# Power-Saving IoT Anomaly Detection: Pareto Front-Optimized Dynamic Tuner*

Gyuha Son, Seo-Yi Kim, and Il-Gu Lee†
{20231086, iglee}@sungshin.ac.kr, sykim.cse@gmail.com

**Abstract**

The rapid proliferation of the Internet of Things (IoT) has highlighted the dual importance of security and sustainability in its operation. In this paper, we propose a Pareto Front Optimized Dynamic Tuner that adaptively selects an adequate hyperparameter set according to the battery state of IoT devices. The battery is divided into three states: Normal, Limit, and Crisis. Each state is assigned a Minimum F1-threshold, complexity weight, and latency weight. Unlike static operation, which fixes hyperparameters throughout execution, the proposed tuner dynamically chooses a parameter set from the Pareto Front based on the policy mechanism corresponding to the current battery state. Through virtual battery simulations, we demonstrate that the proposed method achieves an F1-score comparable to the conventional method while significantly extending battery lifetime. These results highlight the effectiveness of the dynamic tuner in mitigating the inherent trade-off between security and sustainability in resource-constrained IoT environments.

**Keywords—**pareto front optimization, IoT anomaly detection, dynamic tuner, power saving

## 1 Introduction

The proliferation of the Internet of Things (IoT) has significantly enhanced convenience and connectivity in modern society, offering benefits such as real-time monitoring and automation. However, IoT devices face challenges in ensuring long-term reliability due to inherent structural limitations, such as limited battery capacity and computational power. Security functions, particularly anomaly detection, accelerate battery depletion as they require continuous computation and communication. Guo et al. [1] reported that integrating anomaly detection algorithms into IoT devices leads to a substantial increase in energy consumption caused by data processing and transmission. This creates a fundamental dilemma: enhancing security inevitably accelerates battery depletion. Consequently, lightweighting security functions has been considered crucial to address this challenge.

Nevertheless, prior research on IoT lightweighting has primarily focused on improving performance metrics such as the F1-score. In resource-constrained IoT environments, battery exhaustion can degrade system availability and ultimately disable security functions.

---

* Proceedings of the 9th International Conference on Mobile Internet Security (MobiSec′25), Article No. 73, December 16-18, 2025, Sapporo, Japan. 

† Corresponding author

In 2023, Kuaban et al. [2] experimentally demonstrated that various Energy-drain Attacks increase energy consumption in IoT devices, leading to premature battery depletion. Their attack scenarios rendered devices incapable of providing normal service due to premature battery depletion. These findings indicate that maximizing detection performance alone cannot guarantee long-term security; instead, an approach that simultaneously considers detection performance and operational sustainability is necessary.

Most prior studies have relied on static policies with a fixed configuration. However, IoT devices should adopt policies tailored to the real-time state to achieve optimal results within constrained environments since real-world IoT environments change dynamically. For example, detection performance may be the top priority when the battery is in a normal state, whereas minimizing power consumption becomes more essential when the battery reaches a critical state. This study proposes a battery-based dynamic operation tuner that balances detection performance and battery sustainability.

The main contributions of this research are as follows.

- This study proposes a dynamic operation method that utilizes the Pareto Front to find the optimal pipeline hyperparameter combination based on battery status.
- The experiments demonstrated that the proposed method extends battery life by up to 1.1 times relative to the conventional method while maintaining comparable performance. The distinctive contribution of this work lies in the quantitative consideration of both operational sustainability and detection performance, moving beyond previous research that focused solely on detection capabilities.

The remainder of this paper is as follows. Section 2 analyzes the limitations of existing research, while Section 3 proposes a Pareto Front-based dynamic pipeline tuning method. Section 4 evaluates the experiments conducted to demonstrate the effectiveness of the proposed method and analyzes the experimental results. Lastly, Section 5 discusses limitations and concludes the paper.

