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) are 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 solution of GA-based + MLP methods successfully reaches the solution identified by the DQL + MLP approach, confirming the consistency of both methods. 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. 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.).

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 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).

Despite 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. 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 two transformer-based architectures, utilizing DQL for the purpose of task offloading within a cloud-fog-edge environment. The selection of the node (edge, fog or cloud) to execute each incoming task is determined by these models, representing n different actions. 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, we

implement an adaptation of a scenario with a real dataset within the framework of RayClousSim. We make the scenario and the dataset open-source for the community[1]. All algorithms used, including NOTE and T-NOTE, are also implemented in this framework.

- Additionally, we conduct a comparative analysis between GAs and a DQL approach. The implementation of these algorithms in the training of an 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 by Guo et al. (2024); Jin et al. (2024); Sarkar and Kumar (2022). Consequently, deterministic methods are subject to 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 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 approaches have also been increasingly used for their ability to address the NP-hard nature of task offloading. A recent and comprehensive survey by Rahmani et al. (2025) reviews a wide range of metaheuristic algorithms, including Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Grey Wolf Optimizer (GWO)—applied to task offloading in IoT environments. The review identifies key strengths of these methods in balancing latency, energy consumption, and cost efficiency across heterogeneous fog-edge-cloud infrastructures.

Multi-Objective GAs have also proven their worth. (Bernard et al., 2024) introduces 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. Multi-Objective GAs are highly efficient at finding a set of Pareto-optimal solutions, allowing decision-makers to select the most appropriate trade-off based on specific requirements.

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

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, the method uses 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.

Despite their strengths, some limitations in scalability and real-time adaptability persist, particularly in highly dynamic environments where task characteristics and network conditions frequently change. This has led to the exploration of AI-based methods, which can learn and adapt to such dynamics more effectively.

2.3. AI Approaches

Machine learning (ML) has recently demonstrated its efficacy in task offloading. For instance, decision tree classifiers (Suryadevara, 2021) determine whether tasks should be offloaded to the fog or cloud, resulting in reduced latency and energy consumption. Logistic regression models (Bukhari 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 dominance across a large subset of AI applications.

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. (2021) utilized DQL 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 methods (Ren et al., 2021) have also been employed to offload IoT tasks. Each IoT device owns a DRL model and trains it 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. (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. (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 DNNs (Sarkar and Kumar, 2022) and 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 models 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, not taking into account the resources allocation.

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/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 from computation to storage cost, passing by data the transfer costs.
L Latency	Latency is defined as the time delay between the initiation of a request and the reception of the response. This is a crucial consideration for real-time applications.
E Energy	The total energy consumed during task offloading, encompassing both device- and network-level energy utilisation.
ET Execution time	is defined as the total time taken to complete a task from start to finish, including both computation and communication time.
D Temporal discrepancy	The temporal discrepancy between the submission of a task and its commencement, as well as the temporal lag inherent in processing operations. It should be noted that these results are subject to the effects of network and processing delays.
M Make-span	The duration of time required for the completion of all tasks in a given batch or workflow. This index is indicative of the overall efficiency of the system.
QWT Queue waiting time	Refers to the duration for which a task remains in a waiting state within the queue prior to its processing. It has been demonstrated that impacts are associated with alterations to response time and throughput.
TTR Task throw rate	Defined as the rate at which tasks are successfully processed and completed within the system. This rate is a crucial metric for evaluating system throughput, providing a quantitative indication of the efficiency with which tasks are handled by the system.
DLV Deadline violation	Denote the percentage or rate of tasks that are not completed within the specified timeframe. This metric is indicative of the reliability of the system and the quality of the service provided.

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.

The study employs a real-world dataset of IoT-generated tasks to evaluate the performance of the proposed offloading strategies for real applications. The dataset encompasses various tasks collected from IoT devices. Each task is characterized by attributes such as size (in bytes), required CPU cycles, and deadline constraints. This dataset provides a realistic basis for assessing the effectiveness of offloading strategies in handling diverse and dynamic workloads typical of IoT environments.

3.1.1. Architecture

Figure 1 presents a schematic overview of the proposed three-tier architecture. The system comprises IoT devices (sensors, mobile nodes, and other edge devices) that generate tasks. These tasks subsequently reach the Global Gateway (GG), which processes them locally if resources are available or offloads them to the fog or cloud layers 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 possesses 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 each node, including CPU usage, memory availability, and buffer status.
- **Network conditions:** Bandwidth and potential congestion from the GG to fog or cloud nodes.
- **Task requirements** (T-NOTE only)**:** Task size, computational complexity, and deadline or quality of service (QoS) constraints.

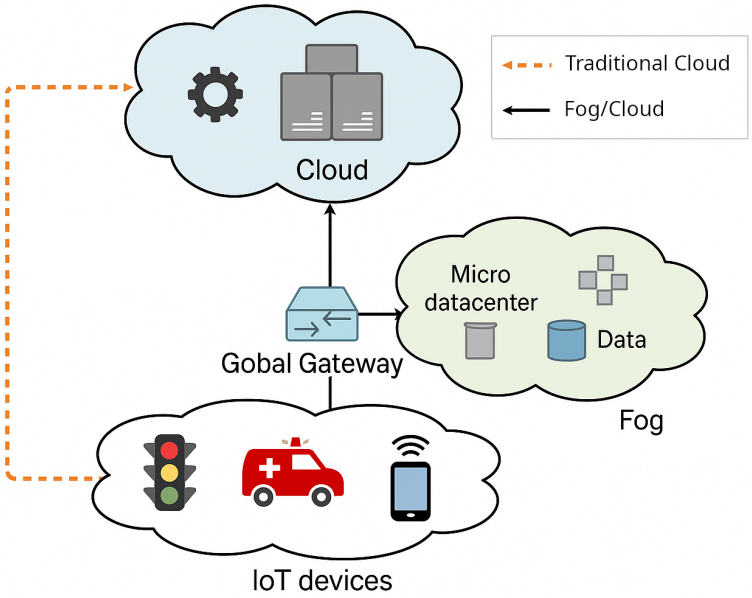

Figure 1: System Architecture

Based on these factors, an offloading decision is made to determine the optimal execution strategy for the task. The GG decides whether the task should be processed locally, offloaded to a fog node, or offloaded to a cloud node. Since the three-tier architecture includes multiple fog and cloud nodes, the decision also specifies the particular node to which the task should be sent. This decision-making process is dynamic and adaptive, responding to the current state of the network, available nodes and the task attributes.

Fog Nodes. Fog nodes provide moderate computational resources that exceed those of edge devices but remain below the capacity of cloud data centres. These intermediate computing nodes offer reduced latency in comparison to distant cloud nodes, due to their closer physical proximity to the network edge. Fog nodes function as a pivotal intermediate tier for tasks that, whilst not requiring the extensive resources of the cloud infrastructure, cannot be processed locally at the GG. This configuration renders them optimal for applications necessitating 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.

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 this modelisation, 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^i_{idle}): Energy drawn by node i when it remains idle, estimated through an idle power coefficient P_{idle}.
- **Execution Energy** ($E^{i,k}_{\text{exe}}$): Energy consumed by node i during the execution of task k, based on execution power P_{exe}.
- **Transmission Energy** ($E^{i,k}_{\text{trans}}$): 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:

$$E_{\text{total}} = \sum_{i=1}^{N} \Big(E^i_{\text{idle}} + \sum_{k=1}^{T} \big(E^{i,k}_{\text{exe}} + E^{i,k}_{\text{trans}} \big) \Big) \tag{9}$$

This formulation is intended to ensure that energy consumption measurements reflect only the computational and communication activities associated with successful task offloading. In this way, consistency with the latency metric definition is maintained, and a meaningful performance evaluation is provided.

Node Power Consumption Model. Based on the work of , 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.

Ismail and Materwala (2021) reported a Standard Error of Estimation (SEE) of 12.9% with this model, 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:

$$E^i_{\text{idle}} = \int_0^T P^i_{\text{idle}}\, dt = P^i_{\text{idle}} \cdot T. \tag{11}$$

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

$$E^i_{\text{exe}} = \int_0^T \big(P^i_{\text{exe}} - P^i_{\text{idle}} \big) \cdot u_{\text{cpu}}(t)\, dt. \tag{12}$$

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

$$E^{i,k}_{\text{exe}} = T^{i,k}_{\text{exe}} \cdot \big(P^i_{\text{exe}} - P^i_{\text{idle}} \big), \tag{13}$$

where $T^{i,k}_{\text{exe}}$ 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 provide 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

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

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.

