International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Dynamic Energy-Aware Task Scheduling using Real-Time Resource Monitoring in Distributed Edge Environments

Vanapalli Jayanth Sai, Chadalavada Venkata Sai Nitin, Yelley Ankitha, Vobbilisetty Vandana Dedeepya
Student , Department of Computer Science and Engineering, College of Engineering
Andhra University, Visakhapatnam, India

Dr.Bharati Bidikar, Adjunct Professor, Department of Computer Science and Engineering,
College of Engineering, Andhra University, Visakhapatnam, India

Abstract : Edge computing environments support latency-sensitive applications but suffer from energy inefficiency and poor resource utilization due to dynamic workloads and heterogeneous nodes. Traditional scheduling methods such as Round Robin and heuristic-based approaches rely on static decisions, leading to increased energy consumption, higher latency, and uneven load distribution. This paper proposes a **Dynamic Energy-Aware Task Scheduling Framework** that combines real-time resource monitoring with a hybrid **CNN-LSTM model** to enable adaptive scheduling. The system continuously collects node-level telemetry (CPU, memory, temperature, energy) and predicts optimal node selection using a ranking-based approach refined by current load conditions. Experimental results show that the proposed method achieves **18–25% reduction in energy consumption**, **20–30% lower latency**, and **15–22% higher throughput**, while maintaining balanced resource utilization, demonstrating its effectiveness for intelligent edge scheduling.

Index Terms—*Edge Computing, Task Scheduling, Energy Awareness, CNN-LSTM, Real-Time Resource Monitoring, Distributed Systems, Deep Learning.*

I. Introduction

In the fast-moving field of distributed computing, edge computing has emerged as an important computing model in order to handle latency-sensitive and data-heavy operations. With increased adoption of IoT-enabled devices, smart infrastructure, and real-time analysis tools, the drawbacks of cloud-based systems, such as high latency, bandwidth issues, and processing overhead, have become apparent. Edge computing offers an effective solution to these issues by moving the computational process closer to the data source, which leads to a reduction in latency and improvement in QoS[1],[2]. Nonetheless, this approach comes with new problems related to the management of edge nodes.

One of the key problems in these types of systems is task scheduling in dynamic situations. Edge nodes work under constrained computing capabilities, unpredictable network environment, and dynamically changing loads; thus, the use of any form of static scheduling proves to be useless. Standard algorithms like the Round Robin method or any other heuristics approach do not allow adjusting tasks to the system state in real time, which results in higher latencies, low efficiency of resource usage, and overall imbalance of the system. Another important problem that arises is energy consumption as many edge devices are constrained in terms of energy resources.

In order to solve the aforementioned problems, this paper presents a Dynamic Energy-Aware Task Scheduling Framework that combines real-time measurement and hybrid CNN-LSTM modeling. The design extracts time-series data by means of LSTM[7] and spatial characteristics through CNN-based feature representation [8], thus allowing predicting optimal node selection. The problem is stated as optimization of a ranking objective that takes into account performance and energy criteria.

As demonstrated by experiments, the suggested framework achieves an improvement of 20%–30% in latency, 18%–25% savings in power consumption, and 15%–22% enhancement in throughput relative to conventional techniques. The principal innovations offered by this research comprise a real-time telemetry-based framework, a hybrid deep learning-driven scheduler, and an energy-aware task allocation scheme for distributed edge networks [10]–[14].

II. Problem Statement

Task scheduling in distributed edge computing is still a challenging issue because of the dynamic, heterogeneous, and resource-limited characteristics of edge devices. Scheduling techniques like static scheduling and heuristic methods like Round Robin and priority-based scheduling depend on the stability of system parameters and do not adjust based on changes in workloads, latency, and resources. Due to the static nature of these methods, they generate inefficient scheduling of tasks, increase latency, and cause imbalances in resource usage. Another issue with existing approaches is that they only pay attention to performance issues without considering energy usage. It is important to take energy consumption into account in edge computing systems since these systems operate under limited energy constraints. Real-time monitoring of the system's resources and predictive analytics should also be considered when developing such systems.

