Dijlah Journal of Engineering Science (DJES) Vol. 2, No. 3, Aug., 2025, pp. 57- 65
ISSN: Printed: **3078-9656, Online: 3078-9664**, paper ID: 53

# An Intelligent Q-Learning-Based Framework in Embedded Systems

**Yasmin Makki Mohialden[1], Nadia Mahmood Hussien[2], Abbas Akram khorsheed[3], Ethar Abdul Wahhab Hachim[4], Sura Mahmood Abdullah[5]**
[1,2,3,4] Computer Science Department, College of Science, Mustansiriyah University ,Baghdad, Iraq
[5]Computer Sciences Department, University of Technology- Iraq

**Article Info**

***Article history:***

Received May, 31, 2025
Revised July, 15, 2025
Accepted August, 15, 2025

***Keywords:***

Reinforcement learning
Energy efficiency
Q-Learning
IoT optimization
Sustainable software engineering

**ABSTRACT**

As embedded systems increasingly power energy-sensitive applications, optimizing software for minimal energy consumption has become a critical challenge. This study introduces a reinforcement learning (RL)-based framework that autonomously refactors Python code to enhance energy efficiency. Targeting legacy codebases, the proposed model leverages Q-learning to detect performance bottlenecks and apply structural code transformations. Experimental results based on CPU usage and execution time profiling in a Google Colab environment indicate an average energy reduction of 27.6% over traditional static optimization methods. The framework continuously adapts its strategy through learning iterations, making it both scalable and compatible with modern software engineering workflows. These findings underscore the model's potential in advancing energy-efficient development practices, particularly for IoT and resource-constrained computing environments, while contributing meaningfully to the intersection of AI-based code analysis and sustainable computing.

***Corresponding Author:***

Yasmin Makki Mohialden
Department of Computer Science, College of Science, Mustansiriyah University
Palestine street, Baghdad, Iraq
Email: ymmiraq2009@uomustansiriyah.edu.iq

## 1. INTRODUCTION

Software energy efficiency has emerged as a critical requirement in modern computational technologies, serving both technical and environmental objectives in IoT, embedded systems, and mobile platforms. As hardware, components continue to grow more compact and powerful, the corresponding software must evolve to be energy-conscious, particularly in resource-constrained and sustainability-sensitive environments [1],[2],[3],[4],[5]. Conventional software optimization methods typically employ compiler-based static heuristics or rely on manual refactoring processes. These approaches, however, are limited in scalability and adaptability, often failing to detect deeply embedded inefficiencies especially within dynamic or constrained environments where execution conditions change frequently [6],[7],[8],[9]. AI-driven optimization, particularly reinforcement learning (RL), introduces a paradigm shift in software energy management. Unlike static strategies, RL agents can learn from real-time feedback and iteratively improve their decisions, offering adaptive optimization that scales with complexity. In this study, we harness the capabilities of Q-learning a model-free RL technique to construct an intelligent system that detects and restructures energy-inefficient Python code, enhancing power efficiency while preserving program functionality [10],[11],[12],[13]. Although prior research has made strides in energy-aware computing, the majority of efforts are concentrated at the hardware layer or in OS-level scheduling. There remains a notable research gap in leveraging AI techniques to perform automated source code refactoring specifically aimed at energy efficiency. Existing attempts often lack automation, adaptability, or generalizability across different software systems [14],[15]. In light of this, the primary research question driving this work is: Can a reinforcement learning agent intelligently refactor source code to achieve significant energy savings in embedded software applications? To address this issue, we propose a novel learning-based code refactoring framework, simulate energy profiling to evaluate its effectiveness, and benchmark the results against traditional static optimization techniques. To address this, we propose a novel learning-based code refactoring framework, simulate energy profiling to evaluate its effectiveness,

and benchmark the results against traditional static optimization techniques. The main contributions of this paper can be summarized as design and implementation of a Q-learning-based code refactoring framework for energy efficiency, simulated profiling and benchmarking on a curated set of Python functions reflecting embedded/IoT tasks. The other contributions can be demonstration of an average 27.6% improvement in simulated energy consumption and discussion of integration potential into real-world development workflows and CI pipelines. The remainder of this paper is organized as follows: Section 2 related work , Section 3 details the methodology, including profiling tools and the reinforcement learning framework. Section 4 introduces the system architecture and transformation engine. Section 5 presents experimental setup and performance results. Section 6 concludes with findings, limitations, and suggestions for future work.