As stated by (Horn et al., 1994), the *NPGA* employs a combination of tournament selection and a niching mechanism. This combination is used to maintain population diversity and to prevent premature convergence to a single region of the Pareto front. The selection process is initiated by the execution of tournaments between candidate solutions selected at random. The winner of these tournaments is determined based on Pareto dominance relationships within a comparison set that is also randomly chosen. Specifically, the competition is between two candidates who are tasked with counting how many individuals they dominate from a given subset of the population. The candidate with the higher dominance count is selected as the winner. This approach is predicated on the maintenance of diversity through the distribution of selection pressure across disparate regions of the Pareto front, thus obviating the necessity for 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 $\mathcal{P}^*$

```
Initialize population P_0 with N random MLP genomes
for g = 0 to G_max − 1 do
    Evaluate: For each individual χ_i ∈ P_g
        Simulate offloading with MLP weights from χ_i
        Compute objectives: f_L (latency), f_E (energy), f_TTR (timeout rate)
    Select: Create mating pool M_g using multi-objective selection
        (NSGA-II non-dominated sorting + crowding distance)
        (NPGA Pareto tournament selection)
    Reproduce: Generate offspring Q_g
    for i = 1 to N do
        Select parents χ_p, χ_q from M_g
        Apply crossover (prob. p_c) and mutation (prob. p_m)
        Add offspring to Q_g
    end for
    Replace: Form next generation P_{g+1} from P_g and Q_g
end for
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 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.

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

The MLP-based DQL reflects the MLP structure utilised in genetic algorithms. However, rather than undergoing weight evolution, it employs Q-learning to train the weights. The architecture comprises a feed-forward MLP that processes the current system state. This state includes metrics such as CPU, buffer, and bandwidth. The MLP outputs Q-values per node in order to guide the offloading decision. The task attributes (size, deadline, cycles/bit, and transmission bit rate) are also incorporated into the input features for a variant of MLP named T-MLP. 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. The process of batch updates is facilitated by the utilisation of a replay buffer, which serves to stabilise the training process and diminish the correlation among transitions. This, in turn, engenders 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.

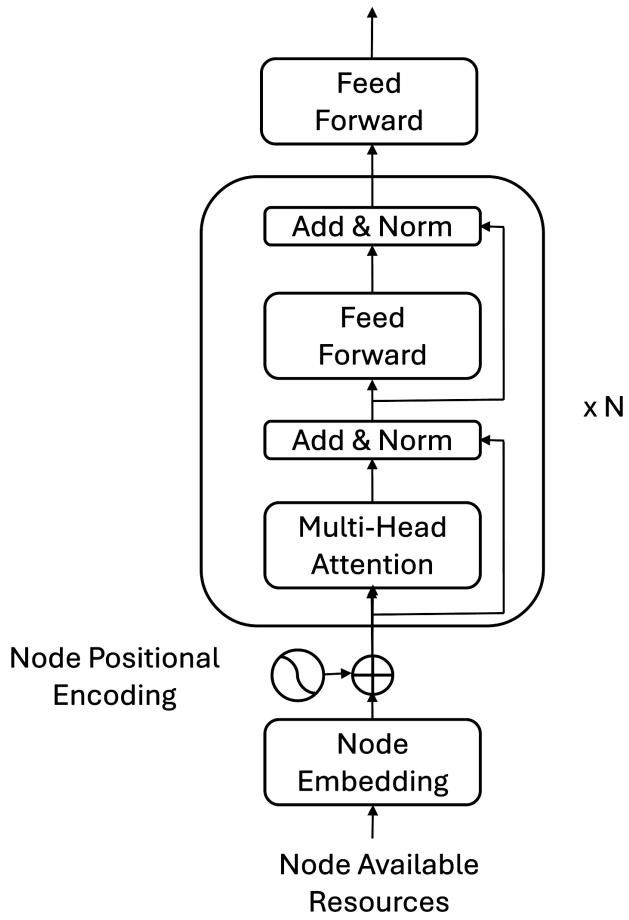

Figure 2: NOTE Architecture

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.

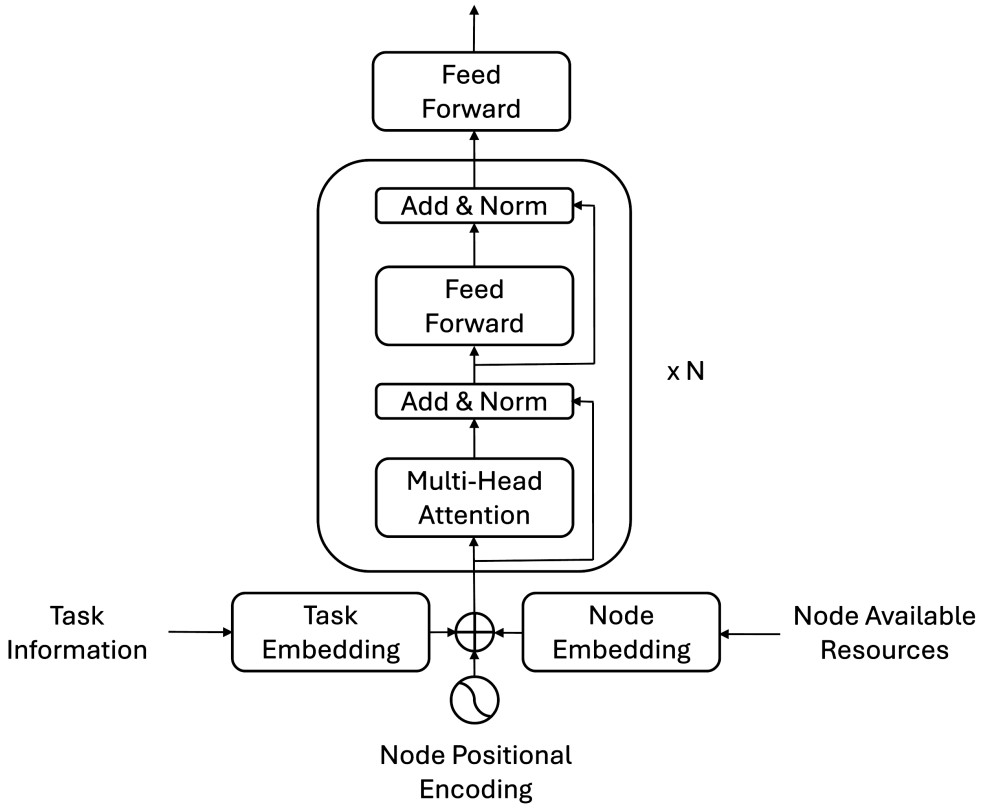

Figure 3: T-NOTE Architecture

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^*

Initialize Q-network Q_θ with random parameters θ
Initialize experience replay buffer $\mathcal{B} \leftarrow \emptyset$
Set exploration rate $\epsilon \leftarrow \epsilon_0$
for epoch $e = 1$ to $E_{\max}$ **do**
  Initialize epoch metrics: $L_{\max}^{(e)} \leftarrow 0$, $E_{\max}^{(e)} \leftarrow 0$
  for each task $\tau_i \in \mathcal{D}$ ordered by generation time **do**
    Wait until $t_{\text{current}} \geq \tau_i$.arrival_time
    Construct state vector s_i from system resources/task features
    Action Selection:
    if rand() $< \epsilon$ **then**
      $a_i \leftarrow$ random edge node
    else
      $a_i \leftarrow \arg\max_a Q_\theta(s_i, a)$
    end if
    Execute offloading action a_i and simulate task execution
    Observe next state s_{i+1} and compute reward:

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

    Store transition (s_i, a_i, r_i, s_{i+1}) in $\mathcal{B}$
    if $|\mathcal{B}| \geq$ batch_size **or** end of epoch **then**
      Sample mini-batch $\mathcal{M}$ from $\mathcal{B}$
      for each $(s, a, r, s') \in \mathcal{M}$ **do**
        Compute target: $y = r + \gamma \cdot \max_{a'} Q_\theta(s', a')$
      end for
      Update Q-network: $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$
      where $\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{M}}\left[(Q_\theta(s, a) - y)^2\right]$
      Decay exploration: $\epsilon \leftarrow \epsilon \cdot \epsilon_{\text{decay}}$
    end if
  end for
end for
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 DQLṪhese 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 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

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

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 are conducted on two distinct hardware configurations.

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

- CPU: 16 cores
- RAM: 64 GB
- GPU: NVIDIA Tesla P100 with 16 GB memory