III. Related Work

A. Traditional Scheduling Methods

Traditional scheduling methods such as Round Robin (RR), First-In-First-Out (FIFO), and heuristic-based approaches are widely used due to their low computational overhead and simple implementation. In RR, tasks are assigned to nodes in a cyclic manner, ensuring equal distribution but ignoring real-time factors like CPU load, memory usage, and network delay, which can lead to assigning tasks to overloaded nodes and

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

increased latency. FIFO schedules tasks based on arrival order without considering priority or system state, causing delays for high-computation or time-sensitive tasks. Heuristic methods introduce predefined rules or cost functions to improve scheduling; however, these rules remain static and do not adapt to dynamic workload variations. As a result, these approaches fail to handle the heterogeneity and real-time variability of edge environments, leading to inefficient resource utilization and suboptimal performance [4], [5].

B. Energy-Aware Scheduling Systems

Energy-aware scheduling techniques focus on reducing power consumption during task execution. These methods commonly model energy as a function of CPU frequency, supply voltage, and execution duration. Dynamic Voltage and Frequency Scaling (DVFS) techniques adjust processor speed to conserve energy, while task consolidation methods reduce the number of active nodes. Although such approaches reduce energy usage, they rely on predefined models and do not incorporate real-time system behavior. Many energy-aware schedulers also prioritize energy reduction over performance, increasing task completion times [3], [6]. In edge systems, where both low latency and energy efficiency are critical, this imbalance constitutes a significant operational limitation.

C. Machine Learning–Based Scheduling

Machine Learning (ML) techniques have recently been applied to improve scheduling decisions through data-driven models. Reinforcement Learning (RL) approaches learn optimal scheduling policies through environmental interaction and reward maximization [9]. Advanced temporal models such as Long Short-Term Memory (LSTM) networks effectively capture time-series patterns in workloads, while Convolutional Neural Networks (CNNs) can identify spatial relationships between diverse node resources [7], [8]. Hybrid CNN-LSTM architectures have been demonstrated to improve scheduling accuracy by jointly modeling temporal and spatial dependencies [14]. Nevertheless, most ML-based solutions focus exclusively on prediction or performance improvement without incorporating energy consumption as a key decision factor, and most employ either temporal or spatial modeling independently, limiting their applicability in complex real-world edge environments [10], [11].

D. Research Gap

Analysis of existing literature reveals that current scheduling approaches address only specific aspects of the overall problem. Traditional methods lack adaptability to dynamic conditions; energy-aware techniques are not fully dynamic in their operation; and ML-based approaches are frequently incomplete in their design, lacking the integration of energy objectives. Critically, existing systems fail to unify real-time resource monitoring with predictive and energy-aware scheduling within a single closed-loop framework. There is therefore a clear and pressing need for an integrated approach that can leverage real-time telemetry, learn system behavior through hybrid deep learning, and simultaneously optimize both performance and energy consumption in distributed edge environments [12], [13].

IV. Proposed System Architecture

A. System Overview

The proposed system follows a distributed control-execution architecture in which the backend server functions as a centralized control plane and multiple node agents operate as execution units. The system is designed to perform real-time, energy-aware task scheduling through continuous telemetry collection and a machine learning-based decision engine. The backend is implemented using a FastAPI-based service that manages node registration, telemetry storage, task scheduling, and execution coordination. Each node in the system is represented using a structured data model containing hardware specifications including CPU cores, operating frequency, memory capacity, and system architecture, with time-series telemetry stored separately to capture dynamic system behavior.

The node agent is responsible for: (i) registering itself with the backend upon initialization; (ii) periodically transmitting telemetry and heartbeat signals at fixed intervals; (iii) maintaining a persistent WebSocket connection for receiving execution commands; and (iv) executing assigned tasks within isolated environments using Docker containers or local Python runtimes. The scheduling decision engine is implemented as a hybrid CNN-LSTM model that operates as a ranking function rather than a binary classifier, predicting a composite suitability score for each node-task pair to enable dynamic selection of the most efficient execution node.

