ISSN: 2395-1680 (ONLINE)
DOI: 10.21917/ijme.2025.0326

REAL-TIME PERFORMANCE OPTIMIZATION OF ELECTRONIC EMBEDDED SYSTEMS USING DEEP REINFORCEMENT LEARNING ALGORITHMS

N. Jagadeeswari[1], R. Sudha[2] and M. Bhavani[3]

[1]*Department of Computer Science and Engineering, Thanthai Periyar Government Institute of Technology, India*
[2]*Department of Electrical and Electronics Engineering, Thanthai Periyar Government Institute of Technology, India*
[3]*Department of Electrical and Electronics Engineering, Government College of Engineering, Srirangam, India*

Abstract

The rapid evolution of electronic embedded systems (EES) has brought significant challenges in optimizing their performance in real-time environments. These systems are often deployed in critical applications, such as automotive, medical, and IoT devices, where efficient resource management and adaptive decision-making are essential for optimal performance. Traditional optimization methods struggle to meet the dynamic and complex demands of modern embedded systems. As the complexity of electronic embedded systems increases, ensuring real-time performance while minimizing energy consumption, latency, and operational costs becomes more difficult. Static configurations or conventional algorithms cannot adapt quickly to changing conditions, leading to suboptimal performance. This problem is further exacerbated by the need for fast decision-making within limited computational resources. This study proposes using Deep Reinforcement Learning (DRL) algorithms to optimize the real-time performance of electronic embedded systems. DRL leverages an agent-based approach to autonomously learn optimal strategies through trial and error in dynamic environments. The proposed method involves training a DRL model to intelligently manage system resources, adjust parameters, and enhance decision-making in real-time based on feedback from the system's environment. Key DRL techniques, such as Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO), are utilized to train agents in various system scenarios. The results show that DRL-based optimization significantly improves system efficiency, leading to reduced latency, enhanced throughput, and optimized power consumption without compromising the system's responsiveness. The proposed method outperforms traditional optimization approaches, particularly in highly dynamic and resource-constrained environments, by enabling continuous adaptation to changing operational conditions.

Keywords:

Deep Reinforcement Learning, Electronic Embedded Systems, Real-Time Optimization, Power Consumption, System Performance

1. INTRODUCTION

The advent of electronic embedded systems (EES) has been transformative across various industries such as automotive, healthcare, industrial automation, and the Internet of Things (IoT). These systems are typically designed to perform specific tasks with limited computational resources, which requires the optimization of key performance factors such as processing speed, power consumption, and memory utilization. Traditional optimization techniques often fall short in addressing the real-time demands of modern embedded systems, especially as they become more complex and integrated with sophisticated sensors and communication devices. Real-time performance optimization in EES is critical for ensuring reliability, efficiency, and responsiveness in applications like autonomous vehicles, medical devices, and smart cities, where performance needs to be consistently monitored and adjusted based on environmental conditions [1-3].

Despite the advances in embedded systems, several challenges persist. One of the key challenges is the need for adaptive decision-making mechanisms that can operate efficiently in real-time while handling large amounts of data generated by sensors or external inputs. Conventional optimization techniques, such as heuristics or pre-configured algorithms, are typically static and incapable of adjusting to changing system parameters, leading to performance degradation over time. Additionally, real-time constraints make it difficult to utilize computationally expensive models or methods. Another major challenge is managing the balance between performance and energy consumption, as modern systems demand more power and longer battery life, which can conflict with the need for higher processing power. These challenges have heightened the necessity for adaptive and efficient optimization strategies that can make real-time decisions without compromising system performance [4-6].

The primary problem in optimizing EES in real-time lies in the system's inability to autonomously adjust to dynamic environmental and operational conditions. Existing techniques are often limited by their inability to learn and adapt quickly to real-time changes, which leads to inefficient use of system resources such as processing power, memory, and energy. Furthermore, static optimization models are poorly suited for systems where workload and inputs vary over time, often resulting in delays or energy inefficiencies. This research focuses on addressing these issues by using Deep Reinforcement Learning (DRL), which enables the system to autonomously learn optimal decision-making policies in real-time environments through interactions with its surroundings. DRL's ability to perform continuous learning and adapt to new conditions offers a promising solution for enhancing the real-time performance of EES [7-10].

The primary objective of this research is to explore the effectiveness of DRL algorithms for real-time performance optimization in electronic embedded systems. Specifically, the goals are:

- To develop a DRL-based framework for dynamically optimizing the performance of embedded systems, including energy consumption, processing speed, and memory utilization.
- To evaluate the effectiveness of DRL algorithms in comparison to conventional optimization methods in real-time environments.

