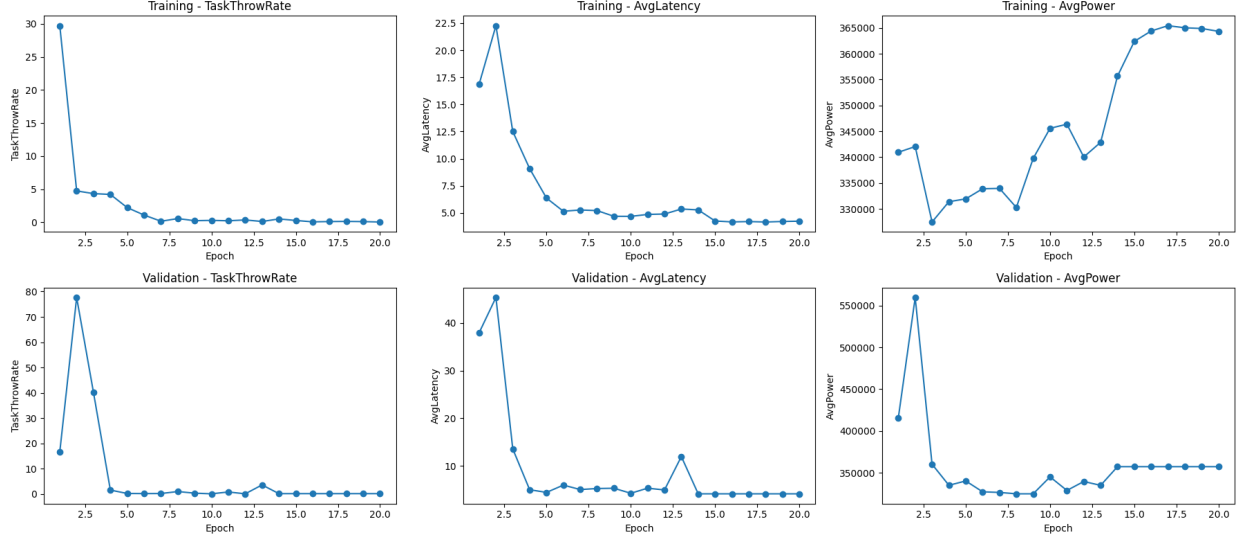
Figure 5: Convergence and Scores for MLP-Based DQL

5.5. Transformer-Based DQL

Transformer architectures, such as NOTE and T-NOTE, were also evaluated under a DQL paradigm. Although these models have far more parameters (e.g., around 350k, which is roughly 50 times more than the MLP), they have demonstrated superior performance in practice.

Robustness and Performance. Despite their complexity, the self-attention mechanisms in NOTE and T-NOTE appear more robust to hyperparameter variations, converging to superior solutions that surpass the MLP-based approaches. This finding aligns with Gholipour et al. (2023), suggesting that Transformers can effectively capture nuanced interactions between tasks and nodes.

5.5.1. NOTE

NOTE encodes only node-level features (CPU, buffer, bandwidth), aggregated over tasks. As illustrated in Figure 6, the learning process initially exhibits some fluctuation but gradually settles into a stable solution. This convergence highlights the capability of self-attention to capture and refine relationships among multiple nodes. Although small oscillations persist, the overall trend consistently improves latency and task throughput rates.
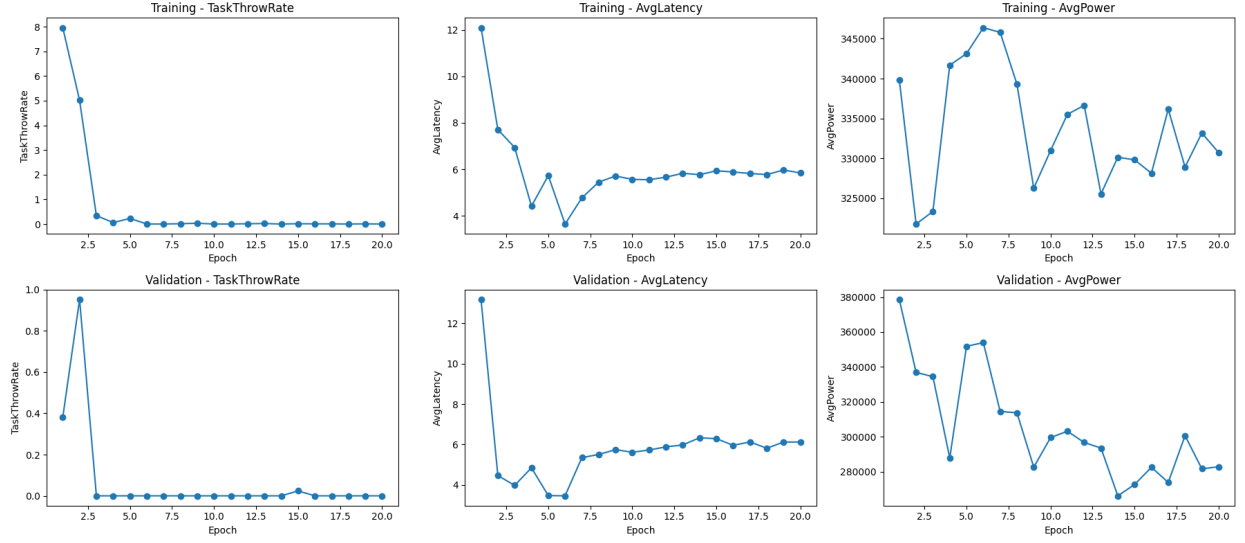
Figure 6: Convergence and Scores for NOTE-Based DQL

5.5.2. T-NOTE

- The task-specific embeddings introduce additional complexity, potentially causing the network to over-fit or oscillate when new tasks arrive. T-NOTE augments NOTE by including task attributes (size, deadline, CPU-cycle requirements). These additional features allow the model to make more informed offloading decisions, resulting in faster convergence and often superior performance in terms of latency and success rate. Figure 7 shows that T-NOTE reaches an optimal solution around the third epoch, but then experiences slight performance degradation as training continues. This fluctuation may occur because:

- DQL's state representation depends on real-time task information, not just node-level resources.