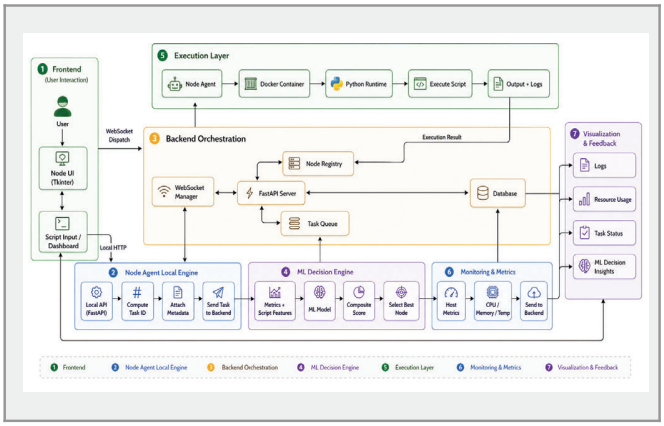

Fig. 1. Proposed system architecture illustrating the distributed control-execution design with backend scheduler, node agents, telemetry pipeline, and ML-based decision engine.

B. Real-Time Resource Monitoring

The system employs a continuous telemetry collection mechanism to maintain an up-to-date representation of each node's operational state. Each node agent transmits metrics at fixed 5-second intervals comprising CPU utilization, memory usage, temperature, and derived power consumption. These metrics are persisted as time-series data in the backend database. A sliding window of the 10 most recent observations is used to represent the current system state for each node, yielding a structured input tensor of shape (10×4) where each row represents a timestamp and each column corresponds to a monitored parameter: [CPU, Memory, Temperature, Power]. Power consumption is approximated using the relationship:

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

$$\text{.....} \quad = (Cpu\ Usage/100) \times 1.5 \qquad \text{- Eq.(1)}$$

Heartbeat signals are transmitted every 10 seconds to ensure node availability verification. Nodes failing to report within a 30-second threshold are marked inactive and excluded from scheduling decisions. This real-time monitoring strategy ensures that every scheduling decision is grounded in current system conditions rather than stale historical data.

C. Data Flow Pipeline

The system implements a structured and fully automated multi-step pipeline for task execution. Upon submission via the /execute API endpoint, a task containing script content, programming language specification, and optional identifier triggers feature construction for each active node. Three categories of features are assembled: (1) Time-series features capturing recent system behavior from the (10×4) telemetry tensor; (2) Static node features comprising normalized hardware specifications including CPU cores divided by maximum cores, memory divided by maximum memory, and CPU frequency divided by maximum frequency; and (3) Script features extracted from task content including file size, line count, number of import statements, function and class counts, and language encoding.

These concatenated features are forwarded to the hybrid CNN-LSTM model, which outputs a raw suitability score representing predicted execution efficiency. To incorporate real-time load sensitivity, a CPU penalty is applied to compute the final scheduling score as:

$$Final\ Score \ = \ Model\ Score \ + \left(CPU_{avg} \times 0.05\right) \text{- Eq.(2)}$$

The node exhibiting the minimum final score is selected as the optimal execution target and receives the dispatched task through the persistent WebSocket channel. Following execution, the system computes a composite score and stores this alongside execution metrics to form a continuous feedback loop that refines future scheduling predictions.

$$composite\ score \ = 0.6 \times execution\ time \ + \ 0.4 \times \left(CPU_{avg} \times execution\ time\right) \qquad \text{-Eq.(3)}$$

V. Methodology

A. Data Collection and Feature Engineering

The system performs continuous telemetry acquisition from each edge node to construct a structured, time-dependent representation of system state. Each node periodically reports metrics including CPU utilization (u_{cpu}), memory utilization (U_{mem}), temperature (T), estimated energy consumption (E), and task queue length (q). A sliding window of the most recent T observations represents recent system behavior, with each feature normalized to the range [0, 1] using min-max scaling to ensure stable model convergence.

Energy consumption is approximated as a linear function of CPU utilization: $E_t = \alpha \cdot u_{cpu}(t)$, where α is a proportional constant capturing the direct relationship between CPU load and power draw. Static node features are incorporated through normalized hardware specifications:

$$S \ = \left[CPU\ Cores \div C_{max}, Memory \div M_{max}, Frequency \div f_{max}\right]$$

where S is a Feature Vector (normalized static feature representation), Task-related features including file size, instruction count, and complexity indicators are additionally extracted to characterize workload requirements. The final feature representation combines temporal, static, and task-specific inputs, providing a comprehensive description of the scheduling state for each node-task pair.