This study introduces a novel approach by applying DRL algorithms to real-time performance optimization in embedded systems. Unlike traditional methods, DRL provides a framework where the system learns and evolves over time, improving

decision-making efficiency and ensuring optimal resource utilization in dynamically changing environments. The main contributions of this research include:

- Proposing a DRL-based solution that integrates learning and optimization to adapt system performance in real-time based on environmental feedback.
- Demonstrating the potential of DRL to address the challenges of power consumption and latency without sacrificing system responsiveness.
- Providing a comparative analysis of DRL-based optimization versus conventional methods, showcasing its superiority in real-time embedded system scenarios.

2. RELATED WORKS

Real-time performance optimization in electronic embedded systems has garnered significant attention in recent years, particularly with the growing demand for smarter, more efficient devices. Numerous approaches have been proposed to optimize key factors like processing speed, power consumption, and memory utilization in EES. Traditional methods, such as heuristic-based optimization, have been employed in various embedded systems but often fail to adapt in real-time scenarios where system parameters fluctuate rapidly. These methods are generally designed for static environments, limiting their effectiveness in dynamic real-time applications.

Machine learning-based approaches have recently gained traction for their ability to adapt to changing conditions. Several studies have explored the use of supervised and unsupervised learning algorithms to enhance system performance. For instance, decision trees and support vector machines (SVM) have been applied to resource allocation problems in embedded systems, achieving some success in optimizing system parameters. However, these approaches still require manual feature engineering and lack the ability to continuously adapt based on feedback from the system environment.

Reinforcement Learning (RL), a subset of machine learning, has proven to be a powerful tool for optimizing decision-making in dynamic environments. RL's ability to learn optimal policies through trial and error makes it particularly suitable for real-time performance optimization. Several studies have applied RL to embedded systems, especially in the context of resource management and power optimization. For Sample, Kim et al. (2019) proposed an RL-based approach for energy-efficient scheduling in embedded systems, demonstrating significant improvements in energy savings and processing time.

Deep Reinforcement Learning (DRL), which combines deep learning with RL, has been increasingly explored for its potential to handle high-dimensional state spaces and complex decision-making processes. DRL techniques such as Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO) have been used for various optimization tasks, including power management, workload scheduling, and resource allocation in embedded systems. Wang et al. (2020) introduced a DRL-based method for optimizing the power consumption of mobile devices, showing that DRL outperforms traditional optimization techniques by continuously adapting to the system's usage patterns.

Another area of interest is the application of DRL in IoT-based embedded systems. Zhang et al. (2021) shown the use of DRL to optimize energy consumption and performance in IoT networks, highlighting the potential of this approach in distributed systems where multiple devices need to cooperate for optimal performance. Similarly, Yu et al. (2020) applied DRL to optimize the communication protocols of embedded IoT devices, achieving better data throughput and lower latency compared to conventional methods.

Despite the promising results, there are challenges in applying DRL to embedded systems. The need for high computational resources during training, limited availability of large datasets, and the difficulty in transferring models to embedded hardware are some of the barriers that remain. Additionally, there is a lack of consensus on the best DRL algorithm for specific embedded system applications, with ongoing research aimed at improving the efficiency and robustness of DRL models.

Recent advancements in DRL optimization, such as using multi-agent systems for collaborative decision-making in embedded devices, have shown further promise. Multi-agent DRL can be used in scenarios where multiple devices need to optimize their performance simultaneously, such as in smart grid systems or collaborative autonomous vehicles. These systems benefit from distributed learning, where each agent can learn from its environment while sharing information with other agents to improve global system performance.

Thus, the application of DRL to real-time performance optimization in embedded systems holds great potential, particularly as embedded systems become more complex and ubiquitous. Future work will focus on addressing the limitations of current DRL methods, such as reducing training time and improving the transferability of learned models to hardware platforms.

3. PROPOSED METHOD

The proposed method utilizes Deep Reinforcement Learning (DRL) algorithms to optimize the real-time performance of electronic embedded systems. The method follows a process in which an agent learns and adapts its actions to optimize key performance metrics such as energy consumption, processing speed, and memory utilization in dynamic environments. The DRL agent interacts with the embedded system environment, receiving feedback based on the system's current state. The process is structured as follows:

- **Environment Modeling**: The embedded system's operational environment is modeled, defining key variables such as processing workload, available resources (e.g., CPU, memory, energy), and system constraints (e.g., real-time latency).
- **State Representation**: The current state of the system is represented as a set of features, such as resource usage and performance metrics, which are continuously monitored and fed to the agent.
- **Action Space Definition**: The agent is equipped with a set of actions it can take to influence the system's performance, such as adjusting processing frequencies, managing task prioritization, or optimizing resource allocation.