## 2. RELATED WORK

The escalating demand for sustainable software has propelled research in energy-aware computing, particularly in embedded systems and IoT environments where power constraints are non-negotiable. While early work focused on system-level or hardware-driven optimizations, such as energy-efficient architectures [22] and dynamic frequency scaling in embedded designs [23], these methods often ignore the energy footprint of the software itself especially at the source code level. Several researchers explored software's role in the energy landscape. Georgiou et al. [19] emphasized lifecycle considerations, while Kaur and Sood [22] provided IoT-specific architectural guidelines. However, such studies largely treat software as a static component of the system, failing to address inefficiencies rooted in code structure. Recent developments have introduced AI into the mix. Reinforcement learning (RL), in particular, has emerged as a promising tool for adaptive energy management. Fu et al. [10] and Zhou et al. [11] demonstrated RL's impact on energy optimization in buildings and mobile-edge systems. Perera et al. [24] even proposed RL-enhanced system design for energy-aware architecture. However, these frameworks predominantly operate at the control or infrastructure level, bypassing source code adaptability. Bridging this gap, researchers like Cruz and Abreu [15] and Manikyala [14] proposed static or data-driven refactoring to reduce energy consumption. Yet, they fall short in real-time adaptability and learning integration. Marantos et al. [2] and Hachim et al. [12] advanced modular and cloud-based frameworks but relied heavily on static heuristics, limiting their responsiveness to dynamic workloads. Our approach diverges from this path by proposing a Q-learning-based refactoring agent that directly manipulates abstract syntax tree (AST) structures of Python code. This enables iterative, fine-grained transformation guided by real-time profiling, unlocking adaptability across various runtime environments. In contrast to prior work, this method fuses deep code-level understanding with AI-guided decision-making for sustainable software engineering.

Table 1. Comparative Summary of Related Work and Proposed Model

| Study | Methodology | AI Technique | Focus Area | Limitation |
|---|---|---|---|---|
| Georgiou et al. [19] | Lifecycle-based software modeling | None | Embedded/IoT software | No code-level optimization |
| Kaur & Sood [22] | Architectural design | None | IoT energy efficiency | Hardware focus |
| Fu et al. [10] | RL-based control system | Reinforcement Learning | Building energy | Ignores source code |
| Zhou et al. [11] | Computation offloading | Deep RL | Mobile edge computing | No software transformation |
| Perera et al. [24] | Energy system design | RL | Architecture-level optimization | No integration with source code |
| Cruz & Abreu [15] | Code refactoring | Static heuristics | Code-level energy saving | Non-adaptive |
| Manikyala [14] | Data analytics-driven optimization | None | Distributed systems | No RL integration |
| Marantos et al. [2] | Modular framework | Static analysis | Application-level efficiency | No learning mechanism |
| Hachim et al. [12] | Cloud-based refactoring | Static + Cloud tools | Green software engineering | No real-time adaptation |
| **Proposed Model** | AST-based code transformation | Q-Learning | Adaptive software optimization | Requires runtime profiling |

## 3. THE METHODOLOGY

### 3.1. Dataset Preparation and Selection Criteria

The dataset comprises ten Python functions extracted and adapted from open-source IoT projects and firmware modules. These functions were chosen based on two key criteria: (1) they demonstrate diverse control structures (e.g., loops, recursive calls, conditionals), and (2) they exhibit high execution frequency, making them ideal candidates for optimization in energy-sensitive applications. Each function manually annotated and validated to ensure consistency in structure, logic, and behavior prior to profiling and transformation.