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

GA Experiments.. The GA-based methods (NPGA, NSGA-II) are trained on a high CPU machine:

- CPU: 192 cores
- RAM: 256 GB

This setup enables 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 are sampled every minute over a one-hour period. To avoid clustering all tasks within the same second of each minute, tasks are 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 is 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—are 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.

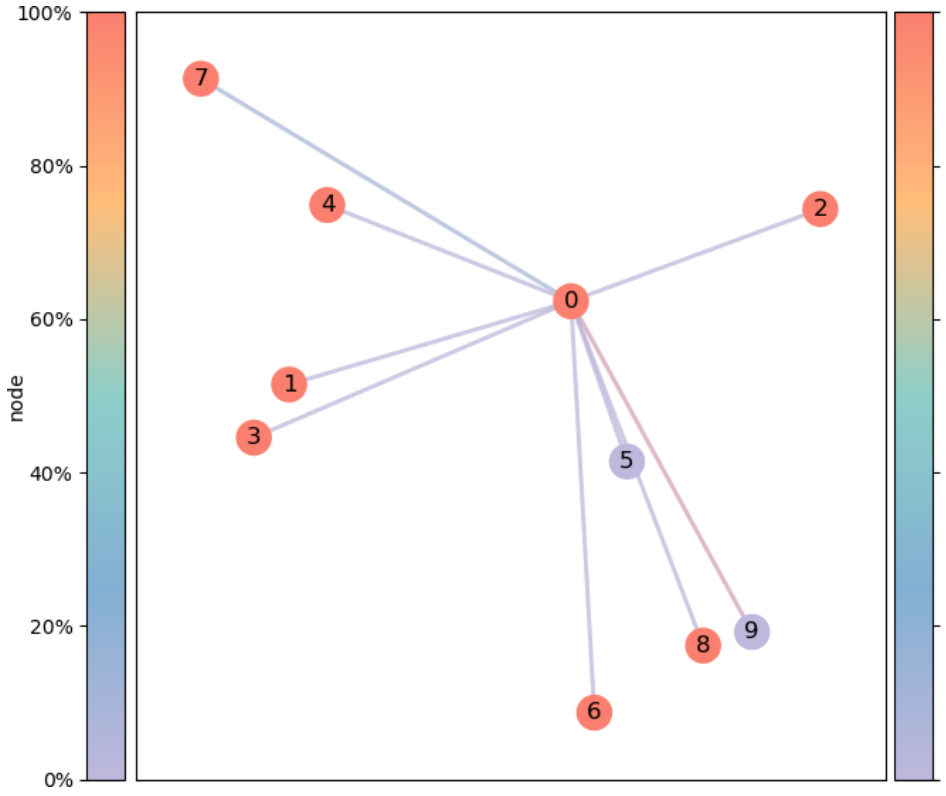

Figure 4: Used Architecture featuring Edge (Global Gateway), Fog (MDCs), and Cloud.

In Table 4, node specifications are listed, including Millions of Instructions Per Second (MIPS) which is the CPU frequency, 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.

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

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 throw rate*. This con-

straint 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 enhance the stability of the optimization process, a hierarchical prioritization scheme has been adopted, with a greater emphasis on latency in comparison to energy efficiency. This is due to the fact that latency exerts a more direct influence on both the user experience and the system's responsiveness. This prioritization is reflected in the reward weight hierarchy:

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

Based on empirical evaluation of different configurations using a MLP architecture trained with DQL, we selected the following reward weight distribution:

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

5.2.1. Model Configurations

The complete hyperparameter specifications are detailed in Table 5. These specifications guide the architecture and training of both the MLP and Transformer models under the DQL algorithm. Feature used as input include task attributes for T-models.

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} + {task_type}	

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

5.4. Evaluation

The evaluation of heuristic baseline methods reveals a fundamental trade-off between energy efficiency and performance, allowing us to categorize offloading strategies into two distinct classes: (i) *energy-centric* strategies (*Random* and *Round Robin*), which prioritize energy efficiency at the cost of elevated task failure rates and increased latency; and (ii) *performance-centric* strategies (*Greedy*), which substantially reduce task failures and latency while consuming more energy.

5.4.1. Performance-Centric Offloading Strategies

Given that the primary objective of this work is to minimize task failures, with latency optimization as a secondary goal, we adopt the Greedy method as our performance baseline. Accordingly, reward weights are configured as $[1, 0.1, 0.05]$ to prioritize task throw rate (TTR), followed by latency and energy consumption, as detailed in Section 5.2.

Table 6 presents a comprehensive comparison of performance-centric offloading strategies evaluated on the Pakistan test set.

Offloading Strategy	**TTR (%)**	**Latency (s)**	**Energy (W)**
Greedy	0.1889	4.394	338.7
MLP	0.1111	4.332	349.0
T-MLP	0.1889	4.768	346.1
NOTE	**0.0000**	4.150	337.6
T-NOTE	**0.0000**	**3.663**	**317.0**

Table 6: Performance comparison of offloading strategies measured by Task Throw Rate (TTR), latency (L), and energy consumption (E) on the Pakistan test set. Best results are shown in **bold**.

The Deep Q-Learning (DQL)-based approaches demonstrate substantial improvements over the Greedy baseline, with progressively refined architectures yielding incremental performance gains. The MLP strategy achieves a TTR of 0.1111%, representing a 41.2% reduction compared to Greedy, while concurrently reducing latency to 4.332 seconds (1.4% improvement). However, this improvement incurs a 3.0% increase in energy consumption (349.0 W versus 338.7 W), suggesting that the MLP's learned policy prioritizes reliability and responsiveness over energy efficiency. Incorporating task attributes into features, as illustrated by T-MLP, consequently degrades the performance of MLP, resulting in worse performance than Greedy.

The transformer-based NOTE architecture advances performance further by eliminating task failures entirely (0.0000% TTR) and reducing latency to 4.150 seconds (5.5% improvement over Greedy), while maintaining comparable energy consumption (337.6 W). This enhancement can be attributed to the transformer encoder's capacity to capture complex interdependencies among node observations through self-attention mechanisms.