- **Reward Function**: A reward function is designed to quantify the system's performance based on the actions taken. The reward encourages the agent to minimize energy consumption, reduce latency, and maximize throughput while ensuring system stability.
- **Policy Learning**: Using a DRL algorithm (such as Deep Q-Network or Proximal Policy Optimization), the agent explores various actions to maximize cumulative rewards over time. The learning process is iterative, with the agent continuously updating its decision-making policy based on feedback from the environment.
- **Real-time Adaptation**: Once trained, the agent is deployed in real-time applications, where it dynamically adjusts the system's configuration based on ongoing performance data. This enables continuous optimization of the system, ensuring that it remains efficient and responsive under varying workloads and environmental conditions.

This DRL-based method enables embedded systems to autonomously adjust their performance parameters in real-time, significantly improving efficiency and adaptability in complex, resource-constrained environments.

3.1 ENVIRONMENT MODELING

The "Environment Modeling" step is critical to the proposed method, as it forms the foundation for the Deep Reinforcement Learning (DRL) agent to interact with and optimize the embedded system's real-time performance. In this phase, we define the system's operational environment by outlining key components, constraints, and variables that the agent must monitor and control. The environment represents a dynamic setting where the agent observes system states, makes decisions, and receives feedback based on its actions.

The embedded system's environment is modeled as a set of states, actions, and rewards, where each component influences the performance of the system. A system can be thought of as consisting of several interacting parts, such as processors, memory units, power management units, and I/O components. Each of these elements has associated resource constraints, and the agent needs to monitor them to make decisions that optimize the overall performance of the system. For Sample, the agent could adjust CPU frequencies, modify memory allocation, or optimize energy consumption based on the current workload and the available system resources.

State variables represent the current conditions of the embedded system. These variables are continuously monitored and include parameters such as CPU usage, memory utilization, task execution times, and battery level. The state is a snapshot of the system's configuration at a given time and serves as the input for the agent's decision-making process.

Table.1. State Variable

State Variable	Description	Value
CPU Usage	Percentage of CPU utilization by active processes	75%
Memory Utilization	Percentage of memory currently in use	60%
Battery Level	Remaining energy or power of the system	50%
Task Execution Time	Time taken to execute a task or process	200ms

The action space defines the set of possible actions the agent can take to influence the system's performance. These actions are designed to manipulate the system's resources and adjust its parameters to achieve the desired performance goals. In our embedded system context, actions could involve adjusting processor frequency, task scheduling, or optimizing power management strategies. For instance, an action could be to increase the CPU frequency to speed up computations or to lower it to conserve energy.

Table.2. Action Space

Action	Description	Value
Adjust CPU Frequency	Change CPU frequency to meet processing demand	Increase to 2.5 GHz
Reallocate Memory	Move data between memory segments to optimize usage	Reallocate 1 GB
Adjust Power Settings	Modify system's power consumption modes (e.g., sleep mode)	Enter low-power mode

The reward function is designed to guide the agent's learning process by providing feedback on the effectiveness of its actions. The reward is a scalar value based on the system's performance after taking an action. For instance, if an action leads to improved system performance, such as reducing latency or energy consumption while maintaining throughput, the agent receives a positive reward. Conversely, if the action results in system inefficiencies or violations of real-time constraints, the agent receives a negative reward.

Table.3. Reward Function

Action	Outcome	Reward
Increase CPU Frequency	Reduced task execution time, higher energy consumption	+10 for performance, -5 for energy usage
Decrease CPU Frequency	Increased task execution time, lower energy consumption	+3 for energy savings, -8 for latency

The environment's dynamics are driven by the interactions between the system's state, the agent's actions, and the corresponding rewards. Each time the agent takes an action, the system's state transitions, which can affect other components of the system, such as memory or power. For Sample, increasing CPU frequency may improve task execution time but may also lead to higher power consumption and faster battery depletion. Similarly, optimizing memory usage can enhance performance by reducing task delays but could lead to increased latency if resources are over-allocated.

The agent observes the resulting changes and adjusts its policy over time to optimize overall system performance. Through repeated interactions with the environment, the agent learns how

to balance trade-offs, such as minimizing power consumption while maintaining system responsiveness.

3.2 STATE REPRESENTATION

State representation plays a crucial role in the proposed method, as it defines how the current conditions of the embedded system are captured and presented to the Deep Reinforcement Learning (DRL) agent. In this phase, the system's state is modeled as a set of features that provide relevant information about the system's performance and resource utilization at any given point in time. The agent uses these features to understand the environment and make decisions that optimize the system's real-time performance.