Despite these variations, T-NOTE still surpasses NOTE in both speed of convergence and final performance, demonstrating the value of incorporating task-aware embeddings.

22
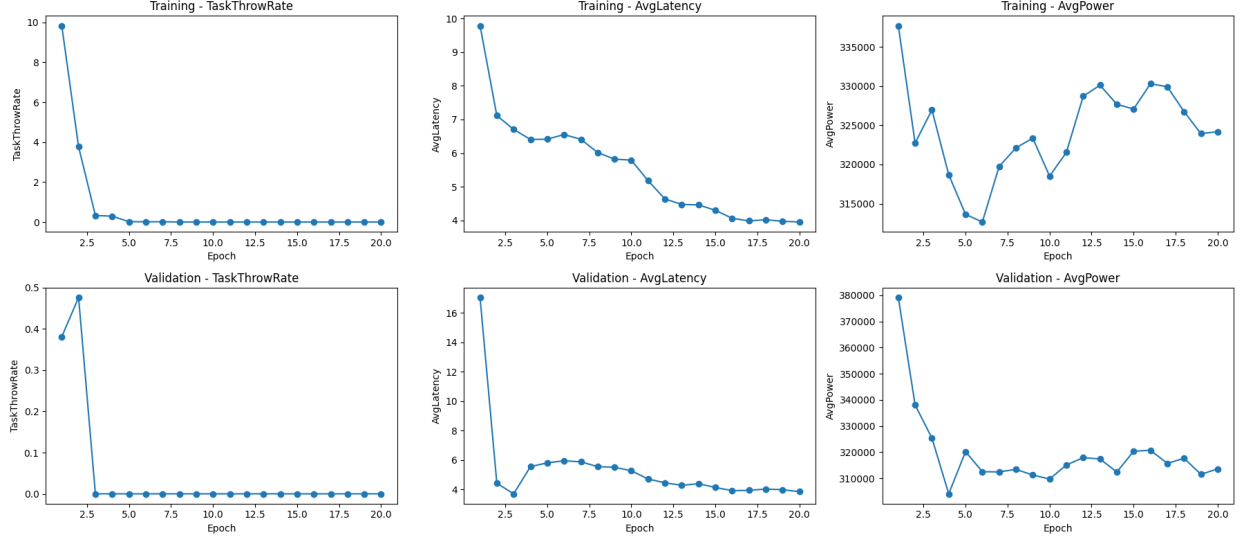
Figure 7: Convergence and Scores for T-NOTE-Based DQL

5.6. Offloading Analysis

This section presents a comprehensive analysis of the offloading strategies employed by different approaches, examining their performance characteristics, resource utilization patterns, and trade-offs. The analysis is structured to provide insights into how each method addresses the fundamental challenges of task allocation in hierarchical edge-fog-cloud computing environments.

A comparative evaluation of offloading policies reveals that *Transformer-based DQL* demonstrates superior robustness by achieving an optimal balance between latency minimization and TTR compared to alternative approaches. While multi-objective GA methods provide a diverse set of Pareto-optimal solutions enabling flexible trade-off selection, MLP-based DQL maintains computational efficiency as its primary advantage. The following subsections provide detailed analysis of node utilization patterns, performance metrics, and the underlying decision-making characteristics of selected methodologies.

5.6.1. Random

The Random offloading strategy serves as a baseline for evaluating the effectiveness of more sophisticated task allocation policies in hierarchical edge-fog-cloud environments. By randomly assigning tasks to available nodes, this approach lacks any awareness of the underlying resource capabilities, often leading to inefficient resource utilization and increased latency.
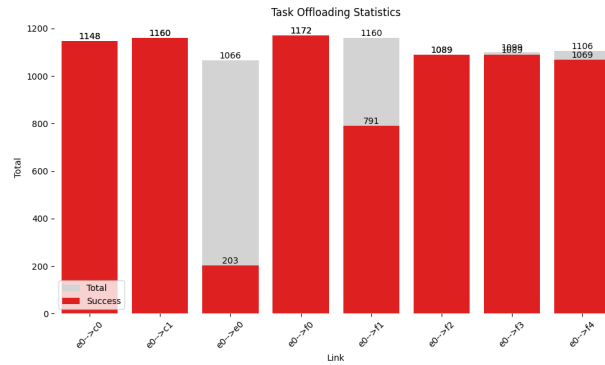
Figure 8: Task offloading distribution across node types for the Random baseline
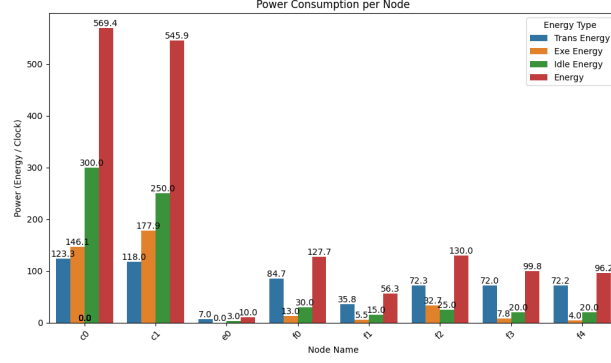
Figure 9: Power consumption per node for the Random baseline