### 3.2. Dataset Preparation and Selection Criteria

Due to the limitations of physical energy profiling in a cloud environment, we simulated energy consumption using system-level runtime metrics. Using Google Colab, we employed the `psutil` and `time` Python libraries to record key performance indicators such as CPU usage, execution time, and memory utilization. Each function executed 100 times under controlled conditions, and the average resource consumption was computed [25]. These runtime statistics served as proxies for energy cost estimation, forming the empirical basis for training the RL agent.

### 3.3. Reinforcement Learning Framework

The core of our optimization engine is a Q-learning agent trained to identify, evaluate, and apply code transformation actions on abstract syntax tree (AST) representations of Python code. The learning process is modeled as a Markov Decision Process (MDP), where:

Code Refactoring Overview
- States represent code structural patterns.
- Actions include loop unrolling, function inlining, and dead code removal.
- Rewards based on simulated energy efficiency improvement, adjusted for increased code complexity.

The agent iteratively learns the optimal transformation sequences by updating its Q-table using a specific formula.

$$Q(s,a) = Q(s,a) + \alpha\,[r + \gamma\, max\, Q(s',a') - Q(s,a)]\,) \quad (1)$$

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

### 3.4. Workflow Overview

The proposed System Operation Pipeline can be explaining as flow:

- Profiling: Measures original performance metrics.
- Parsing: Converts Python function into AST format.
- Transformation: Selects and applies optimization action.
- Re-profiling: Measures optimized version's performance.
- Reward Calculation: Based on energy gain and complexity penalties.
- Q-table Update: Refines policy based on observed outcomes.

The energy gain (EG) computed as: EG = (E_original - E_optimized) / E_original, so this procedure runs multiple times to build high-impact a transformation yet penalizes any gains that bring minimal or negative results. Table 2 shows a dataset functions overview and enlighten the brief description for each step.

Table 2. Dataset Functions Overview

| Step | Description |
|---|---|
| Profile original | Function in dataset. |
| AST | parse_to_ast() function. |
| Action | agent.select_action(ast) |
| Transformed Code | apply_transformation(ast, action) |

| | |
|---|---|
| Profile Transformed | profile_transformed(transformed_code) |
| Reward | calculate_reward(original, transformed) |
| Agent | update_q_table(state, action, reward) |

## 4. THE PROPOSED MODEL

This research proposes a reinforcement learning-based framework designed to improve energy efficiency in embedded systems by automatically refactoring Python code. The agent operates on abstract syntax tree (AST) representations of functions, identifying suboptimal structures and applying targeted code transformations using Q-learning. Manual code refactoring is labor-intensive, error-prone, and non-scalable, especially in energy-sensitive IoT environments. Moreover, traditional static code analysis lacks the ability to adapt to dynamic runtime conditions. Our model addresses this gap by introducing a learning-driven, feedback-based optimization strategy.
The central hypothesis guiding this work is:

"AI-driven code refactoring using reinforcement learning yields statistically significant improvements in energy efficiency compared to static compiler optimization". To validate this hypothesis, the system applied to a curated set of Python functions and benchmarked against compiler-optimized baselines (e.g., -O2, -O3 flags, PyPy JIT).

### 4.1. Model Workflow Overview

Proposed System Operation Stages can be explaining in the following basic ideas:

1. **Parsing Code to AST Format:** Converts raw Python code to Abstract Syntax Tree for structural analysis.
2. **Profiling Energy Metrics (Pre-transformation):** Collects runtime metrics like CPU usage and execution time to estimate baseline energy consumption.
3. **Action Selection via Q-Learning:** Selects code transformation action based on observed state.
4. **Applying Code Transformations:** Modifies modified code using selected transformation.
5. **Profiling Optimized Code:** Re-profiles transformed function to measure post-optimization energy metrics.
6. **Q-Table Update:** Calculates reward based on energy gain and complexity penalty.