The state is essentially a snapshot of the system, capturing information about various system parameters such as CPU usage, memory utilization, task execution times, energy consumption, and system latency. By representing the state in a structured manner, the agent can process the data effectively, ensuring that its decisions are based on an accurate and up-to-date view of the system's performance.

The state representation is defined as a vector of features, each corresponding to a key parameter of the embedded system. These features are continuously updated, allowing the agent to observe and adapt to dynamic changes in the system's behavior and workload.

3.2.1 State Variables:

The key state variables include metrics related to resource utilization, task performance, and energy consumption. These state variables provide the DRL agent with the necessary input to evaluate how well the system is performing and determine the best course of action.

Table.4. State Variables

State Variable	Description	Value
CPU Usage	The percentage of CPU resources currently being used	75%
Memory Utilization	The percentage of available memory being used by processes	60%
Task Execution Time	The time taken to complete a task or process	200ms
Energy Consumption	The amount of energy used by the system during operation	250mW
System Latency	The delay in processing or communication within the system	50ms

These state variables represent the current operating conditions of the embedded system, with each one playing a pivotal role in determining system performance. For Sample, CPU Usage and Memory Utilization reflect how much computational and memory resources are being consumed, which directly impacts the system's ability to process tasks efficiently. Energy Consumption is another critical metric, as the system's power usage must be optimized to extend battery life, especially in resource-constrained environments.

3.2.2 State Representation Vector:

The state at any given time is represented as a vector of these variables, which serves as the input to the DRL agent. The vector provides the agent with a consolidated view of the system's current status. By observing this vector, the agent can infer how the system is performing in terms of resource usage, processing efficiency, and energy management.

Table.5. State representation vector for the embedded system

State Feature	Value
CPU Usage	75%
Memory Utilization	60%
Task Execution Time	200ms
Energy Consumption	250mW
System Latency	50ms

This vector provides the agent with a holistic view of the system's performance and resource utilization. The agent uses this vector to evaluate the current conditions and determine which action to take based on the learned policy.

3.2.3 Dynamic State Update:

The state representation is continuously updated as the system changes. For Sample, when a task starts, the Task Execution Time increases. If the system consumes more energy to complete a task, the Energy Consumption value will rise. Similarly, as tasks are processed, Memory Utilization and CPU Usage will fluctuate, reflecting the dynamic nature of the system's operations.

To ensure that the agent makes decisions based on the most recent information, the state variables are updated at regular intervals. The DRL agent, therefore, observes this dynamic state and adjusts its actions accordingly to optimize system performance.

Table.6. Dynamic State Update

State Variable	Initial Value	Updated Value
CPU Usage	75%	80%
Memory Utilization	60%	65%
Task Execution Time	200ms	250ms
Energy Consumption	250mW	300mW
System Latency	50ms	55ms

In this Sample, the CPU Usage and Memory Utilization have increased, possibly due to a higher workload, while Energy Consumption has also risen. The Task Execution Time has increased as the task took longer to complete, and System Latency has slightly increased as a result of these changes.

3.3 ROLE IN AGENT DECISION-MAKING

By continuously observing these state variables, the DRL agent can learn the relationship between system performance and actions. For instance, if the agent detects high Energy Consumption and Task Execution Time, it might decide to reduce the CPU frequency to lower energy usage and improve processing speed. On the other hand, if Memory Utilization is low and the

task requires higher processing power, the agent may choose to increase the CPU frequency to complete the task faster.

The dynamic nature of the state representation allows the agent to make real-time adjustments to optimize system performance based on the current operational conditions. The agent adapts over time, learning which state features are most indicative of the system's needs and how best to adjust actions to meet the desired objectives.

3.3.1 Action Space Definition:

The Action Space is a fundamental aspect of the Deep Reinforcement Learning (DRL) model used to optimize the real-time performance of embedded systems. It defines the set of all possible actions that the DRL agent can take in response to the state of the system, ultimately influencing how the system's resources (such as CPU, memory, and energy) are allocated. The DRL agent learns to choose actions from this space to maximize a predefined reward function, which leads to improved system performance under dynamic conditions.

In the context of embedded systems, the actions are typically designed to adjust various system parameters such as CPU frequency, memory allocation, task scheduling, and power management. The agent takes an action based on the current state of the system, which can affect performance metrics like processing speed, power consumption, latency, and task completion time.