By incorporating task-level observations alongside node characteristics, the T-NOTE architecture enables more contextually informed offloading decisions. This extension achieves superior performance across all metrics: maintaining zero task failures while attaining the lowest latency (3.663 seconds, representing a 16.6% improvement over Greedy) and the most efficient energy consumption (317.0 W, a 6.4% reduction compared to Greedy). These results demonstrate that task-aware attention mechanisms facilitate more effective resource allocation strategies in hierarchical edge-fog-cloud environments.

5.4.2. Energy-Optimized Performance

To evaluate the flexibility of the proposed DRL-based approach with respect to energy optimization, we compare T-NOTE against energy-centric heuristic baselines (Random and Round Robin) under appropriately adjusted reward weights that prioritize energy efficiency. Table 7 presents these results.

The heuristic baselines exhibit TTRs exceeding 14% and latencies above 18 seconds. These poor performance characteristics stem from their simplistic task assignment mechanisms, which distribute tasks

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

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

uniformly without consideration of node resource capacities. While this approach achieves low energy consumption (approximately 239 W) by utilizing a larger number of fog nodes 5.8.1, it fundamentally fails to meet performance requirements.

In contrast, the Greedy approach achieves a TTR of 0.1889% and latency of 4.394 seconds by preferentially selecting nodes with the highest available CPU resources 5.8.2, but increased energy consumption (338.7 W). This leads to the classification of performance-centric strategies.

When configured with energy-optimized reward weights, T-NOTE achieves a TTR approximately 40 times lower and latency approximately 3.5 times lower than the heuristic baselines, while maintaining comparable energy consumption (238.0 W). This result demonstrates that T-NOTE can be effectively adapted to diverse optimization objectives through reward weight configuration, highlighting both the effectiveness and flexibility of the proposed DRL-based approach for task offloading in hierarchical edge-fog-cloud environments.

5.5. MLP-Based DQL

With approximately 75,000 parameters, the MLP-based model maintains computational efficiency and suitability for resource-constrained environments. However, this architectural simplicity may limit the model's capacity to predict effective node assignments. T-MLP suggests that the architecture struggles to capture or generalize complex task-node interactions, potentially due to high variability in task features and the MLP's limited representational capacity. Indeed, the model exhibits poor generalization, as T-MLP obtains better results than MLP on the validation set but worse results on the test set, indicating overfitting.

Robustness and Performance. The MLP-based DQL approach demonstrates sensitivity to hyperparameter configurations, particularly learning rate and reward weight specifications. While capable of achieving improvements over heuristic baselines, the model exhibits less stable convergence behavior compared to transformer-based methods. This instability can be attributed, in part, to the MLP's reliance on fixed-size input vectors, which constrains its ability to adapt to varying system complexities.

Convergence Analysis. Figure 5 illustrates the convergence behavior of the MLP-based DQL model throughout training. Model selection is based on the highest average score, computed as a weighted combination of (TTR), latency (L), and energy consumption (E) metrics according to the specified reward weights.

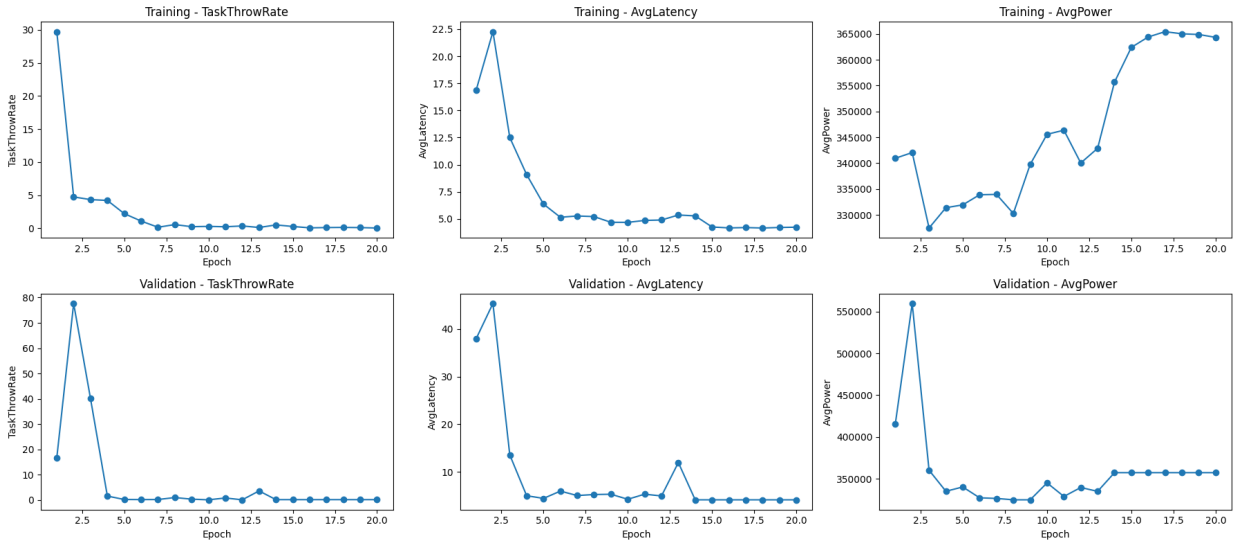

Figure 5: Training convergence and performance scores for MLP-based DQL approach.

The training process exhibits significant fluctuations during the initial four epochs, indicating instability in the learning dynamics. As training progresses beyond this initial phase, the model stabilizes, with scores gradually improving and oscillations diminishing until the optimal model is selected at epoch 10. This pattern suggests that while the MLP architecture can learn effective offloading policies, it requires careful hyperparameter tuning and extended training periods to achieve consistent performance.

5.6. Transformer-Based DQL

Transformer architectures, specifically NOTE and T-NOTE, are evaluated within the DQL framework. Despite their substantially larger parameter count (approximately 350,000 parameters, representing a 12-fold increase over the MLP), these models demonstrate superior empirical performance.

Robustness and Performance. The self-attention mechanisms inherent to NOTE and T-NOTE architectures exhibit greater robustness to hyperparameter variations, consistently converging to superior solutions that surpass MLP-based approaches. This finding aligns with prior work Gholipour et al. (2023), which suggests that transformer architectures can effectively capture nuanced task-node interactions through their attention mechanisms.