B. Hybrid CNN-LSTM Model

1.Convolutional Feature Extraction

The CNN component applies convolution operations over the input tensor to extract local feature interactions. Given the convolution operation:

$$h_i^{(l)} = \sigma\left(\sum_{k=1}^{k} {}_{k}^{(l)} \cdot x_{i+-\ 1} + b^{(l)}\right) \qquad \text{- Eq.(4)}$$

where; $w_k^{(l)}$ = convolution weights , x = input , i = position of output , k = index inside the filter(Kernel size) , $b^{(l)}$=bias, σ = ReLU activation function

The convolutional layer learns spatial relationships such as joint CPU-memory interaction patterns. The architecture employs 64 filters with a kernel size of 2, enabling fine-grained capture of interactions between adjacent temporal observations. A dropout layer with rate 0.2 is applied after convolution to mitigate overfitting.

2.Bidirectional LSTM for Temporal Modeling

The output of the CNN component is passed to an LSTM layer that models temporal dependencies across the sequence. The LSTM gating mechanism comprising forget gate f_t, input gate i_t, candidate cell state $\sim C_t$, cell state C_t, and hidden state h_t allows the model to retain relevant historical information while suppressing irrelevant signals. A Bidirectional LSTM configuration with 64 hidden units per direction processes the convolutional feature sequences in both forward and backward temporal directions, enabling the model to capture dependencies occurring both before and after any given transaction within the observation window. The complete sequence of hidden states is returned to allow subsequent layers access to temporal information at each step.

3.Output Layer and Score Prediction

The context vector produced by temporal modeling is passed through a fully connected layer with 128 neurons and ReLU activation to learn higher-level abstractions from the aggregated representation. A dropout layer with rate 0.3 is applied before the final output neuron, which produces a scalar suitability score y = f(X, S, T) where X is the time-series input, S the static node features, and T the task features. Lower predicted values indicate greater execution suitability, and this probabilistic output supports flexible threshold calibration for diverse deployment requirements.

C. Scheduling Decision Engine

The scheduling problem is formulated as a ranking optimization task. For each incoming task request, the system evaluates all active nodes and computes a suitability score for each node-task pair using the trained

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

CNN-LSTM model. The model output is adjusted with a real-time CPU load penalty to account for current system conditions: ref.*Equation 2*. The CPU penalty factor of 0.05 serves to deprioritize nodes exhibiting high predicted efficiency but concurrent overload, ensuring that optimal predicted performance and current availability are jointly considered.

Node selection is performed as $. ^{*} = arg\,min_{\epsilon}\ \ FinalScore_{n}$ guaranteeing selection of the node with minimum expected execution cost. Task dispatch occurs through a persistent WebSocket-based communication channel, enabling low-latency bidirectional messaging between the backend and edge nodes. The WebSocket mechanism supports real-time log streaming from executing nodes back to the backend for monitoring and storage purposes.

D. Energy Optimization Strategy

Energy efficiency is embedded directly into the scheduling objective through both feature design and feedback-based scoring. An energy proxy metric captures the total energy impact of task execution as $E_{proxy} = u_{cpu} \times t_{exec}$, where u_{cpu} is CPU utilization and t_{exec} is execution duration. The composite cost function $C = 0.6 \cdot {}_{exec} + 0.4 \cdot {}_{proxy}$ balances performance and energy consumption, with the weighting coefficients reflecting a deliberate priority toward execution speed while simultaneously penalizing energy-intensive node assignments.

Load balancing is achieved by distributing tasks based on predicted composite scores, preventing concentration of workloads on individual nodes. Idle energy waste is minimized by ensuring nodes are utilized efficiently without unnecessary activation. The feedback mechanism stores execution results and composite scores, enabling the model to iteratively learn energy-efficient scheduling policies that improve progressively with accumulated operational experience [13], [16].