3.3.2 Types of Actions:

Actions in the proposed method are defined to manipulate the following system components:

- **CPU Frequency Adjustment**: This action involves altering the operating frequency of the CPU to match processing demands. Increasing the frequency can reduce task execution time but may increase energy consumption, while lowering the frequency helps save power but might result in slower execution.
- **Memory Reallocation**: The agent can dynamically allocate or deallocate memory to different processes or tasks to optimize memory usage, reduce delays, and improve task completion times. The action could involve shifting processes between memory regions based on availability.
- **Power Management Adjustments**: Actions in this category involve managing the system's power consumption, such as entering low-power or sleep modes to conserve energy when the system is idle or underutilized.

Each action within this space is designed to address a specific aspect of the embedded system, contributing to overall optimization by balancing the trade-offs between performance and resource consumption.

3.3.3 Action Space Representation:

The action space is represented as a vector that encompasses all possible actions the agent can take at any given time. The vector can be partitioned into discrete and continuous actions depending on the nature of the system's resources. For Sample, CPU frequency might have a discrete set of values (e.g., low, medium, high), while memory allocation could be represented as a continuous range of values (e.g., between 0 and 100% of available memory).

Table.7. Action space representation

Action Type	Description	Values
CPU Frequency	Adjust CPU operating frequency to optimize processing speed	{1.2 GHz, 1.5 GHz, 2.0 GHz}
Memory Allocation	Reallocate system memory to different tasks or processes	{10%, 50%, 80%}
Power Mode Adjustment	Adjust the power consumption mode of the system	{Idle, Active, Low Power}

In this table, CPU Frequency is represented by a discrete set of frequencies, Memory Allocation is represented by the percentage of memory assigned to processes, and Power Mode Adjustment represents different power states the system can be in.

3.3.4 Mathematical Representation of Action Space:

To formalize the action space, we define the action space as a vector **A** that includes actions related to the system's resources. Let the action space be represented as:

$$A = [A_{\text{CPU}}, A_{\text{Memory}}, A_{\text{Power}}] \tag{1}$$

These actions are selected based on the current system state, allowing the agent to choose the optimal set of actions at each time step to improve system performance.

Furthermore, actions are typically represented in terms of their impact on system performance. The effect of these actions on system performance can be expressed by an objective function, which is influenced by the CPU frequency f_{CPU}, memory allocation m_{Memory} and power state p_{Power}. The total action impact I_{action} can be modeled as:

$$I_{\text{action}} = \alpha_1 f_{\text{CPU}} + \alpha_2 m_{\text{Memory}} + \alpha_3 p_{\text{Power}} \tag{2}$$

The values of α_1, α_2, α_3 are determined based on the system's priorities, such as whether the objective is to reduce energy consumption, improve processing speed, or balance both. This equation gives an overall impact score that reflects how the chosen actions influence system performance.

3.4 EXPLORATION VS. EXPLOITATION IN ACTION SELECTION

In reinforcement learning, the agent faces the exploration-exploitation trade-off. During the exploration phase, the agent tries various actions to understand their effects on the system, while during the exploitation phase, the agent uses its learned policy to select the actions that maximize the reward based on past experiences.

For the proposed method, the agent may initially explore different combinations of CPU Frequency, Memory Allocation, and Power Mode Adjustment to understand their individual and combined impacts on the system. Over time, the agent refines its policy by exploiting actions that have shown to result in optimal system performance, based on historical feedback.

3.4.1 Reward Function:

In the proposed method for real-time performance optimization of embedded systems, the Reward Function plays a crucial role in guiding the Deep Reinforcement Learning (DRL) agent to make decisions that enhance the system's performance while considering resource constraints like energy consumption,

CPU utilization, and task completion time. The reward function quantifies how beneficial or detrimental a particular action is based on the agent's state and chosen action, helping the agent learn optimal behaviors over time.

3.4.2 Reward Components:

The reward function incorporates multiple factors that reflect the trade-offs between system performance and resource consumption. Key components of the reward function include:

- **Performance Improvement**: This component rewards the agent for actions that enhance system performance, such as reducing task execution time, increasing throughput, or improving responsiveness.
- **Energy Efficiency**: This component rewards the agent for actions that minimize power consumption, such as transitioning the system to low-power states when the demand is low or adjusting the CPU frequency to avoid unnecessary power usage.
- **Resource Utilization**: Efficient use of resources, such as CPU and memory, is another factor in the reward function. The agent is rewarded for actions that optimize resource allocation, avoiding both underutilization (wasting resources) and overutilization (leading to bottlenecks or system instability).
- **Latency Reduction**: In real-time embedded systems, reducing latency is often critical. The agent receives rewards for minimizing delays in task processing, which is crucial for time-sensitive applications.
- The total reward at any given time step t is a weighted sum of these components:

$$\begin{aligned} R(t) = {} & \lambda_1 \times \text{Performance Improvement}(t) \\ & - \lambda_2 \times \text{Energy Consumption}(t) \\ & + \lambda_3 \times \text{Resource Utilization}(t) \\ & - \lambda_4 \times \text{Latency}(t) \end{aligned} \quad (3)$$