5.6.2. Greedy (baseline)

The Greedy offloading strategy serves as a fundamental baseline for evaluating the effectiveness of more sophisticated task allocation policies in hierarchical edge-fog-cloud environments. The Greedy method's simplicity enables rapid decision-making but often leads to suboptimal global performance due to its myopic nature.
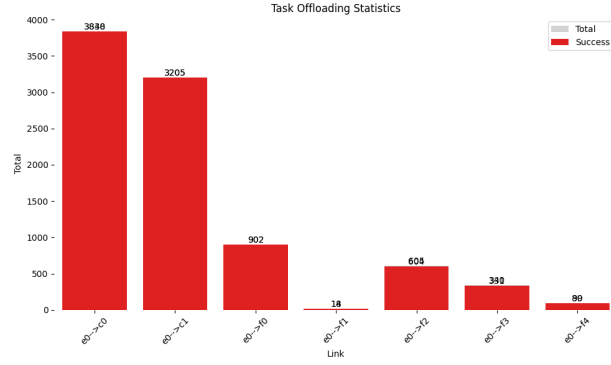

Figure 10: Task offloading distribution across node types for the Greedy baseline

Figure 10 illustrates the distribution of task assignments across edge, fog, and cloud nodes under the Greedy policy. The results indicate a strong preference for offloading to nodes with the higher CPU frequency, often resulting in a disproportionate allocation to cloud nodes, making these nodes more asked for CPU resources as illustrated in Figure 11.
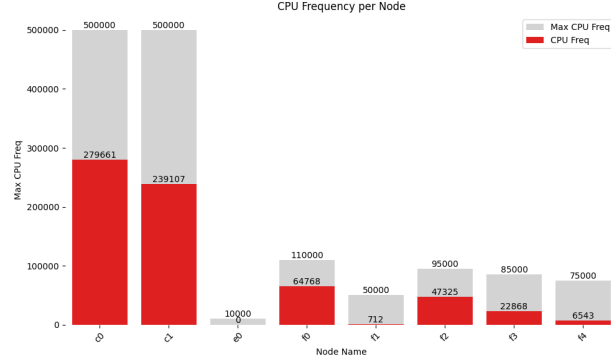
Figure 11: Average CPU frequency utilization per node for the Greedy baseline.

The average communication latency per network link, depicted in Figure 12, demonstrates that the Greedy approach achieves moderate latency performance under typical load conditions.
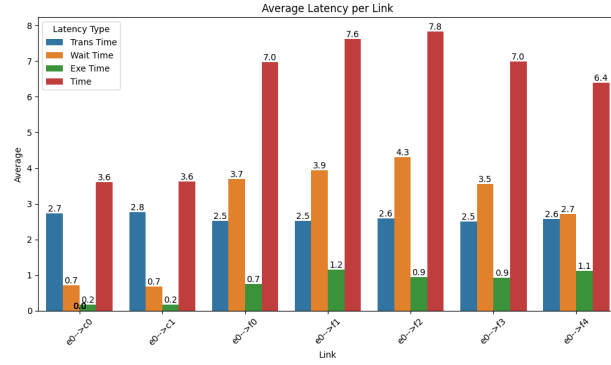
Figure 12: Average communication latency per network link for the Greedy baseline.

Figures 13 present the power consumption across different nodes. The Greedy policy's tendency to overload cloud nodes leads to localized energy hotspots and increased power draw at the cloud tier. In contrast, fog nodes remain underutilized except during peak demand, resulting in an uneven distribution of energy consumption throughout the system.
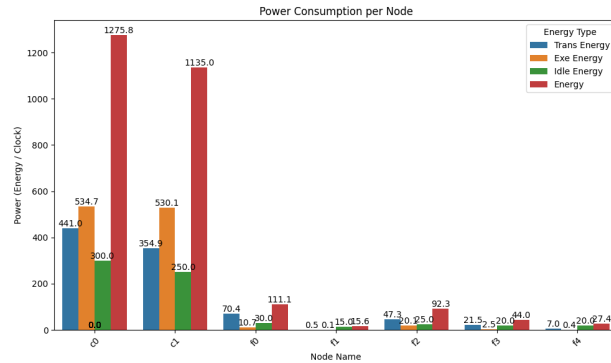
Figure 13: Power consumption distribution across different node types for the Greedy baseline.

25

5.6.3. MLP-Based DQL Performance Analysis

The MLP-based approach, whether evolved through genetic algorithms or trained via deep Q-learning, demonstrates convergence to similar offloading patterns in the final policy. For comprehensive analysis, this study focuses on the DQL variant of the MLP implementation, as the solution founded for DQL is really near than the GA-based multi-objective that minimize the reward weights (λ).

As illustrated in Figure 14, the MLP-based policy exhibits an aggressive cloud-centric offloading strategy. This approach prioritizes cloud node utilization to minimize both task throw rate (TTR) and processing latency, directly reflecting the optimization objective defined by the high weighting factors λ_0 (failure penalty) and λ_1 (latency penalty) in the reward function. The policy demonstrates a clear preference for leveraging the superior computational resources available in cloud nodes, which possess higher processing capabilities and more stable connectivity compared to edge and fog alternatives.
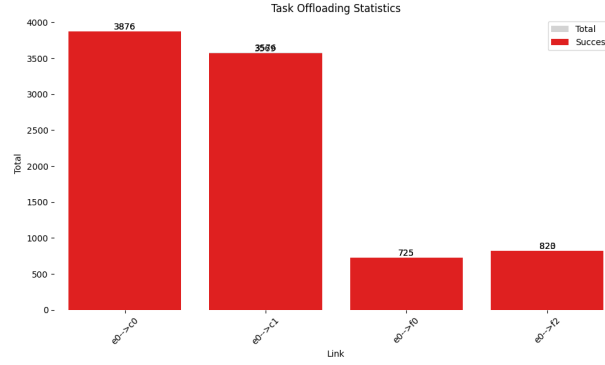

Figure 14: Task offloading distribution across node types for MLP-based DQL, showing the proportion of tasks allocated to edge, fog, and cloud nodes