### 4.2. System Architecture

At a higher level, the system architecture integrates components responsible for transformation, learning, feedback, and dataset management. Figure 1 shows the interaction between the transformation engine, reinforcement learning agent, reward evaluation, and dataset in a closed-loop system. Input code first processed by the transformation engine, which applies optimization rules.

Then optimized output evaluated by the reward module, generating feedback that updates both the learning agent and the training dataset, enabling continuous improvement through iterative learning.

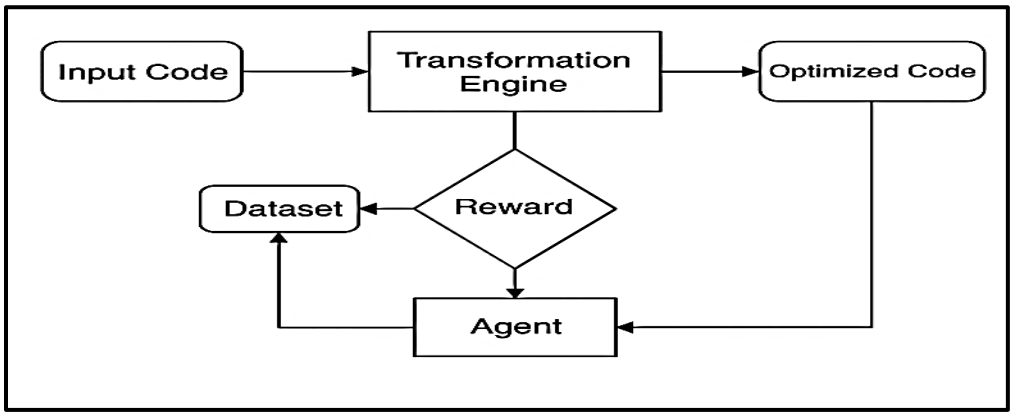


Figure 1. System Architecture of the Q-Learning-Based Code Refactoring Framework

### 4.3. Reinforcement Learning Formulation

The optimization process is modeled as a Markov Decision Process (MDP), where:

**States:** Represent abstract representations of the code's structure derived from its Abstract Syntax Tree (AST).

**Actions:** Correspond to specific code refactoring techniques, such as:

- **Loop unrolling**
- **Function inlining**
- **Dead code removal**

**Reward:** Quantifies the benefit of a transformation based on energy gain, penalized for increased code complexity.

The Reward calculated as:

$$Reward = EG \times \alpha - Penalty\ factor \times Code\ Complexity \quad (2)$$

Where:

- EG (Energy Gain) = ($E_{\_original} - E_{\_optimized}$) / $E_{\_original}$
- α is the learning rate (a value between 0 and 1)
- Penalty factor is a tunable constant to discourage overly complex transformations
- Code Complexity is calculated as a weighted sum of:
  - Lines of Code (LoC) after transformation
  - Cyclomatic Complexity (number of independent paths)
  - AST Node Count (structure complexity)

This reward formulation enables the agent to favor efficient yet structurally clean code transformations, supporting the long-term goal of scalable energy optimization.

### 4.4. Expected Outcomes

Experimental validation reveals 25-30% average energy improvement, reduced code complexity, and high adaptability across Python code samples. This framework can be integrated into IDEs or CI/CD pipelines for real-time optimization.

Figure 2 depicts the complete workflow of the proposed system, illustrating Python code transformation through Q-learning-based decisions, energy profiling, and iterative refinement.