Table.8. Reward Function Value

Factor	Description	Values
Performance Improvement	Measures the increase in system performance (e.g., reduced processing time)	{High, Medium, Low}
Energy Consumption	Measures the power consumed by the system	{High, Medium, Low}
Resource Utilization	Measures how effectively CPU and memory are utilized	{High, Medium, Low}
Latency	Measures task completion delay or latency	{High, Medium, Low}

In the table, higher values for Performance Improvement and Resource Utilization would be rewarded, while higher Energy Consumption and Latency would be penalized.

3.4.3 Impact of the Reward Function:

The reward function provides the feedback necessary for the agent to adjust its actions during training. Over time, the agent learns which actions maximize the reward by balancing the competing objectives, such as improving performance while minimizing energy consumption. The goal is to find a policy that achieves optimal system performance while adhering to the system's constraints, such as power usage and resource limitations.

3.4.4 Policy Learning:

Policy Learning is the process by which the Deep Reinforcement Learning (DRL) agent learns to select actions based on its current state to maximize the long-term cumulative reward. In the context of embedded systems optimization, this involves learning a policy that efficiently allocates resources (CPU, memory, power) based on the system's real-time state to optimize performance metrics like processing speed, energy consumption, and task completion time.

The proposed method uses Q-learning or Policy Gradient methods for policy learning. In these methods, the agent iteratively improves its decision-making process by evaluating the effectiveness of actions through the rewards received. The agent's goal is to maximize the expected cumulative reward over time, which is achieved by adjusting its policy.

3.4.5 Steps in Policy Learning

- **State-Action Evaluation**: At each time step, the agent observes the current state of the system, selects an action, and then observes the resulting state and reward. This observation-feedback loop enables the agent to evaluate the effectiveness of its actions.
- **Policy Update**: Based on the rewards received, the agent updates its policy. This can be done through a value-based approach (Q-learning) or a policy-based approach (Policy Gradient). In Q-learning, the value of a state-action pair is updated using the Bellman equation:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right) \quad (4)$$

In Policy Gradient methods, the policy is directly optimized by adjusting the parameters of the policy network to maximize the expected reward.

- **Exploration vs. Exploitation**: During training, the agent balances exploration (trying new actions) and exploitation (choosing the best-known action based on its current policy). Initially, the agent explores many actions to gather experience, and over time, it exploits the learned policy to maximize the reward.

Table.9. Policy Learning

Step	Description	Values
State Observation	Agent observes the current state of the system (e.g., CPU usage, memory allocation, power state)	{CPU = 80%, Memory = 50%}
Action Selection	Agent selects an action based on the policy (e.g., increase CPU frequency, allocate more memory)	{Increase CPU Frequency}
Reward Received	Agent receives a reward based on the impact of the action on system performance	{Reward = 50}

Policy Update	Agent updates its policy based on the reward and learned experiences	{New policy parameters}

In the table, the agent starts by observing the current system state and then chooses an action according to its policy. The reward received from that action is used to update the policy for future decision-making.

3.4.6 Impact of Policy Learning:

Over time, the DRL agent refines its policy to make better decisions, improving the system's performance while considering resource constraints. The policy is learned through continuous interaction with the system, with the agent receiving feedback in the form of rewards. Eventually, the agent can make real-time decisions that optimize performance based on current system conditions, effectively adapting to dynamic changes in workload and resource availability. By continuously adjusting its policy using the rewards it receives, the agent learns to make efficient trade-offs between system performance and resource consumption, leading to a well-optimized embedded system.

4. RESULTS AND DISCUSSION

In the proposed method for real-time performance optimization of embedded systems using Deep Reinforcement Learning (DRL), a comprehensive experimental setup is designed to evaluate the effectiveness of the approach. For simulation purposes, Python and TensorFlow are used to implement the DRL algorithms. The DRL model is trained on a simulated environment that mirrors real-time embedded systems with dynamic workloads, resource constraints, and task scheduling scenarios.

For comparison, the proposed DRL-based optimization approach is benchmarked against three existing methods:

- **Energy-Aware Task Scheduling (EATS)**: This method focuses on optimizing energy consumption by scheduling tasks based on their energy demands.
- **Dynamic Voltage and Frequency Scaling (DVFS)**: This approach adjusts the system's voltage and frequency dynamically to balance energy consumption and system performance.
- **Reinforcement Learning-based Optimization (RL-Optimization)**: A traditional RL-based approach that does not leverage deep reinforcement learning and operates with simpler policies and models.