However, this strategy results in significant underutilization of fog devices, as any tasks are offloaded to the fog, which represents an intermediate tier of computational resources that could potentially provide a more balanced trade-off between performance and energy efficiency. The heavy reliance on cloud computing infrastructure leads to a concentration of energy consumption at the cloud level, as depicted in the subsequent power consumption analysis.
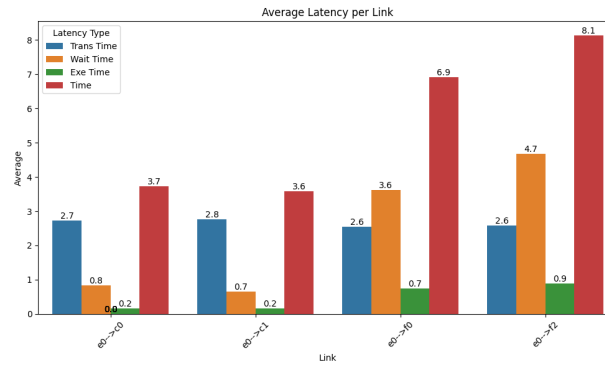

Figure 15: Average communication latency per network link for MLP-based DQL
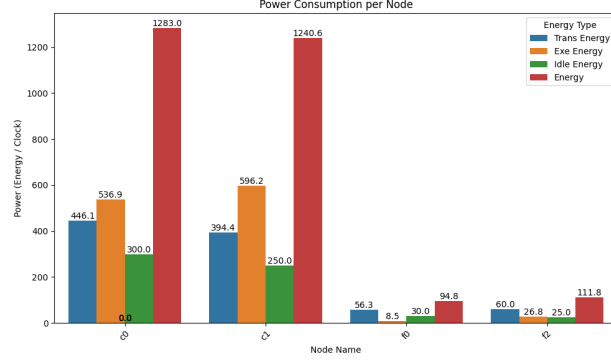
Figure 16: Power consumption distribution across different node types for MLP-based DQL

The performance characteristics shown in Figures 15 and 16 confirm the cloud-centric strategy's effectiveness in achieving low latency objectives while simultaneously revealing its limitations in terms of energy distribution. The concentration of task processing at cloud nodes results in minimal communication and processing latency but creates a substantial energy burden on cloud infrastructure. This trade-off reflects the single-objective optimization focus inherent in the DQL approach, where the primary goal is to maximize a unified reward function rather than explicitly balancing multiple competing objectives.

5.6.4. Transformer-Based DQL (T-NOTE)

The T-NOTE architecture represents a significant advancement over traditional MLP-based approaches by incorporating explicit attention mechanisms for task attribute processing. While both NOTE and T-NOTE variants demonstrate improvements over MLP-based solutions, T-NOTE consistently outperforms NOTE due to its enhanced capability to model task-specific characteristics and their relationships to optimal node selection.
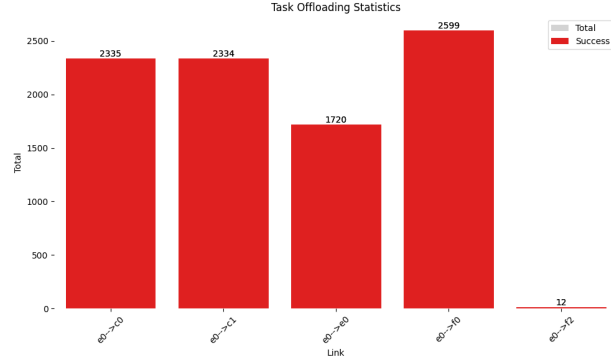

Figure 17: Task offloading distribution for T-NOTE

Figure 17 reveals that T-NOTE maintains a preference for cloud nodes while demonstrating more sophisticated resource allocation strategies. The approach judiciously employs fog resources when cloud queues experience congestion or when specific tasks require lower-latency local processing. This adaptive behavior stems from the Transformer architecture's ability to capture complex relationships between task characteristics, network conditions, and node capabilities through its attention mechanism.

The balanced allocation strategy becomes more apparent when examining the latency characteristics across different network links. Unlike the MLP approach, T-NOTE occasionally utilizes edge nodes for specific use cases, particularly for small computational tasks or time-sensitive operations where local processing provides advantages despite limited computational resources.
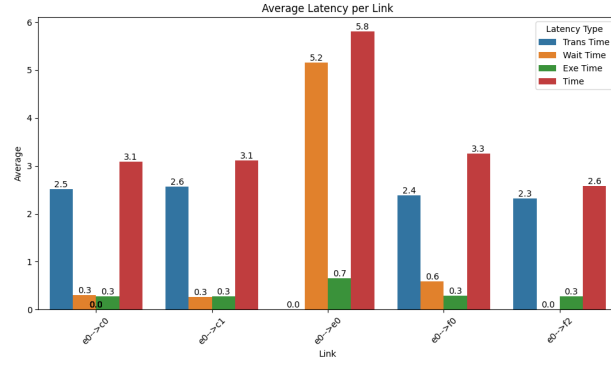
27

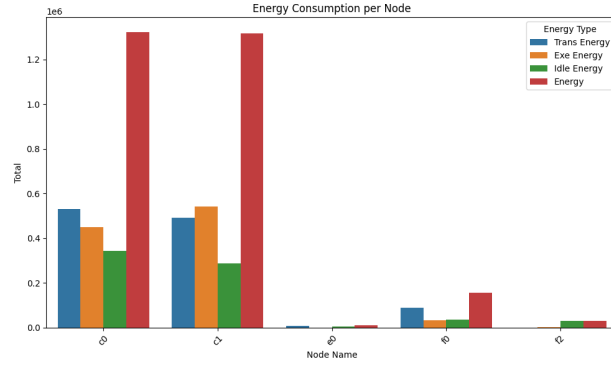Figure 18: Average latency per network link for T-NOTE