Convergence Analysis. The NOTE architecture encodes exclusively node-level features (CPU availability, buffer capacity, and bandwidth), aggregated across tasks. As illustrated in Figure 6, the learning process rapidly stabilizes and converges at epoch 4, where the optimal model is selected. This expedited convergence demonstrates the effectiveness of self-attention mechanisms in capturing and refining multi-node relationships.

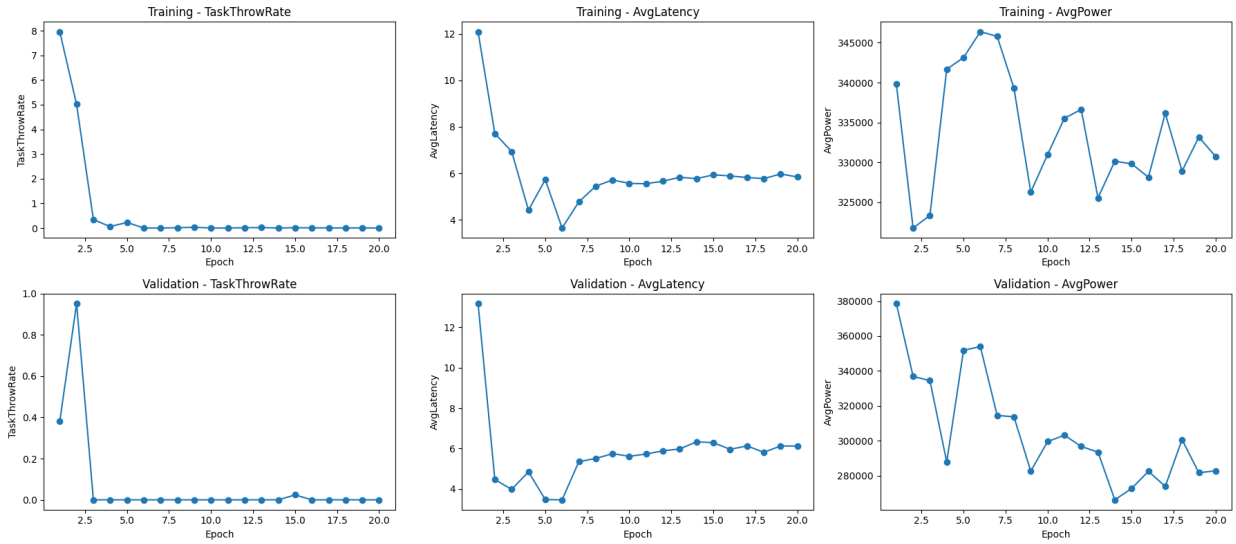

Figure 6: Training convergence and performance scores for NOTE-based DQL approach.

However, performance degradation occurs after epoch 4, potentially indicating convergence to a local optimum. This behavior suggests that while NOTE efficiently learns effective offloading policies, it may benefit from additional exploration strategies or more expressive feature representations. The incorporation of task-level features in T-NOTE effectively addresses this limitation.

The additional features, added by the task-level attributes, enable more contextually informed offloading decisions, resulting in improved convergence characteristics and superior performance with respect to both latency and task success rate. Figure 7 demonstrates the convergence behavior of T-NOTE throughout the training process.

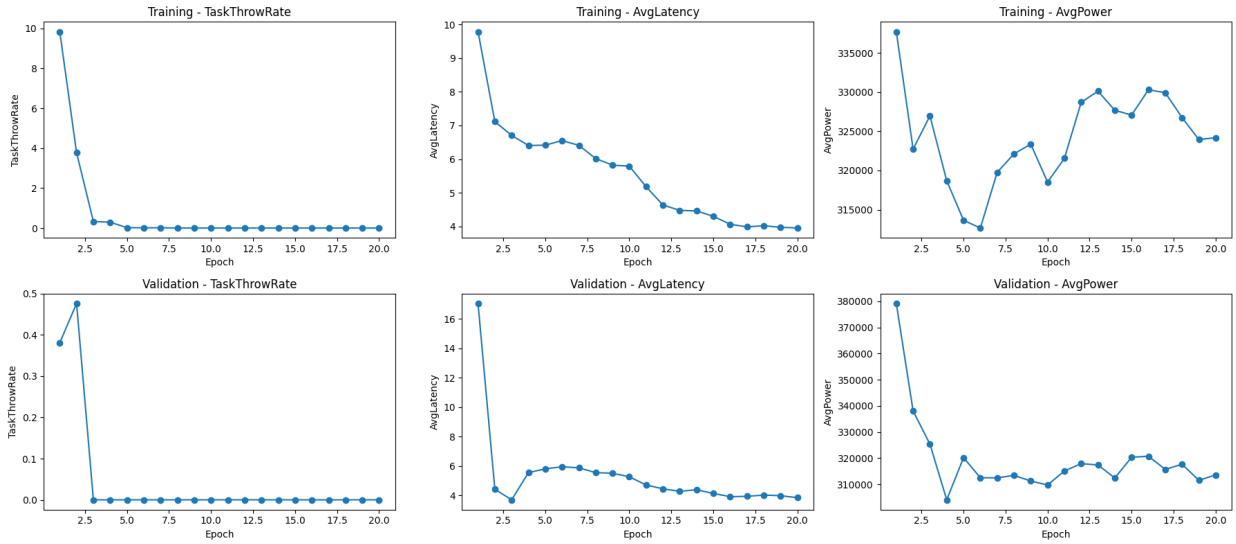

Figure 7: Training convergence and performance scores for T-NOTE-based DQL approach.

Similar to NOTE, T-NOTE exhibits rapid score improvement during the initial epoch, reflecting quick adaptation to the task offloading problem. However, unlike NOTE, which experiences performance degra-

dation after epoch 4, T-NOTE demonstrates continued exploration capabilities. The score exhibits a secondary descent from epoch 16 through the final training epoch. This behavior indicates that the model continues to refine its policy in pursuit of improved solutions, suggesting that the task-aware attention mechanism provides sufficient representational capacity to avoid premature convergence to local optima. This sustained exploration capability, combined with the model's ability to leverage task-specific information, enables T-NOTE to discover superior offloading strategies that achieve the best overall performance across all evaluation metrics during the test, as demonstrated in Table 6.

5.7. Discussion

The results demonstrate the effectiveness of DQL-based approaches for task offloading in hierarchical edge-fog-cloud environments. While computationally efficient, the MLP-based DQL model exhibits limitations in capturing complex task-node interactions, leading to less stable convergence and suboptimal performance compared to Transformer-based methods. In contrast, the NOTE and T-NOTE architectures leverage self-attention mechanisms to effectively model inter-node relationships and task-specific contexts, achieving superior performance across all evaluation metrics. However, this increased performance comes with higher computational complexity, especially for Transformer-based models, which may limit their applicability in resource-constrained environments.

DQL also requires careful reward weight tuning to achieve the desired trade-off between TTR, latency, and energy consumption. Reward weight tuning does not always follow a linear relationship with performance metrics or exhibit intuitive behavior, making it challenging to predict the exact impact of weight adjustments on overall performance. These problems can lead to unexpected outcomes, where small changes in weights result in significant performance variations, complicating the optimization process.

5.8. Offloading Analysis

This section analyzes the offloading strategies employed by different approaches, examining their performance characteristics, resource utilization patterns, and trade-offs. The analysis provides insights into how each method addresses task allocation challenges in hierarchical edge-fog-cloud computing environments.

Comparative evaluation reveals that *Transformer-based DQL* achieves superior robustness by optimally balancing latency minimization and task throw rate (TTR). While multi-objective genetic algorithm (GA) methods provide diverse Pareto-optimal solutions enabling flexible trade-off selection, MLP-based DQL offers computational efficiency as its primary advantage. The following subsections analyze node utilization patterns, performance metrics, and decision-making characteristics of the selected methodologies.

5.8.1. Random Baseline

The random offloading strategy serves as a baseline for evaluating sophisticated task allocation policies. By randomly assigning tasks to available nodes without awareness of resource capabilities, this approach leads to inefficient resource utilization, increased latency, and high task failure rates on low-end nodes, as shown in Figure 8.

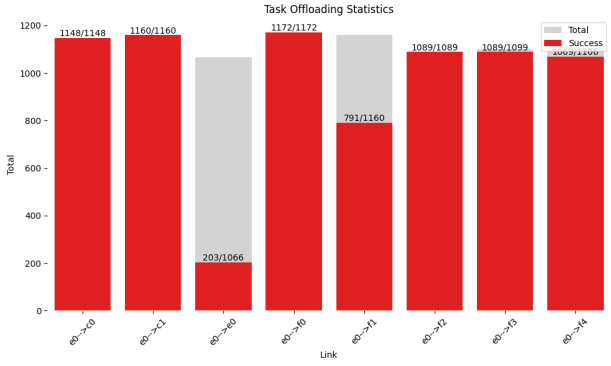