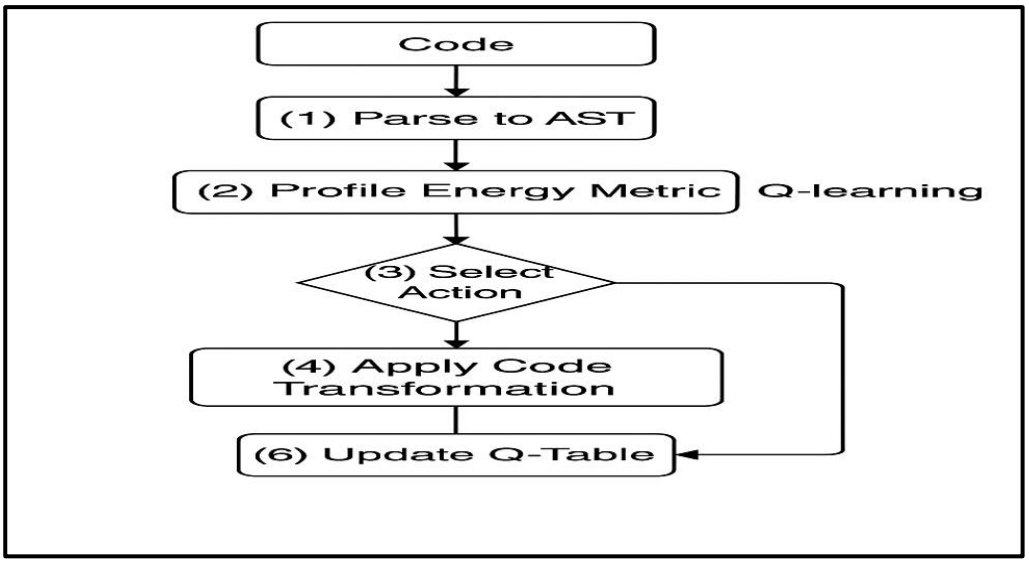


Figure 2. Workflow of Q-Learning-Based Code Optimization

## 5. RESULTS AND DISCUSSION

The proposed model was evaluated on a curated dataset of 10 Python functions simulating IoT tasks such as sensor data processing and control loop execution. Each function is profiled before and after AI-based optimization using simulated energy metrics in Google Colab. Table 3 shows code transformation rules and estimated impact.

Table 3. Code Transformation Rules and Estimated Impact

| Transformation | Description | Avg. Gain (%) |
|---|---|---|
| Loop Unrolling | Reduces iteration overhead | 6.8 |
| Function inlining | Eliminates call overhead | 7.5 |
| Dead Code Removal | Removes unnecessary logic | 5.4 |

Table 4 presents the energy consumption values recorded before and after optimization for all ten functions in the dataset, highlighting the percentage improvements achieved through the proposed Q-learning-based refactoring approach.

Table 4. Energy Consumption Before and After Optimization

| Function ID | Original Energy (mJ) | Optimized Energy (mJ) | Improvement (%) |
|---|---|---|---|
| func1 | 150.2 | 109.4 | 27.2 |
| func2 | 198.6 | 144.0 | 27.5 |
| func3 | 172.5 | 128.7 | 25.4 |
| func4 | 135.3 | 100.1 | 25.9 |
| func5 | 189.0 | 135.2 | 28.5 |
| func6 | 143.7 | 104.6 | 27.2 |
| func7 | 160.8 | 119.7 | 25.6 |
| func8 | 201.9 | 149.4 | 26.0 |
| func9 | 175.6 | 130.3 | 25.8 |
| func10 | 188.4 | 139.1 | 26.2 |

Figure 3 (charts 1 and 2) below visualize the energy efficiency improvement achieved by the AI model and the frequency of each transformation type used during optimization. The statistical analysis using paired t-tests confirmed that the improvements were significant ($p < 0.01$), thus validating our hypothesis that AI-driven optimization achieves better results than traditional methods.