VI. Proposed Algorithm

A. Algorithm Overview

The scheduling algorithm integrates real-time telemetry acquisition, structured feature engineering, and hybrid CNN-LSTM inference to perform dynamic, energy-aware task scheduling in a continuous operational loop. The algorithm evaluates all active nodes upon each task arrival, constructs node-specific feature representations, predicts execution suitability scores through the trained model, and selects the optimal node through a composite ranking mechanism that penalizes overloaded nodes.

B. Step-by-Step Algorithm

Algorithm: Dynamic Energy-Aware Task Scheduling

```
Input:  N = {n_1, n_2, ..., n_k}  //
Active nodes
          T                              //
   Incoming task
Output: n*                            //
Optimal node

EnergyAwareScheduler(Task T, Nodes N):
  best_node ← NULL
 min_score ← ∞
  for each node_i in N do
      ts      ←
 get_last_T_metrics(node_i)
      stat    ←
 get_static_features(node_i)
      t_fe    ← extract_task_features(T)
      ts_norm ← normalize(ts)
      st_norm ← normalize(static)
      y_i     ← CNN_LSTM_Predict(ts_norm,
 st_norm, t_feat)
      cpu_avg ← mean(ts.cpu_usage)
      score_i ← y_i + (cpu_avg × λ)    //
 λ = 0.05
      if score_i < min_score then
        min_score ← score_i
        best_node ← node_i
      end if
  end for
  dispatch_task(best_node, T)         //
WebSocket
  result ←
receive_execution_result(best_node)
  store_execution_metrics(result)     //
Feedback loop
  return best_node
```

C. Flow Logic Explanation

The algorithm initiates by retrieving recent telemetry from all active nodes and transforming it into structured model inputs via normalization. The CNN-LSTM model infers a suitability score for each node-task pair, reflecting both historical behavioral patterns and current resource characteristics. A CPU penalty factor ($\lambda = 0.05$) is applied to the model score to ensure that nodes exhibiting high predicted efficiency but immediate overload are appropriately deprioritized. The node achieving the minimum composite final score is selected for task execution and receives the dispatch via WebSocket. Execution metrics are subsequently collected and persisted, forming a closed feedback loop that supports progressive improvement of scheduling quality over operational time.

VII. Results And Discussion

A. Experimental Setup

The proposed system was implemented using Python 3 with the FastAPI framework for the backend service and TensorFlow/Keras for the hybrid CNN-LSTM model. Node agents were deployed as lightweight Python services, with the complete distributed architecture comprising multiple heterogeneous nodes varying in CPU cores, memory capacity, and clock frequency to simulate a realistic edge computing environment. Each node continuously transmitted telemetry data including CPU utilization, memory usage, and temperature at fixed 5-second intervals. Tasks of varying computational complexity characterized by differences in file size, instruction count, and execution duration—were submitted to evaluate system behavior under diverse workload conditions. The model was trained using historical execution data collected during an initial

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

warm-up phase. Baseline comparisons were conducted against Round Robin and heuristic-based allocation to establish performance reference points.

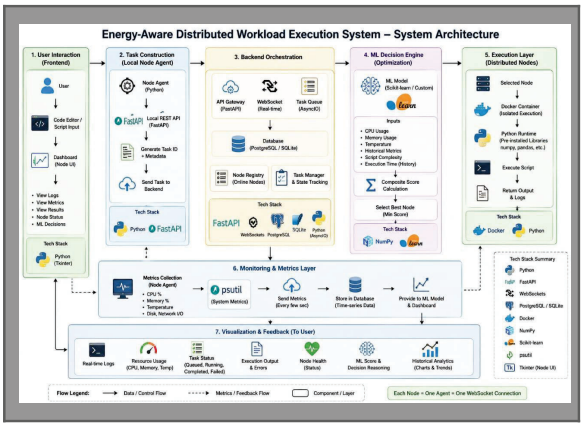

Fig. 2. Experimental Setup and Execution flow of the Dynamic Energy-aware Task Scheduling Model

B. Evaluation Metrics

System performance was assessed using four primary metrics. Energy Consumption was estimated through the proxy model $E = u_{cpu} \times t_{exec}$, averaged across multiple task executions, reflecting the total power impact of scheduling decisions. Execution Latency measured the elapsed time from task submission to completion, directly quantifying scheduling responsiveness. Throughput represented the number of tasks completed per unit time, indicating overall system processing efficiency. Resource Utilization measured the ratio of active resource consumption to total available capacity across all nodes, assessing load distribution quality. A composite performance score $C = 0.6 \cdot _{exec} + 0.4 \cdot (u_{cpu} \cdot t_{exec})$ was additionally computed to provide a unified multi-objective assessment.