Table.10. Parameters

Parameter	Value
Simulation Tool	Python, TensorFlow, OpenAI Gym
Training Algorithm	Deep Q-Learning (DQN)
System Setup	Intel i7 Processor, 16GB RAM, NVIDIA GTX 1080 Ti GPU
Training Episodes	5000 Episodes
Batch Size	64
Learning Rate (α)	0.001
Discount Factor (γ)	0.99
Exploration Factor (ε)	0.1
Action Space	CPU Frequency Adjustment, Memory Allocation, Task Scheduling
Reward Function Weights (λ1-λ4)	λ1=1, λ2=0.5, λ3=0.3, λ4=0.2
Evaluation Period	1000 Simulation Steps per Method

4.1 PERFORMANCE METRICS

- *Performance Improvement*: This metric assesses how effectively the system can process tasks in less time, contributing to better overall system performance. An improvement in performance generally indicates that the system is better able to handle workloads and reduce task processing delays.
- *Energy Efficiency*: Embedded systems often operate under strict energy constraints. This metric quantifies how much power is saved by employing techniques like DVFS or other resource optimization strategies, balancing between performance and energy efficiency.
- *Resource Utilization*: Effective resource utilization ensures that no resources are wasted and that the system operates within its optimal capacity. This metric reflects the degree to which resources are used efficiently to achieve task completion.
- *Latency*: In real-time embedded systems, minimizing latency is critical. This metric evaluates the responsiveness of the system and how quickly it can process and complete tasks, which is especially important in time-sensitive applications.
- *Stability*: Stability is important for embedded systems to ensure continuous, reliable performance, particularly in systems that require 24/7 uptime. This metric evaluates whether the optimization approach maintains system stability across varied workloads and resource demands.

Table.11. Performance Metrics

Metric	EATS	DVFS	RL-Opt	Proposed DRL
Throughput (tasks/sec)	25.4	30.1	32.3	38.5
Processing Speed (sec/task)	1.25	1.05	0.98	0.82
Energy Efficiency (J/task)	2.8	2.4	2.2	1.7
Resource Utilization (%)	75	80	85	90
Latency (ms/task)	120	110	100	85
Stability (failure rate)	5%	4%	3%	1%

Across the 5000 episodes, the Proposed DRL shows consistent improvement in comparison to the existing methods. The Throughput increased to 38.5 tasks per second, surpassing existing methods by a significant margin. Processing Speed improved to 0.82 seconds per task, reducing task completion time compared to other methods. Energy Efficiency was optimized to 1.7 joules per task, showing a 23% improvement over the best existing method. The Resource Utilization reached 90%, reflecting better resource allocation. Latency reduced to 85 ms, and the Stability increased with only 1% failure rate, indicating high system reliability.

Table.12. Performance Metrics for λ1=1, λ2=0.5, λ3=0.3, λ4=0.2

Metric	EATS	DVFS	RL-Opt	Proposed DRL
Throughput (tasks/sec)	24.6	29.7	31.2	38.9
Processing Speed (sec/task)	1.28	1.10	1.00	0.78
Energy Efficiency (J/task)	3.0	2.6	2.3	1.6
Resource Utilization (%)	73	78	84	91
Latency (ms/task)	122	111	98	82
Stability (failure rate)	6%	5%	4%	2%

With the selected values of λ1=1, λ2=0.5, λ3=0.3, and λ4=0.2, the Proposed DRL shows superior performance across all metrics. Throughput increased to 38.9 tasks per second, outperforming the existing methods by up to 24%. Processing Speed improved to 0.78 seconds per task, reducing task completion time. Energy Efficiency was optimized to 1.6 joules per task, showing a 30% improvement over the best existing method. Resource Utilization reached 91%, highlighting better resource management. Latency dropped to 82 ms, and Stability was enhanced with only 2% failure rate, demonstrating reliability.

5. CONCLUSION

The proposed Deep Reinforcement Learning (DRL) method for real-time performance optimization of embedded systems has shown significant improvements over traditional optimization approaches. By leveraging dynamic learning and adaptive decision-making, the DRL model effectively balances the competing objectives of performance, energy efficiency, and resource utilization, providing substantial gains in system throughput, processing speed, and energy consumption. The experimental results show that the DRL approach outperforms existing methods such as Energy-Aware Task Scheduling (EATS), Dynamic Voltage and Frequency Scaling (DVFS), and RL-Optimization in several critical areas.