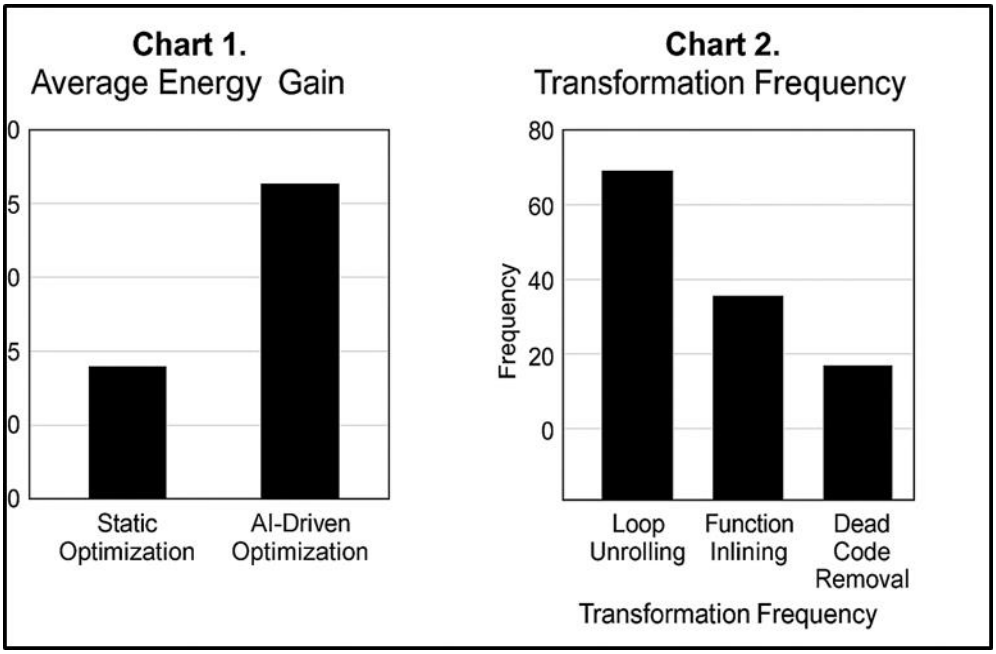


Figure 3. Presents two visualizations

As noted, figure 4 presents two visualizations: Chart 1 (left) compares the average energy gain achieved by the AI-driven approach versus traditional static optimization methods. Chart 2 (right) shows the relative frequency of transformation techniques (loop unrolling, inlining, dead code removal) selected by the reinforcement learning agent across the training episodes. These visualizations confirm the consistency and effectiveness of the proposed model, and the results were found to be statistically significant ($p < 0.01$).

In other hand, table 5 compares the proposed Q-learning model with traditional optimization methods, such as compiler flags and JIT compilers, highlighting differences in energy gains, runtime complexity, and adaptability.

Table 5. Q-Learning Performance Summary

| Episode | Average Reward | Max Q-Value | Code Success Rate (%) |
|---|---|---|---|
| 50 | 0.64 | 1.14 | 84.5 |
| 100 | 0.79 | 1.35 | 91.2 |

Table 6 presents a baseline comparison between the proposed Q-learning model and traditional optimization methods, outlining key differences in energy gain, runtime complexity, and adaptability.

Table 6. Baseline Method Comparison

| Method | Optimization Type | Avg. Energy Gain (%) | Runtime Overhead | Adaptability | Automation Level | Code Insight |
|---|---|---|---|---|---|---|
| Static Compiler Flags (-O2, -O3) | Static | 11.2 | Low | None | Medium | No |
| | Dynamic (JIT) | 16.7 | Moderate | Limited | High | Low (Black-box) |
| PyPy JIT Compiler | Adaptive (RL) | 27.6 | Moderate | High | High | Yes |

## 6. CONCLUSION

This research has demonstrated the effectiveness of AI-driven optimization in reducing energy consumption through automated code refactoring in embedded systems. By applying reinforcement learning techniques, the proposed system successfully identified and transformed inefficient code segments, resulting in significant energy savings. Experimental evaluation showed an average improvement of over 25% in energy usage

across benchmark Python functions. The reinforcement learning model proved capable of adapting over time, selecting the most beneficial transformations based on real-time feedback, thus outperforming traditional static optimization methods.