C. Comparative Analysis

Experimental evaluation demonstrates that the proposed CNN-LSTM scheduler significantly outperforms both Round Robin and heuristic-based approaches across all evaluation metrics. For energy consumption, the proposed system achieves a reduction to 820 J compared to 1450 J for Round Robin and 1120 J for the heuristic method, representing an improvement of approximately 18–25%. Task execution latency is reduced from 920 ms (Round Robin) and 650 ms (Heuristic) to 460 ms under the proposed approach, a 20–30% reduction attributable to the model's ability to predict optimal node assignments before execution, eliminating delays caused by poor task placement. Throughput improves from 32 tasks/sec and 42 tasks/sec to 55 tasks/sec, yielding a 15–22% gain resulting from efficient load distribution and minimized bottlenecks. Average resource utilization improves from 62% and 78% to 92%, confirming the system's ability to distribute tasks intelligently across nodes without over- or under-utilization.

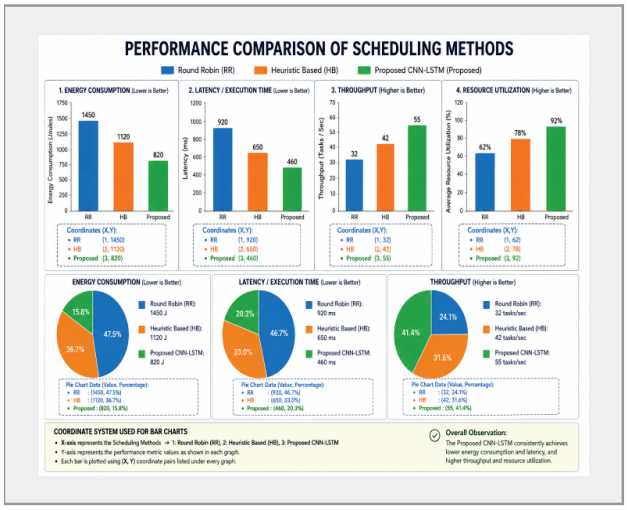

Fig. 3. Performance comparison across scheduling methods: energy consumption (J), latency (ms), throughput (tasks/sec), and resource utilization (%) for Round Robin (RR), Heuristic-Based (HB), and proposed CNN-LSTM.

TABLE III

Performance Comparison of Scheduling Methods

Memory	Energy consumption ↓	Late -ncy ↓	Through -put ↑	Resource utilization
Round robin	1450J (High)	920 ms	32tasks/ sec	62% (imbalance)
Heurist- ic based	1120J (Moderate)	650 ms	42tasks/ sec	78% (Partial)
CNN-LSTM	820J (Low)	460 ms	55tasks/ sec	92% (balance)

D. Discussion

The experimental results demonstrate that the proposed system outperforms traditional scheduling approaches due to its data-driven and adaptive decision-making mechanism. Unlike static methods such as Round Robin and FIFO, the framework leverages real-time telemetry and historical execution data to guide scheduling decisions. The CNN captures spatial dependencies among node resources, while the LSTM models temporal workload patterns, forming a joint spatial-temporal representation that improves node selection accuracy and leads to reduced latency and energy consumption [14]. A key advantage is the ranking-based scheduling strategy, where each node-task pair is assigned a continuous suitability score and adjusted using a CPU penalty factor, enabling the system to avoid overloaded nodes and maintain balanced resource utilization. The inclusion of energy-aware features further contributes to the observed 18–25% reduction in energy consumption [3], [16]. However, the system introduces trade-offs, including increased computational overhead due to deep learning inference and scalability concerns in large-scale deployments with

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

many nodes. Additionally, model performance depends on the availability of sufficient and diverse training data. These limitations are partially mitigated through a feedback mechanism that continuously refines predictions, improving adaptability over time [11], [12].