Figure 8: Task offloading distribution across node types for the random baseline.

Figure 9 illustrates power consumption across different nodes. Cloud nodes, with their higher computational capabilities, consume more power than edge and fog nodes for an equivalent number of tasks.

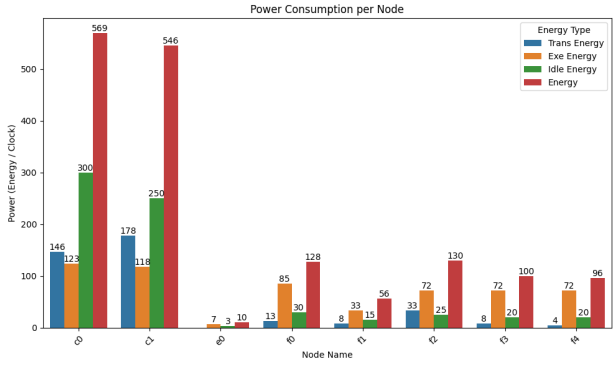

Figure 9: Power consumption per node for the random baseline.

5.8.2. Greedy Baseline

The greedy offloading strategy provides another fundamental baseline for evaluation. While its simplicity enables rapid decision-making, the method's myopic nature often leads to suboptimal global performance.

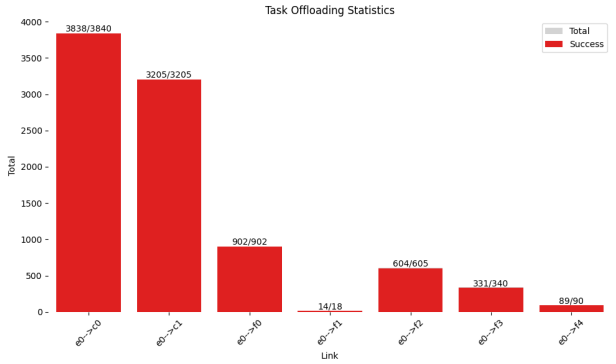

Figure 10: Task offloading distribution across node types for the greedy baseline.

Figure 10 shows the distribution of task assignments across edge, fog, and cloud nodes under the greedy policy. The results indicate a strong preference for nodes with higher CPU frequency, resulting in disproportionate allocation to cloud nodes and increased CPU resource demands, as illustrated in Figure 11.

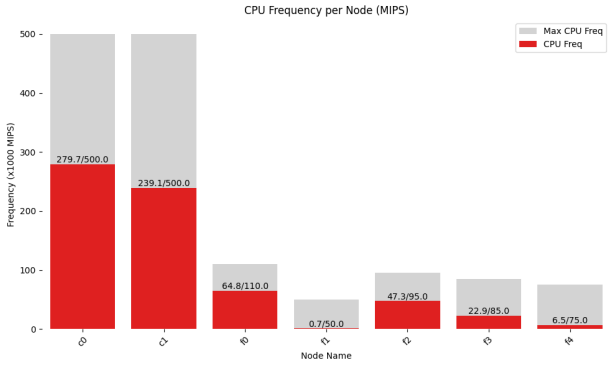

Figure 11: Average CPU frequency utilization per node for the greedy baseline.

The average communication latency per network link, depicted in Figure 12, shows transmission delays are nearly uniformly distributed across links, with slight increases for cloud node connections due to greater distances. These transmission delays, combined with queuing time at fog nodes whose processing capacity is insufficient to handle task flow, are the main contributors to overall task processing latency.

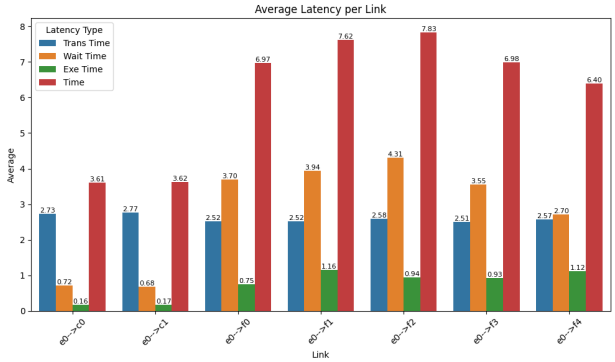

Figure 12: Average communication latency per network link for the greedy baseline.

Figure 13 presents power consumption across different nodes. Cloud nodes consume more power than edge and fog nodes for equivalent task volumes. This trend is particularly pronounced in the greedy strategy, which favors cloud offloading, leading to increased energy consumption at these nodes.

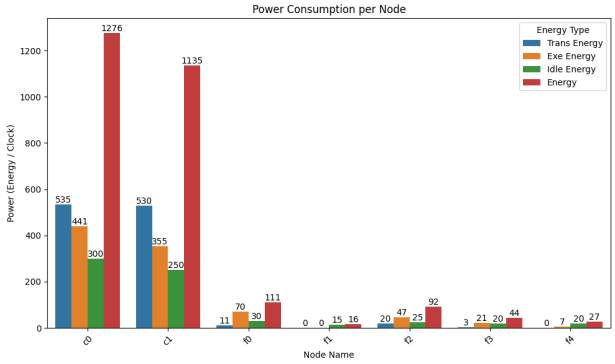

Figure 13: Power consumption distribution across node types for the greedy baseline.

5.8.3. MLP-Based DQL

As illustrated in Figure 14, the MLP-based policy exhibits a cloud-centric offloading strategy. This approach prioritizes cloud node utilization to minimize both 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 clear preference for leveraging superior computational resources in cloud nodes, which possess higher processing capabilities and more stable connectivity than edge and fog alternatives.

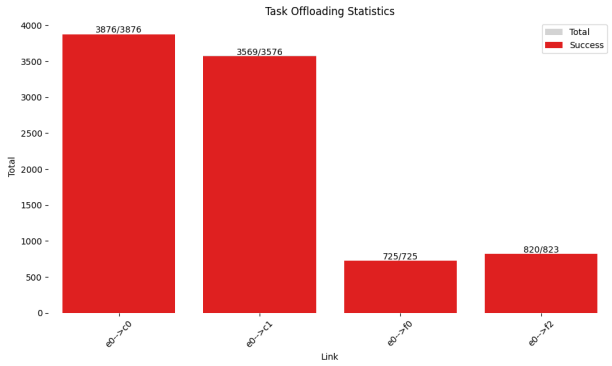

Figure 14: Task offloading distribution across node types for MLP-based DQL.

However, this strategy results in significant fog device underutilization, with minimal task offloading to this intermediate tier that could potentially provide balanced trade-offs between performance and energy efficiency. The heavy reliance on cloud infrastructure concentrates energy consumption at the cloud level, as shown in Figure 15.

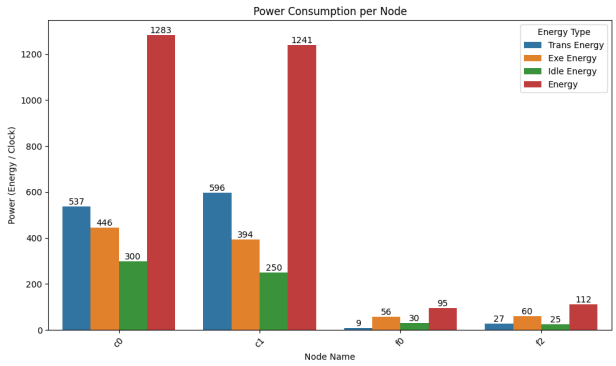

Figure 15: Power consumption distribution across node types for MLP-based DQL.

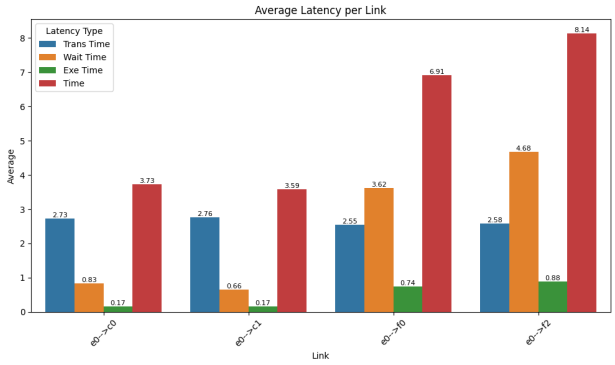

Figure 16: Average communication latency per network link for MLP-based DQL.

The performance characteristics in Figure 16 confirm the cloud-centric strategy's effectiveness in achieving low latency objectives while revealing limitations in energy distribution. Concentrating task processing at cloud nodes minimizes communication and processing latency but creates 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 maximizing a unified reward function rather than explicitly balancing multiple competing objectives.

5.8.4. NOTE

The NOTE architecture represents a significant advancement over traditional MLP-based approaches by incorporating explicit attention mechanisms for task attribute processing.

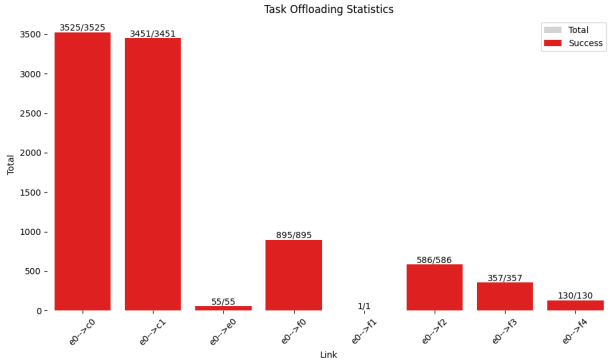

Figure 17: Task offloading distribution for NOTE

Even though NOTE have a similar task offloading distribution as Greedy, as shown in Figure 17, it demonstrates more sophisticated resource orchestration strategies since it the performance is significantly better than Greedy, as shown in Table 6. This highlights the effectiveness of the Transformer architecture in capturing complex relationships amoung the nodes and links of environement through its attention mechanism. The Figure 22 confirm this observation, showing an overall lower wait time across all nodes compared to Greedy, as shown in Figure 12.

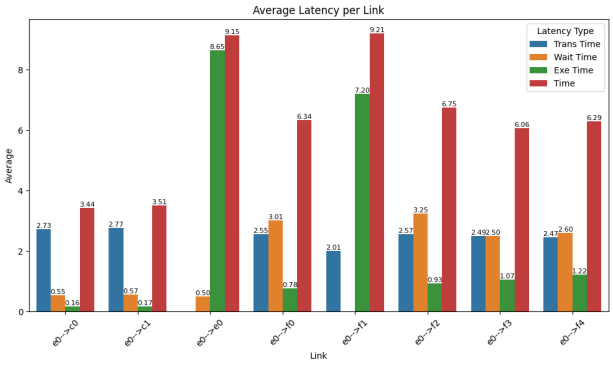

Figure 18: Average latency per network link for NOTE

The power consumption and the CPU frequency utilization across nodes is also similar to Greedy, as shown in Figure 19 and Figure 23, with intensive cloud utilisation and power consumption while other nodes steps back.