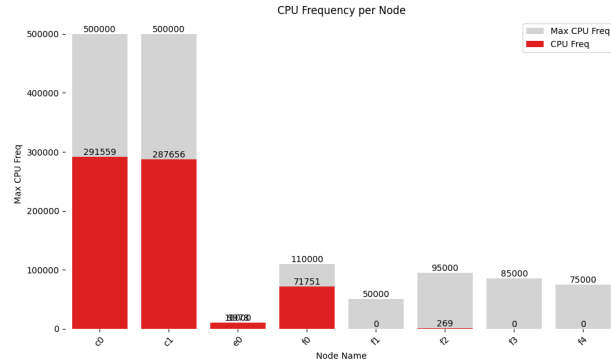Figure 19: Energy consumption distribution for T-NOTE

Figure 20: CPU frequency utilization per node type for T-NOTE

The energy consumption and CPU utilization patterns shown in Figures 19 and 20 illustrate that cloud nodes continue to dominate in both power usage and computational utilization. This dominance aligns with the high resource availability at cloud level and the reward function's emphasis on minimizing task failures and processing latency. However, the partial utilization of fog nodes represents a strategic improvement over the MLP approach, as it helps mitigate congestion at the cloud level while maintaining overall system performance.

28

The T-NOTE's adaptive decision-making capability enables it to recognize scenarios where distributed processing across multiple node types provides superior outcomes compared to pure cloud-centric strategies. This flexibility becomes particularly valuable under varying network conditions and diverse task requirements, demonstrating the architecture's potential for real-world deployment scenarios where computational demands and network characteristics exhibit significant temporal and spatial variation.

5.7. Energy Consideration

To highlight the performance of the proposed DRL-based approaches with respect to energy consumption, we compare T-NOTE, our best model, with the heuristic baselines, Random and Round Robin, as shown in Table 7. By appropriately adjusting the reward weights to prioritize energy efficiency, T-NOTE achieves 40 times lower TTR and 3.5 times lower latency than the heuristic baselines, while maintaining comparable energy consumption. This demonstrates that T-NOTE can be adapted to different optimization objectives through reward weight tuning, highlighting the effectiveness and flexibility of the proposed DRL-based approach in optimizing task offloading across hierarchical edge-fog-cloud environments.

Offloading Strategies	TTR (%)	Latency (s)	Energy (W)
Random	14.21	20.89	238.3
Round Robin	14.98	18.36	239.9
T-NOTE	**0.3556**	**5.976**	**238.0**

Table 7: Performance comparison of T-NOTE with energy-optimized reward weights against heuristic baselines on Pakistan test set. Best results are in **bold**.

5.8. GA-based Results and Comparison

Both the GA-based (NSGA II and NPGA) and DQL methods can train a MLP with 6,664 trainable parameters, using the same input features (CPU, buffer, and bandwidth). However, each approach yields distinct convergence properties and solution spaces.

5.8.1. Convergence Behavior

GA-based. The GA-based approach is more sensitive to initial population generation, and convergence behaviors can vary widely with different random seeds. As shown in Figure 21 for NSGA II, wich is almost similar to NPGA, training often exhibits fluctuations when searching for Pareto-optimal solutions, especially in the early stages of evolution. Although the algorithm eventually discovers high-quality solutions, it may require many generations (e.g., 100 generations with 40 individuals each) to achieve consistency.
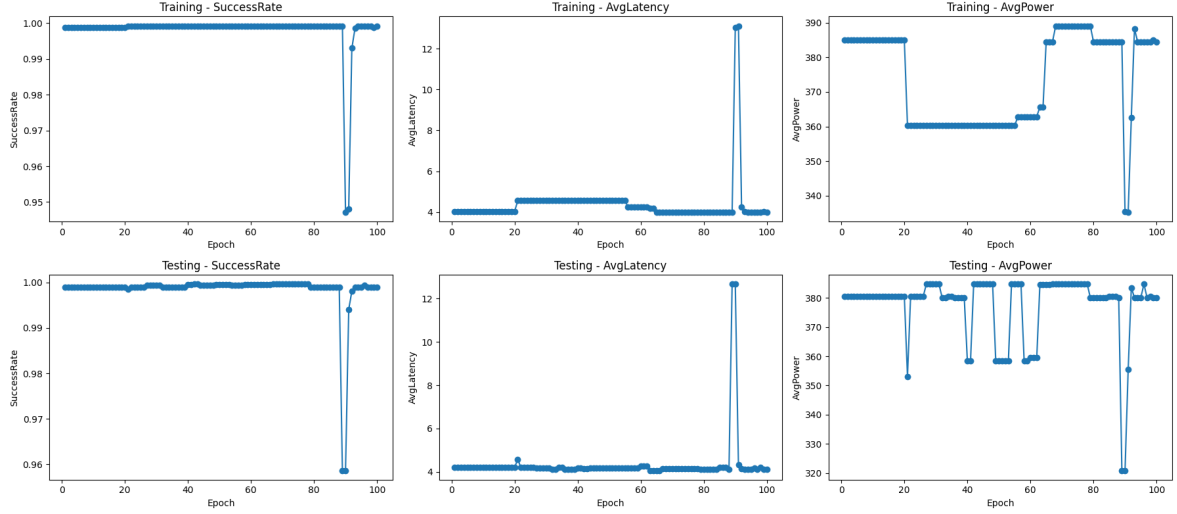
Figure 21: Training Convergence for MLP with NSGA2 for the best individual, selected according to $\lambda_0 \gg \lambda_1 \gg \lambda_2$.

Offloading Strategies	TTR (%)	Latency (s)	Energy (W)
NSGA2 + MLP	<u>0.07</u>	4.16	385
NPGA + MLP	0.14	4.24	380
DQL + MLP	<u>0.07</u>	4.16	385

Table 8: Comparison of MLP with DQL, NPGA, and NSGA2 on Pakistan test set.

Performance.