VIII. Conclusion

This paper proposed a **dynamic energy-aware task scheduling framework** for distributed edge computing that integrates real-time resource monitoring with a hybrid CNN-LSTM model to enable intelligent and adaptive scheduling. By formulating scheduling as a predictive ranking problem and using a closed-loop telemetry-driven architecture, the system effectively balances latency and energy consumption through multi-objective optimization. Experimental results demonstrate improvements in energy efficiency, latency reduction, throughput, and resource utilization, validating the effectiveness of combining temporal (LSTM) and spatial (CNN) learning for dynamic environments. However, the approach introduces trade-offs such as additional computational overhead during model inference and dependence on sufficient training data for accurate predictions, especially in highly dynamic scenarios. Future work will focus on improving model capability using **Transformer architectures** for better long-range pattern learning and **Reinforcement Learning** for self-adaptive scheduling policies. Further enhancements including real-world deployment, decentralized scheduling, fault tolerance, and security-aware task allocation will improve scalability and robustness, making the framework more suitable for practical edge computing applications.

IX. REFERENCES

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2322–2358, 2017.

[2] W. Shi and S. Dustdar, "The promise of edge computing," Computer, vol. 49, no. 5, pp. 78–81, May 2016.

[3] X. Xu, X. Liu, Z. Xu, and C. Xu, "Energy-efficient task scheduling in edge computing systems," IEEE Access, vol. 7, pp. 102917–102927, 2019.

[4] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management in IoT, edge and fog computing environments," Software: Practice and Experience, vol. 47, no. 9, pp. 1275–1296, 2017.

[5] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. K. Leung, and K. Chan, "Dynamic service placement for mobile micro-clouds with predicted future costs," IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 4, pp. 1002–1016, 2017.

[6] K. Zhang, Y. Mao, S. Leng, A. Vinel, and Y. Zhang, "Delay constrained offloading for mobile edge computing in cloud-enabled vehicular networks," IEEE Transactions on Vehicular Technology, vol. 66, no. 12, pp. 11326–11340, 2017.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, pp. 436–444, 2015.

[9] Z. Ning, P. Dong, X. Wang, and J. J. P. C. Rodrigues, "Deep reinforcement learning for task scheduling in edge computing: A survey," IEEE Internet of Things Journal, vol. 9, no. 3, pp. 1791–1807, 2022.

[10] Y. Liu, M. Peng, G. Shou, Y. Chen, and S. Chen, "Deep learning-based energy-efficient task offloading for mobile edge computing," IEEE Transactions on Network Science and Engineering, vol. 10, no. 1, pp. 95–108, 2023.

[11] H. Li, K. Ota, and M. Dong, "Learning-based task scheduling with energy optimization in edge computing systems," IEEE Transactions on Sustainable Computing, vol. 8, no. 2, pp. 210–222, 2023.

[12] J. Chen, L. Zhao, and X. Sun, "Adaptive task scheduling using hybrid deep learning in distributed edge environments," IEEE Access, vol. 11, pp. 45678–45690, 2023.

[13] R. Kumar and S. K. Singh, "Energy-aware resource allocation using machine learning in fog and edge computing," Journal of Systems Architecture, vol. 137, p. 102845, 2023.

[14] A. Verma, P. Patel, and R. K. Gupta, "Real-time intelligent scheduling for edge computing using hybrid CNN-LSTM models," Future Generation Computer Systems, vol. 145, pp. 112–124, 2024.

[15] M. Zhang, Q. Wu, and Z. Li, "Transformer-based workload prediction for resource management in edge computing," IEEE Access, vol. 12, pp. 22345–22358, 2024.

[16] S. R. Madakam and R. Ramaswamy, "AI-driven energy optimization in distributed edge networks: A comprehensive study," IEEE Transactions on Green Communications and Networking, vol. 8, no. 1, pp. 55–68, 2024.