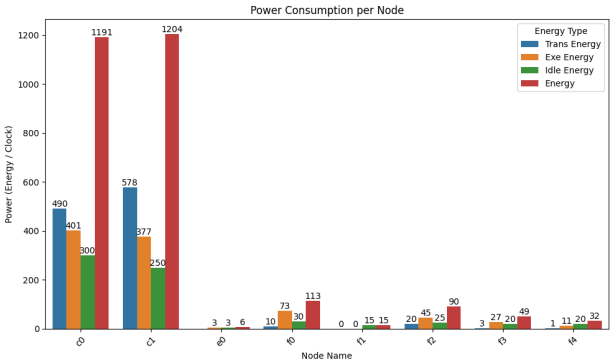

Figure 19: Power consumption distribution for NOTE

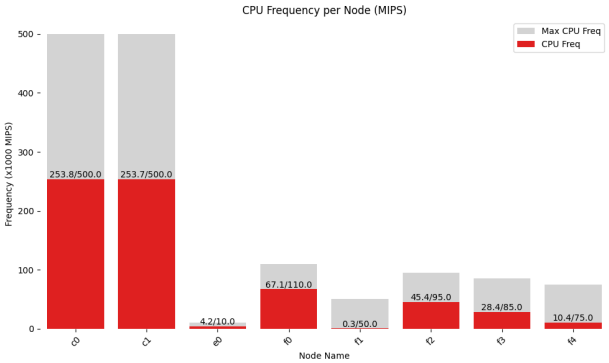

Figure 20: Average CPU frequency utilization per node for NOTE

5.8.5. T-NOTE

Incorporating task-level features in the T-NOTE architecture leads to surprising results in terms of offloading strategies. Tasks are predominantly offloaded to the edge, as shown in Figure 21, which differs

from other approaches since the edge has a lower CPU frequency. Tasks are also frequently offloaded to Fog Node 1, which has the lowest CPU frequency among fog nodes. By examining the node specifications in Table 4, we observe that these nodes have the highest MIPS/power consumption ratio, making them the most energy-efficient nodes.

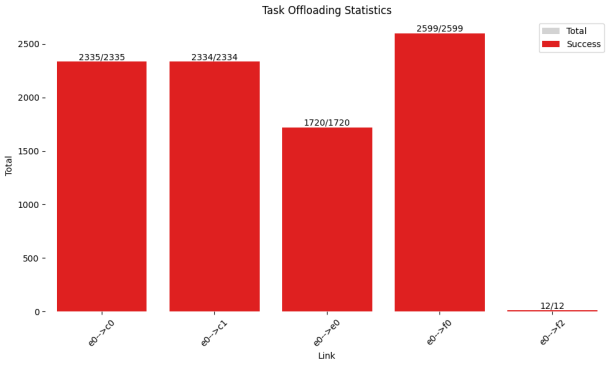

Figure 21: Task offloading distribution for T-NOTE

The cloud remains widely utilized but less so than in other approaches, with usage nearly equal to that of e0 and f1. This indicates that T-NOTE has learned to balance performance and energy efficiency by leveraging the most powerful and energy-efficient nodes.

Figure 22 demonstrates that while the edge experiences high wait times, this is compensated by the absence of transmission time since it is co-located at the GG.

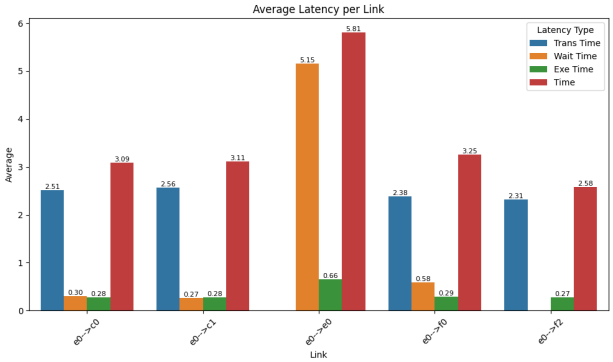

Figure 22: Average latency per network link for T-NOTE

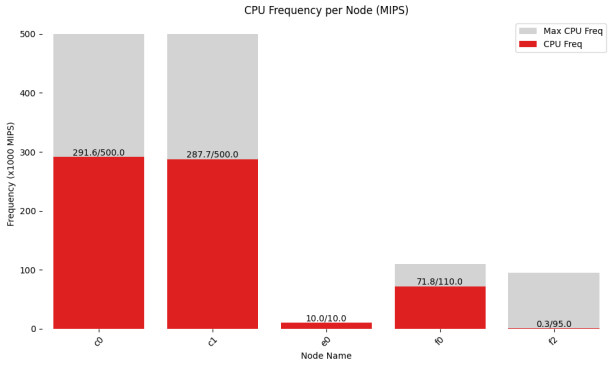

Figure 23: CPU frequency utilization per node type for T-NOTE

Figure 23 reveals higher MIPS utilization than other strategies despite a lower number of offloaded tasks. By calculating the ratio of CPU frequency used per task offloaded, we obtain 130 for cloud nodes, 5.9 for edge nodes, and 27.7 for Fog Node 1. This indicates that T-NOTE, through its awareness of task attributes, offloads low CPU-intensive tasks (characterized by low task size and low cycles per bit) to low CPU frequency but energy-efficient nodes, while directing high CPU-intensive tasks to high CPU frequency nodes. This behavior highlights the effectiveness of task-aware models.

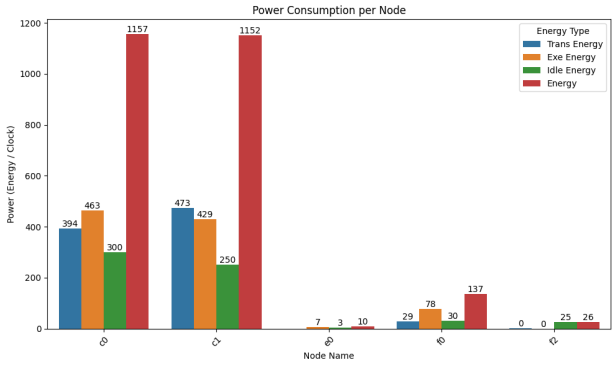

Figure 24: Power consumption distribution for T-NOTE

Figure 24 shows lower power consumption on cloud nodes compared to all performance-centric approaches, despite higher CPU frequency utilization. Examining the power consumption breakdown reveals a slight increase in execution power consumption (proportional to task size × cycles per bit) but a significant decrease in transmission power consumption (proportional to task size). This suggests that T-NOTE offloads tasks with a high cycles-per-bit-to-size ratio to the cloud, while routing other tasks to edge and fog nodes, which offer more efficient transmission power consumption.

5.9. GA-based Results and Comparison

As tuning reward weights in DQL is challenging, multi-objective optimization techniques offer a well-suited alternative to address this issue. To evaluate the effectiveness of these techniques in this context, we assess the performance of NSGA-II and NPGA for training the MLP architecture. This comparative analysis elucidates the strengths and limitations of multi-objective evolutionary strategies relative to DQL approaches in hierarchical edge-fog-cloud task offloading. Since GA-based methods are not primarily designed to handle high-dimensional DNNs models, both GA-based approaches (NSGA-II and NPGA) and DQL methods train a 2-layer MLP with a model dimension of 64, totaling 6,664 trainable parameters and utilizing identical input

features (CPU, buffer, and bandwidth). However, each approach yields distinct convergence properties and solution spaces.

5.9.1. Convergence Behavior

GA-based Approaches. GA-based approaches are more sensitive to initial population generation, with convergence behavior varying widely across different random seeds. As shown in Figure 25, which evaluates the best candidate according to reward weight priorities $\lambda_0 \gg \lambda_1 \gg \lambda_2$ for each generation, NSGA-II training (similar to NPGA) often exhibits fluctuations when searching for Pareto-optimal solutions, especially in early evolution stages. Although the algorithm eventually discovers high-quality solutions, it may require many generations (e.g., 100 generations with 40 individuals each) to achieve consistency.

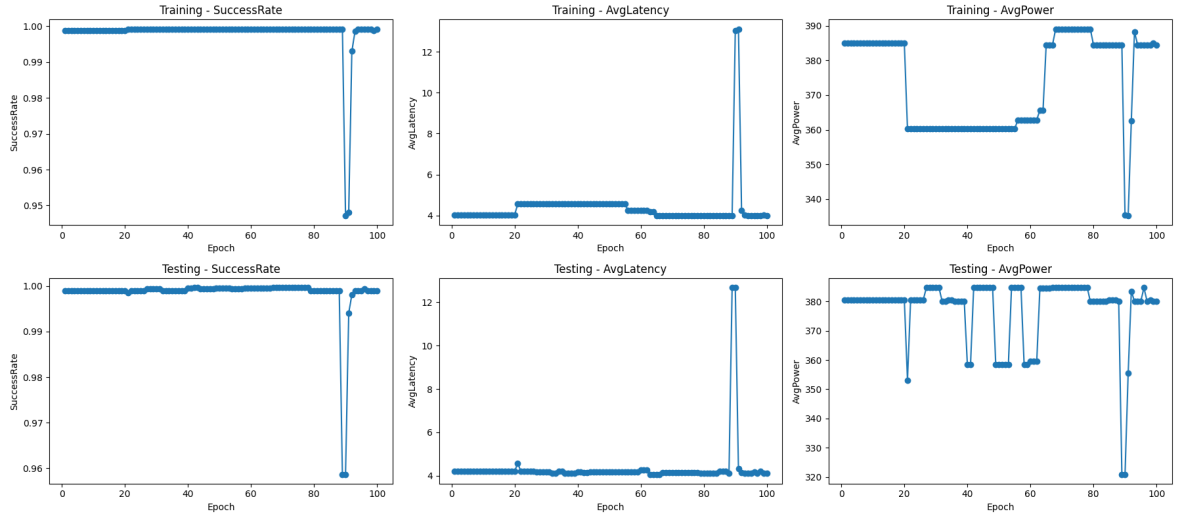

Figure 25: Training convergence for MLP with NSGA-II, showing the best individual selected according to $\lambda_0 \gg \lambda_1 \gg \lambda_2$.

Performance Comparison. Table 8 summarizes the performance of MLP models trained using DQL, NSGA-II, and NPGA. Both DQL and NSGA-II achieve identical performance metrics: TTR of 0.07%, latency of 4.16 seconds, and energy consumption of 385 W. NPGA performs slightly worse, with TTR of 0.14%, latency of 4.24 seconds, and energy consumption of 380 W. This indicates that while both DQL and NSGA-II effectively optimize the offloading policy, NPGA may struggle to find equally optimal solutions under identical conditions.