Notably, the proposed method achieved an optimized energy consumption of 1.6 joules per task, a reduction in processing time to 0.78 seconds per task, and a substantial increase in resource utilization, reaching 91%. Additionally, the system exhibited improved latency (82 ms) and enhanced stability with only a 2% failure rate. These results underscore the potential of DRL in tackling the complexities of real-time embedded system optimization.

REFERENCES

[1] R. Rotaeche, A. Ballesteros and J. Proenza, "Speeding Task Allocation Search for Reconfigurations in Adaptive Distributed Embedded Systems using Deep Reinforcement Learning", *Sensors*, Vol. 23, No. 1, pp. 1-6, 2023.

[2] J. Aldahmashi and X. Ma, "Real-Time Energy Management in Smart Homes through Deep Reinforcement Learning", *IEEE Access*, Vol. 12, pp. 55-72, 2024.

[3] B. Haouari, R. Mzid and O. Mosbahi, "A Reinforcement Learning-based Approach for Online Optimal Control of Self-Adaptive Real-Time Systems", *Neural Computing and Applications*, Vol. 35, No. 27, pp. 20375-20401, 2023.

[4] Z. Li, A. Samanta, Y. Li, A. Soltoggio, H. Kim and C. Liu, "R^3 :Device Real-Time Deep Reinforcement Learning for Autonomous Robotics", *IEEE Real-Time Systems Symposium*, pp. 131-144, 2023.

[5] G. Ai, X. Zuo, G. Chen and B. Wu, "Deep Reinforcement Learning based Dynamic Optimization of Bus Timetable", *Applied Soft Computing*, Vol. 131, pp. 1-7, 2022.

[6] X. Tang, B. Long and L. Zhou, "Real-Time Monitoring and Analysis of Track and Field Athletes based on Edge Computing and Deep Reinforcement Learning Algorithm", *Alexandria Engineering Journal*, Vol. 114, pp. 136-146, 2025.

[7] D. Hu and Y. Zhang, "Deep Reinforcement Learning based on Driver Experience Embedding for Energy Management Strategies in Hybrid Electric Vehicles", *Energy Technology*, Vol. 10, No. 6, pp. 1-7, 2022.

[8] N. Mazaheri, D. Santamargarita, E. Bueno, D. Pizarro and S. Cobreces, "A Deep Reinforcement Learning Approach to DC-DC Power Electronic Converter Control with Practical Considerations", *Energies*, Vol. 17, No. 14, pp. 1-6, 2024.

[9] A.R. Sayed, X. Zhang, G. Wang, J. Qiu and C. Wang, "Online Operational Decision-Making for Integrated Electric-Gas Systems with Safe Reinforcement Learning", *IEEE Transactions on Power Systems*, Vol. 39, No. 2, pp. 2893-2906, 2023.

[10] T.M. Alabi, L. Lu and Z. Yang, "Real-Time Automatic Control of Multi-Energy System for Smart District Community: A Coupling Ensemble Prediction Model and Safe Deep Reinforcement Learning", *Energy*, Vol. 304, pp. 1-6, 2024.

[11] J. Wu, Z. Huang, Z. Hu and C. Lv, "Toward Human-in-the-Loop AI: Enhancing Deep Reinforcement Learning Via Real-Time Human Guidance for Autonomous Driving", *Engineering*, Vol. 21, pp. 75-91, 2023.

[12] J. Chen, S. Li, K. Yang, C. Wei and X. Tang, "Deep Reinforcement Learning-based Integrated Control of Hybrid Electric Vehicles Driven by Lane-Level High-Definition Map", *IEEE Transactions on Transportation Electrification*, Vol. 10, No. 1, pp. 1642-1655, 2023.

[13] Z. Wang, S. Zhang, W. Luo and S. Xu, "Deep Reinforcement Learning with Deep-Q-Network based Energy Management for Fuel Cell Hybrid Electric Truck", *Energy*, Vol. 306, pp. 1-6, 2024.

[14] Q. Xing, Y. Xu, Z. Chen, Z. Zhang and Z. Shi, "A Graph Reinforcement Learning-based Decision-Making Platform for Real-Time Charging Navigation of Urban Electric Vehicles", *IEEE Transactions on Industrial Informatics*, Vol. 19, No. 3, pp. 3284-3295, 2022.

[15] X. Tang, J. Chen, K. Yang, M. Toyoda, T. Liu and X. Hu, "Visual Detection and Deep Reinforcement Learning-based Car Following and Energy Management for Hybrid Electric Vehicles", *IEEE Transactions on Transportation Electrification*, Vol. 8, No. 2, pp. 2501-2515, 2022.