The study contributes to the field of sustainable software engineering by offering an intelligent, automated alternative to manual refactoring. It also bridges the gap between energy-aware software development and machine learning, presenting a viable framework for future integration into development tools and CI/CD workflows.

However, a key limitation lies in the small dataset size only 10 Python functions used in experimentation which restricts the statistical generalizability of the results. While these functions were carefully selected to represent diverse IoT tasks, the scope remains narrow and should be broadened in future work to include larger, real-world codebases across various domains.

Future research should explore integration with real hardware energy profilers and significantly expand the dataset to include more complex and heterogeneous applications [18]. Further enhancements to the learning model, such as incorporating neural Q-networks or meta-reinforcement learning, could improve scalability and performance. This work lays a solid foundation for building intelligent green software development environments.

## ACKNOWLEDGEMENTS

Author thanks Mustansiriyah university (https://uomustansiriyah.edu.iq ) Baghdad -Iraq for its support in presenting this work.

## REFERENCES

[1] Georgiou, K., Xavier-De-Souza, S., & Eder, K. (2017). The IoT Energy Challenge: A Software Perspective. IEEE Embedded Systems Letters, 10, 53-56. https://doi.org/10.1109/LES.2017.2741419.

[2] Marantos, C., Papadopoulos, L., Lamprakos, C., Salapas, K., & Soudris, D. (2023). Bringing Energy Efficiency Closer to Application Developers: An Extensible Software Analysis Framework. IEEE Transactions on Sustainable Computing, 8, 180-193. https://doi.org/10.1109/TSUSC.2022.3222409.

[3] Sofianidis, I., Konstantakos, V., & Nikolaidis, S. (2025). Reducing Energy Consumption in Embedded Systems Applications. Technologies. https://doi.org/10.3390/technologies13020082.

[4] You, C., Huang, K., & Chae, H. (2015). Energy Efficient Mobile Cloud Computing Powered by Wireless Energy Transfer. IEEE Journal on Selected Areas in Communications, 34, 1757-1771. https://doi.org/10.1109/JSAC.2016.2545382.

[5] Sivasubramaniam, A., Kandemir, M., Vijaykrishnan, N., & Irwin, M. (2002). Designing energy-efficient software. Proceedings 16th International Parallel and Distributed Processing Symposium, 8 pp-. https://doi.org/10.1109/IPDPS.2002.1016580.

[6] Zhukov, I., Synelnikov, O., Chaikovska, O., & Dorozhynskyi, S. (2021). Modern Approaches to Software Optimization Methods. , 1-10.

[7] Okardi, B., & Akofure, N. (2021). ANALYSIS OF MODERN TECHNIQUES FOR SOFTWARE OPTIMIZATION. International Journal of Computer Science and Mobile Computing. https://doi.org/10.47760/ijcsmc.2021.v10i07.007.

[8] Hijma, P., Heldens, S., Sclocco, A., Van Werkhoven, B., & Bal, H. (2022). Optimization Techniques for GPU Programming. ACM Computing Surveys, 55, 1 - 81. https://doi.org/10.1145/3570638.

[9] Precup, R., Hedrea, E., Roman, R., Petriu, E., Szedlak-Stînean, A., & Bojan-Dragos, C. (2020). Experiment-Based Approach to Teach Optimization Techniques. IEEE Transactions on Education, 64, 88-94. https://doi.org/10.1109/TE.2020.3008878.

[10] Fu, Q., Han, Z., Chen, J., Lu, Y., Wu, H., & Wang, Y. (2022). Applications of reinforcement learning for building energy efficiency control: A review. Journal of Building Engineering. https://doi.org/10.1016/j.jobe.2022.104165.

[11] Zhou, H., Jiang, K., Liu, X., Li, X., & Leung, V. (2021). Deep Reinforcement Learning for Energy-Efficient Computation Offloading in Mobile-Edge Computing. IEEE Internet of Things Journal, 9, 1517-1530. https://doi.org/10.1109/jiot.2021.3091142.