Trade-Off in Efficiency. A key advantage of DQL is the reduced computational burden—only one model is trained and updated across epochs (e.g., 10–20 epochs), compared to GA approaches that must evaluate a large population (e.g., 40 individuals) for multiple generations (e.g., 100). Hence, GA methods generally consume more time hours despite their ability to track multiple objectives.

5.8.2. Solution Space and Pareto Frontiers

Although DQL is effectively treating the problem as a single-objective function (via the weighted sum of different metrics in the reward), multi-objective GA methods (e.g., NSGA2, NPGA) discover a *set* of Pareto-optimal solutions. This advantage is illustrated in Figures 22 and 23, where multiple configurations of (latency, energy, throw rate) emerge along the final Pareto front. Practitioners can then choose the solution that best fits their operational needs (e.g., emphasizing latency more strongly or focusing on energy savings).

DQL produces a single high-performing policy, which in practice falls somewhere within or near the Pareto front discovered by GA. Experimental results indicate that the DQL policy is generally comparable to those in the GA's Pareto set. However, users requiring fine-grained trade-offs among metrics may prefer GA-based multi-objective solutions.
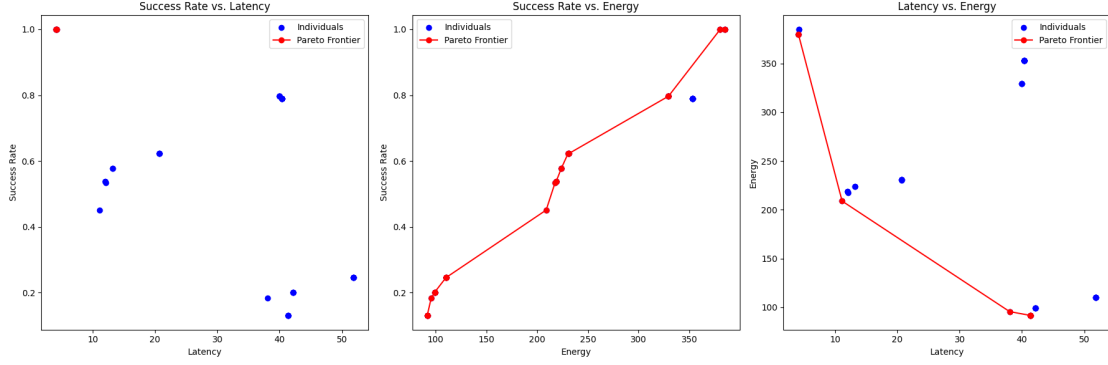
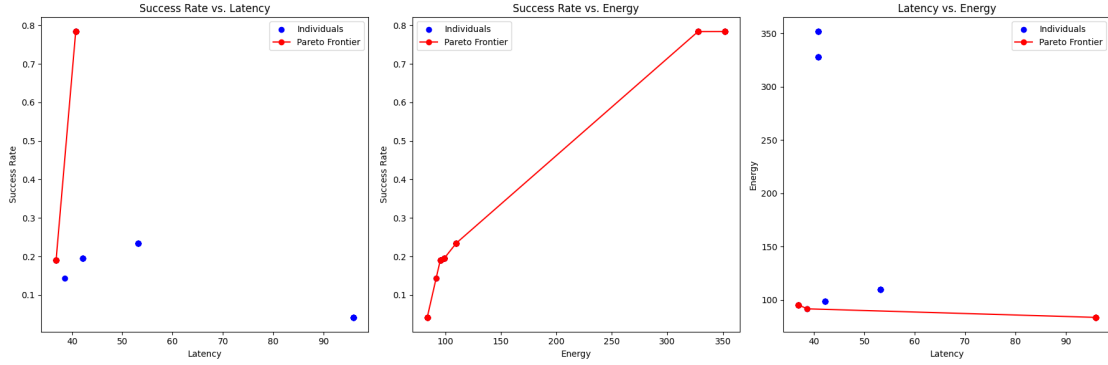Figure 22: Pareto Frontiers Discovered by NSGA2 (MLP Genome)

Figure 23: Pareto Frontiers Discovered by NPGA (MLP Genome)

Why GA Is Not Considered for Transformers. Given the high dimensionality of Transformer parameters (approximately 350k), evolutionary algorithms would face immense difficulty converging within reasonable time and computational budgets. Hence, the DQL approach is used for Transformer-based models, exploiting gradient descent to handle large parameter counts more efficiently.

Discussion. The comparative analysis reveals distinct advantages and limitations for each approach. The genetic algorithm demonstrates superior capability in generating a diverse range of Pareto-optimal solutions, thereby providing greater flexibility in objective weighting and multi-criteria optimization scenarios. In contrast, the deep Q-reinforcement learning approach exhibits faster convergence characteristics, rapidly converging to a single high-quality policy when hyperparameters are appropriately configured, while demonstrating reduced sensitivity to initial random seed selection. However, both methodologies present notable limitations that warrant consideration. The GA-based training approach incurs significant computational overhead, potentially limiting its applicability in resource-constrained environments. Conversely, the DQL framework is susceptible to local optima entrapment when learning rates or exploration parameters are suboptimally configured, which can substantially compromise solution quality and training stability.

6. Conclusion and Perspectives

This work addressed the critical challenge of task offloading in hierarchical edge-fog-cloud computing environments through a comprehensive investigation of metaheuristic and deep reinforcement learning approaches. A realistic evaluation framework was established using real IoT task traces from Pakistan, inte-

grated within the RayCloudSim simulation platform, and made publicly available to facilitate reproducible research and community advancement.

The proposed NOTE and T-NOTE architectures demonstrated the effectiveness of Transformer-based approaches for task offloading optimization. T-NOTE, which incorporates both node-level and task-level observations through attention mechanisms, achieved superior performance across all evaluated metrics compared to baseline methods. Specifically, T-NOTE eliminated task failures entirely (0% TTR) while achieving 16.6% lower latency and 6.4% reduced energy consumption compared to the Greedy baseline. These results underscore the value of attention mechanisms in capturing complex relationships between heterogeneous node resources and diverse task requirements.