Offloading Strategy	**TTR (%)**	**Latency (s)**	**Energy (W)**
DQL + MLP	0.07	4.16	385
NSGA-II + MLP	0.07	4.16	385
NPGA + MLP	0.14	4.24	380

Table 8: Performance comparison of MLP with DQL, NPGA, and NSGA-II on the Pakistan test set. The best candidate from the multi-objective GA population is selected according to $\lambda_0 \gg \lambda_1 \gg \lambda_2$.

Computational Efficiency Trade-off. A key advantage of DQL is its reduced computational burden. Only one model is trained and updated across epochs (e.g., 10–20 epochs), compared to GA approaches that must evaluate large populations (e.g., 40 individuals) for multiple generations (e.g., 100). Consequently, GA methods generally require substantially more computational time despite their ability to optimize multiple objectives simultaneously.

5.9.2. Solution Space and Pareto Frontiers

While DQL effectively treats the problem as single-objective optimization (via weighted sum of different metrics in the reward function), multi-objective GA methods (e.g., NSGA-II, NPGA) discover a *set* of Pareto-optimal solutions. This advantage is illustrated in Figures 26 and 27, where multiple configurations of latency, energy, and throw rate emerge along the final Pareto front. Practitioners can then select solutions that best fit their operational requirements (e.g., emphasizing latency reduction or prioritizing energy savings).

DQL produces a single high-performing policy that, in practice, falls within or near the Pareto front discovered by GA. Experimental results indicate the DQL policy is generally comparable to those in the GA Pareto set. However, users requiring fine-grained trade-offs among metrics may prefer GA-based multi-objective solutions.

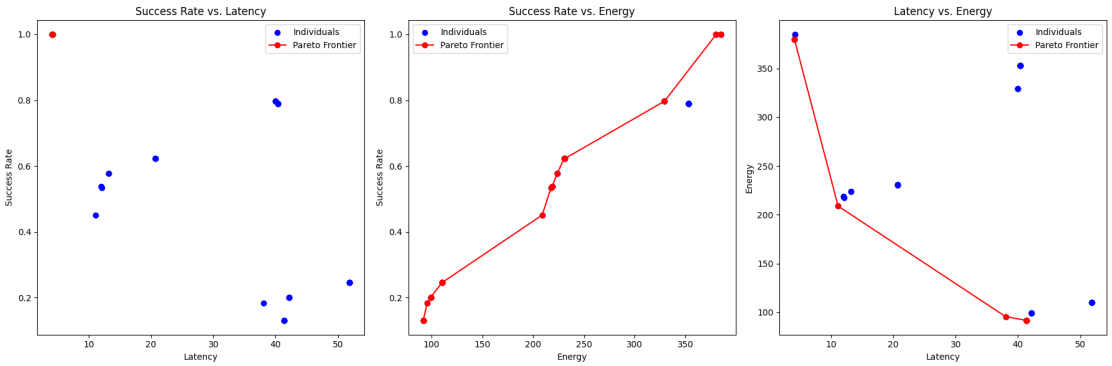

Figure 26: Pareto frontiers discovered by NSGA-II with MLP genome.

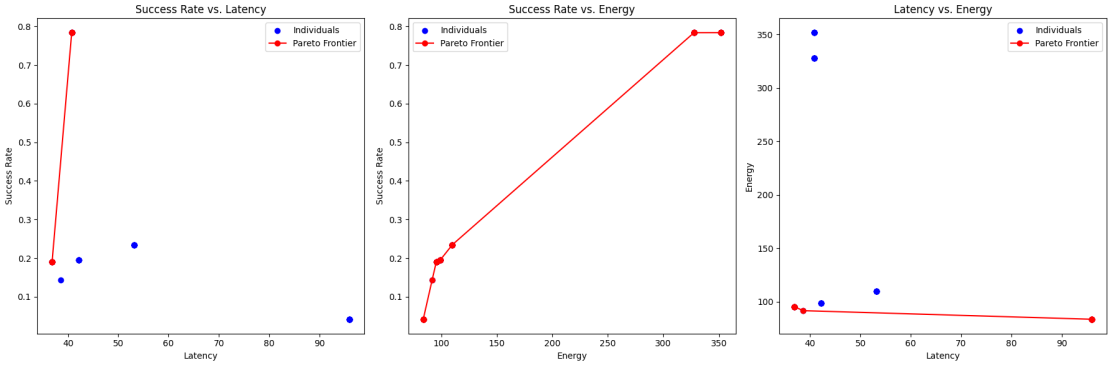

Figure 27: Pareto frontiers discovered by NPGA with MLP genome.

Inapplicability of GA for Transformer Architectures. Given the high dimensionality of Transformer parameters (approximately 350k), evolutionary algorithms would face immense difficulty converging within reasonable time and computational budgets. Therefore, DQL is employed for Transformer-based models, exploiting gradient-based optimization to handle large parameter counts more efficiently.

Discussion. The comparative analysis reveals distinct advantages and limitations for each approach. Genetic algorithms demonstrate superior capability in generating diverse Pareto-optimal solutions, providing greater flexibility in objective weighting and multi-criteria optimization scenarios. In contrast, deep Q-learning exhibits faster convergence, rapidly achieving 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. GA-based training incurs significant computational overhead, potentially limiting applicability in resource-constrained environments. Conversely, DQL 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 addresses 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 is established using real IoT task traces from Pakistan, integrated within the RayCloudSim simulation platform, and made publicly available to facilitate reproducible research and community advancement.

The proposed NOTE and T-NOTE architectures demonstrate the effectiveness of Transformer-based approaches for task offloading optimization. T-NOTE, which incorporates both node-level and task-level observations through attention mechanisms, achieves superior performance across all evaluated metrics compared to baseline methods. Specifically, T-NOTE eliminates 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 reveals complementary strengths. Multi-objective GA methods (NSGA-II, NPGA) successfully generate diverse Pareto-optimal solution sets, providing practitioners with flexible trade-off options among competing objectives. However, these methods incur substantial computational overhead due to population-based evaluation across multiple generations. In contrast, DQL-based approaches demonstrate 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 highlight 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

Reward weights tuning. Automatically tuning the reward weights would be a significant improvement to the DQL approach. This could involve meta-learning techniques or adaptive weight adjustment based on observed performance metrics during training. Automating this process would reduce the reliance on manual tuning and enhance the robustness of the offloading strategy across varying scenarios.

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.

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.

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 com-

puting 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.