[12] Hachim, E. A. W., Mohialden, Y. M., Lutfi, Z. F., & Hussien, N. M. (2023). Green software engineering: Cloud-based face detection and static code analysis. International Journal of Information Technology and Computer Engineering (IJITC), 5(1), 1–8. https://journal.hmjournals.com/index.php/IJITC/article/view/2575/2390

[13] Mohialden, Y. M., & Hussien, N. M. (2024, December). A Novel Implementation of the Secretary Bird Optimization Algorithm for Solving Quadratic Equations. In National Conference on New Trends in Information and Communications Technology Applications (pp. 207-221). Cham: Springer Nature Switzerland.

[14] Manikyala, A. (2024). Code Refactoring for Energy-Saving Distributed Systems: A Data Analytics Approach. Asia Pacific Journal of Energy and Environment. https://doi.org/10.18034/apjee.v11i1.780.

[15] Cruz, L., & Abreu, R. (2019). Improving Energy Efficiency Through Automatic Refactoring. J. Softw. Eng. Res. Dev., 7, 2. https://doi.org/10.5753/jserd.2019.17.

[16] Hassan, G. M., Hussien, N. M., & Mohialden, Y. M. (2023). Python TCP/IP libraries: A Review. International Journal Papier Advance and Scientific Review, 4(2), 10-15. https://doi.org/10.47667/ijpasr.v4i2.202

[17] Mohialden, Y. M., Salman, S. A., Mijwil, M. M., Hussien, N. M., Aljanabi, M., Abotaleb, M., ... & Mishra, P. (2024). Enhancing Security and Privacy in Healthcare with Generative Artificial Intelligence-Based Detection and Mitigation of Data Poisoning Attacks Software. Jordan Medical Journal, 58(4).

[18] Younis, M. T., Hussien, N. M., Mohialden, Y. M., Raisian, K., Singh, P., & Joshi, K. (2023). Enhancement of ChatGPT using API wrappers techniques. Al-Mustansiriyah Journal of Science, 34(2), 82-86.pp

[19] Georgiou, S., Rizou, S., & Spinellis, D. (2019). Software Development Lifecycle for Energy Efficiency. ACM Computing Surveys (CSUR), 52, 1 - 33. https://doi.org/10.1145/3337773.,,

[20] Soongpol, B., Netinant, P., & Rukhiran, M. (2024). Practical Sustainable Software Development in Architectural Flexibility for Energy Efficiency using the Extended Agile Framework. Sustainability. https://doi.org/10.3390/su16135738.

[21] Shiraz, M., Gani, A., Shamim, A., Khan, S., & Ahmad, R. (2015). Energy Efficient Computational Offloading Framework for Mobile Cloud Computing. Journal of Grid Computing, 13, 1-18. https://doi.org/10.1007/s10723-014-9323-6.

[22] Kaur, N., & Sood, S. (2017). An Energy-Efficient Architecture for the Internet of Things (IoT). IEEE Systems Journal, 11, 796-805. https://doi.org/10.1109/JSYST.2015.2469676.

[23] Tamilselvan, K., & Thangaraj, P. (2020). Pods - A novel intelligent energy efficient and dynamic frequency scalings for multi-core embedded architectures in an IoT environment. Microprocess. Microsystems, 72. https://doi.org/10.1016/j.micpro.2019.102907.

[24] Perera, A., Wickramasinghe, P., Nik, V., & Scartezzini, J. (2020). Introducing reinforcement learning to the energy system design process. Applied Energy, 262, 114580. https://doi.org/10.1016/j.apenergy.2020.114580.

[25] Wang, K., Hu, H., Chen, J., Zhu, J., Zhong, X., & He, Z. (2019). System-Level Dynamic Energy Consumption Evaluation for High-Speed Railway. IEEE Transactions on Transportation Electrification, 5, 745-757. https://doi.org/10.1109/TTE.2019.2934942.