The comparative analysis between genetic algorithms and deep Q-learning revealed complementary strengths. Multi-objective GA methods (NSGA-II, NPGA) successfully generated diverse Pareto-optimal solution sets, providing practitioners with flexible trade-off options among competing objectives. However, these methods incurred substantial computational overhead due to population-based evaluation across multiple generations. In contrast, DQL-based approaches demonstrated faster convergence and lower computational requirements while achieving competitive performance, making them particularly suitable for scenarios requiring rapid deployment and real-time adaptation.

The experimental results highlighted that task-aware Transformer architectures can effectively balance multiple competing objectives—latency minimization, energy efficiency, and reliability—through appropriate reward weight tuning. The demonstrated flexibility of T-NOTE in adapting to different optimization priorities (as shown in the energy consideration experiments) validates the practical applicability of the proposed approach across diverse deployment scenarios and operational requirements.

6.1. Perspectives

Several promising research directions emerge from this work that warrant further investigation:

Advanced Reinforcement Learning Architectures. Future work should explore policy gradient methods such as Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C), and Deep Deterministic Policy Gradient (DDPG), which may offer improved stability and sample efficiency compared to value-based DQL approaches. Additionally, investigating hybrid architectures that combine the exploratory capabilities of evolutionary algorithms with the gradient-based optimization of deep learning could yield more robust solutions.

Multi-Objective Optimization. A more extensive investigation of multi-objective genetic algorithms could provide valuable insights into the trade-offs between different QoS metrics. Developing lightweight variants of GA suitable for high-dimensional parameter spaces would enable the discovery of comprehensive Pareto frontiers for complex neural architectures.

Scalability and Generalization. Evaluating the proposed approaches on larger-scale networks with heterogeneous IoT devices, diverse application domains, and varying traffic patterns would validate their scalability and generalization capabilities. Transfer learning techniques could be investigated to enable pre-trained models to adapt rapidly to new deployment environments with minimal retraining.

References

Aazam, M., Islam, S.U., Lone, S.T., Abbas, A., 2022. Cloud of Things (CoT): Cloud-Fog-IoT Task Offloading for Sustainable Internet of Things. IEEE Transactions on Sustainable Computing 7, 87–98. URL: https://ieeexplore.ieee.org/document/9214500/, doi:10.1109/TSUSC.2020.3028615.

Adhikari, M., Mukherjee, M., Srirama, S.N., 2020. DPTO: A Deadline and Priority-Aware Task Offloading in Fog Computing Framework Leveraging Multilevel Feedback Queueing. IEEE Internet of Things Journal 7, 5773–5782. URL: https://ieeexplore.ieee.org/document/8863944/, doi:10.1109/JIOT.2019.2946426.

Benaboura, A., Bechar, R., Kadri, W., . A comprehensive survey of task offloading techniques in IoT-Fog-Cloud computing .

Bernard, L., Yassa, S., Alouache, L., 2024. D-NPGA : a new approach for tasks offloading in fog/cloud environment. URL: https://ieeexplore.ieee.org/abstract/document/10708605/authors, doi:10.1109/CoDIT62066.2024.10708605. iSSN: 2576-3555.

Bukhari, M.M., Ghazal, T.M., Abbas, S., Khan, M.A., Farooq, U., Wahbah, H., Ahmad, M., Adnan, K.M., 2022. An Intelligent Proposed Model for Task Offloading in Fog-Cloud Collaboration Using Logistics Regression. Computational Intelligence and Neuroscience 2022, 1–25. URL: https://www.hindawi.com/journals/cin/2022/3606068/, doi:10.1155/2022/3606068.

Das, R., Inuwa, M.M., 2023. A review on fog computing: Issues, characteristics, challenges, and potential applications. Telematics and Informatics Reports 10, 100049. URL: https://linkinghub.elsevier.com/retrieve/pii/S2772503023000099, doi:10.1016/j.teler.2023.100049.

Deb, K., Agrawal, S., Pratap, A., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans. Evol. Comput. 6, 182–197. URL: https://api.semanticscholar.org/CorpusID:9914171.

Fahimullah, M., Ahvar, S., Trocan, M., 2022. A Review of Resource Management in Fog Computing: Machine Learning Perspective. URL: http://arxiv.org/abs/2209.03066, doi:10.48550/arXiv.2209.03066. arXiv:2209.03066 [cs].

Gholipour, N., Assuncao, M.D.d., Agarwal, P., gascon samson, j., Buyya, R., 2023. TPTO: A Transformer-PPO based Task Offloading Solution for Edge Computing Environments. URL: http://arxiv.org/abs/2312.11739, doi:10.48550/arXiv.2312.11739. arXiv:2312.11739 [cs].

Google, 2024. Google Data Center Locations. https://www.google.com/about/datacenters/locations/. Accessed: 2024-03-23.

Guo, L., Lin, J., Xu, X., Li, P., 2024. Algorithmics and Complexity of Cost-Driven Task Offloading with Submodular Optimization in Edge-Cloud Environments. URL: http://arxiv.org/abs/2411.15687, doi:10.48550/arXiv.2411.15687. arXiv:2411.15687 [cs] version: 1.

Hendrycks, D., Gimpel, K., 2023. Gaussian error linear units (gelus). URL: https://arxiv.org/abs/1606.08415, arXiv:1606.08415.

Horn, J., Nafpliotis, N., Goldberg, D., 1994. A niched pareto genetic algorithm for multi-objective optimization, pp. 82 – 87 vol.1. doi:10.1109/ICEC.1994.350037.

Iftikhar, S., Gill, S.S., Song, C., Xu, M., Aslanpour, M.S., Toosi, A.N., Du, J., Wu, H., Ghosh, S., Chowdhury, D., Golec, M., Kumar, M., Abdelmoniem, A.M., Cuadrado, F., Varghese, B., Rana, O., Dustdar, S., Uhlig, S., 2023. AI-based Fog and Edge Computing: A Systematic Review, Taxonomy and Future Directions. Internet of Things 21, 100674. URL: http://arxiv.org/abs/2212.04645, doi:10.1016/j.iot.2022.100674. arXiv:2212.04645 [cs].

Ismail, L., Materwala, H., 2021. Computing Server Power Modeling in a Data Center: Survey, Taxonomy, and Performance Evaluation. ACM Computing Surveys 53, 1–34. URL: https://dl.acm.org/doi/10.1145/3390605, doi:10.1145/3390605.

Jazayeri, F., Shahidinejad, A., Ghobaei-Arani, M., 2021. Autonomous computation offloading and auto-scaling the in the mobile fog computing: a deep reinforcement learning-based approach. Journal of Ambient Intelligence and Humanized Computing 12, 8265–8284. URL: https://link.springer.com/10.1007/s12652-020-02561-3, doi:10.1007/s12652-020-02561-3.

Jiang, F., Ma, R., Gao, Y., Gu, Z., 2021. A reinforcement learning-based computing offloading and resource allocation scheme in F-RAN. EURASIP Journal on Advances in Signal Processing 2021, 91. URL: https://asp-eurasipjournals.springeropen.com/articles/10.1186/s13634-021-00802-x, doi:10.1186/s13634-021-00802-x.

Jin, X., Zhang, S., Ding, Y., Wang, Z., 2024. Task offloading for multi-server edge computing in industrial Internet with joint load balance and fuzzy security. Scientific Reports 14, 27813. URL: https://www.nature.com/articles/s41598-024-79464-2, doi:10.1038/s41598-024-79464-2. publisher: Nature Publishing Group.

Mishra, K., Rajareddy, G.N.V., Ghugar, U., Chhabra, G.S., Gandomi, A.H., 2023. A Collaborative Computation and Offloading for Compute-Intensive and Latency-Sensitive Dependency-Aware Tasks in Dew-Enabled Vehicular Fog Computing: A Federated Deep Q-Learning Approach. IEEE Transactions on Network and Service Management 20, 4600–4614. URL: https://ieeexplore.ieee.org/document/10143987/, doi:10.1109/TNSM.2023.3282795.

Pakmehr, A., Gholipour, M., Zeinali, E., 2024. ETFC: Energy-efficient and deadline-aware task scheduling in fog computing. Sustainable Computing: Informatics and Systems 43, 100988. URL: https://linkinghub.elsevier.com/retrieve/pii/S2210537924000337, doi:10.1016/j.suscom.2024.100988.

Rahmani, A.M., Haider, A., Khoshvaght, P., Gharehchopogh, F.S., Moghaddasi, K., Rajabi, S., Hosseinzadeh, M., 2025. Optimizing task offloading with metaheuristic algorithms across cloud, fog, and edge computing networks: A comprehensive survey and state-of-the-art schemes. Sustainable Computing: Informatics and Systems 45, 101080. URL: https://linkinghub.elsevier.com/retrieve/pii/S2210537924001252, doi:10.1016/j.suscom.2024.101080.

Ren, Y., Sun, Y., Peng, M., 2021. Deep Reinforcement Learning Based Computation Offloading in Fog Enabled Industrial Internet of Things. IEEE Transactions on Industrial Informatics 17, 4978–4987. URL: https://ieeexplore.ieee.org/document/9183960/, doi:10.1109/TII.2020.3021024.

Sarkar, I., Kumar, S., 2022. Deep learning-based energy-efficient computational offloading strategy in heterogeneous fog computing networks. The Journal of Supercomputing 78, 15089–15106. URL: https://doi.org/10.1007/s11227-022-04461-z, doi:10.1007/s11227-022-04461-z.

Such, F.P., Madhavan, V., Conti, E., Lehman, J., Stanley, K.O., Clune, J., 2018. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. URL: https://arxiv.org/abs/1712.06567, arXiv:1712.06567.

Suryadevara, N.K., 2021. Energy and latency reductions at the fog gateway using a machine learning classifier. Sustainable Computing: Informatics and Systems 31, 100582. URL: https://linkinghub.elsevier.com/retrieve/pii/S2210537921000731, doi:10.1016/j.suscom.2021.100582.

Tu, Y., Chen, H., Yan, L., Zhou, X., 2022. Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning for Efficient Edge Computing in IoT. Future Internet 14, 30. URL: https://www.mdpi.com/1999-5903/14/2/30, doi:10.3390/fi14020030.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., 2023. Attention is all you need. URL: https://arxiv.org/abs/1706.03762, arXiv:1706.03762.

Wiesner, P., Thamsen, L., 2021. LEAF: Simulating large energy-aware fog computing environments, in: 2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC), pp. 29–36. doi:10.1109/ICFEC51620.2021.00012.

Yan, J., Bi, S., Zhang, Y.J., Tao, M., 2020. Optimal Task Offloading and Resource Allocation in Mobile-Edge Computing With Inter-User Task Dependency. IEEE Transactions on Wireless Communications 19, 235–250. URL: https://ieeexplore.ieee.org/document/8854339/, doi:10.1109/TWC.2019.2943563.

Zhang, R., Chu, X., Ma, R., Zhang, M., Lin, L., Gao, H., Guan, H., 2022. Osttd: Offloading of splittable tasks with topological dependence in multi-tier computing networks. IEEE Journal on Selected Areas in Communications .

Zhang, S., Yi, N., Ma, Y., 2024. A Survey of Computation Offloading with Task Type. URL: http://arxiv.org/abs/2401.01017, doi:10.48550/arXiv.2401.01017. arXiv:2401.01017 [cs] version: 3.