# 2 Related Work

Umar et al. [3] proposed a lightweight deep learning-based Intrusion Detection System (IDS) framework, termed Deep Neural Networks (DNN)-K-means Dynamic Quantization (KDQ), to improve energy efficiency by reducing latency and computational load in edge computing environments. The framework utilizes Adaptive Sampling and Sliding Window techniques to dynamically adjust data processing rates, while it also reduces the computational complexity and the model size through KDQ. The results showed a reduction in model size and inference latency compared to conventional IDS while maintaining high detection accuracy. Although the approach presented an efficient detection method in terms of model lightweighting, it lacked practical validation regarding battery sustainability, which is a fundamental challenge in IoT devices.

Halgamuge and Niyato [4] proposed an Artificial Intelligence (AI) and Machine Learning (ML)-based Adaptive Edge Security Framework to enhance security and adaptability. This model dynamically applies the most suitable security policy for the current IoT network situation by integrating risk assessment results with policy decisions. This approach reduced the average risk score of the IoT network from 0.17 to 0.15, corresponding to an 11.76% decrease compared to static policies. However, a key limitation remains: although resource constraints such as battery and CPU were considered in the policy design, no quantitative result was provided to evaluate these factors.

Esra Altulaihan et al.[5] proposed an anomaly-based Intrusion Detection System (IDS) with minimized storage and computational overhead for detecting DoS attacks in resource-constrained IoT networks. To achieve this, they employed an anomaly-based approach that detects deviations from

normal profiles and constructed a pipeline that performs feature selection and comparative analysis of various classifiers after preprocessing. The proposed method in this paper is not limited to signature-based approaches and can respond to mutated or newly emerging attacks. It also attempts a lightweight design through feature selection. However, it lacks sufficient quantitative analysis to substantiate these claims and does not address limitations such as power consumption resulting from model lightweighting.

Previous studies have primarily focused on model lightweighting and detection performance. However, they provide insufficient consideration for battery/power consumption, which are the most critical constraints for actual IoT devices. To ensure the availability of IoT systems, it is essential to consider the balance between detection performance and battery/power consumption.

# 3 Methodology

Our proposed method, Pareto Front Optimized Dynamic Tuner, dynamically selects an optimal hyperparameter combination based on the remaining battery level, whereas the conventional method operates with a fixed hyperparameter combination.

## 3.1 Pareto Front

The Pareto Front used in this study refers to the boundary formed by efficient solutions that remain undominated by any other combinations when no single combination can simultaneously optimize multiple metrics such as F1-score and Latency. By utilizing the Pareto Front, the dynamic tuner can swiftly select a parameter combination most suitable for the current battery state and performance requirements from within known efficient candidate sets, without the need to explore all possible combinations.

## 3.2 Proposed Method

In this study, the pipeline for IoT network anomaly detection is shown in Figure 1. During Feature Selection, the dataset is reduced by selecting the top-k features based on feature importance values obtained through Light Gradient-Boosting Machine (LightGBM) model training. LightGBM, with its tree-based structure, is more robust to noise than linear models and can efficiently identify important features without additional computations.

Subsequently, the Feature Extraction stage employs Stacked Denoising AutoEncoder (SDAE) and Random Projection (RP). SDAE is an autoencoder trained to reconstruct input data with noise injection, thereby enhancing robustness. Additionally, RP is applied because it can operate on any input values that can be converted into numerical vector values, thereby reducing constraints on input format. The output from SDAE is then compressed into a lower dimension through RP. Finally, in the Classification stage, anomaly detection is performed using a LightGBM classifier, which is lightweight and efficient in handling large-scale datasets.

While various hyperparameter combinations can be applied when configuring the pipeline, this study focuses on three parameters as targets for dynamic adjustment: the number of top features selected in Feature Selection ($k$), the dropout rate of SDAE ($p$), and the dimensionality of RP ($d_{\mathrm{RP}}$). These candidate values for each hyperparameter were set as Table 1 for the current experiment, and they can be expanded in future work. The candidate values for each hyperparameter are summarized in Table 1. Accordingly, the number of possible hyperparameter combinations is $|k|\times|p|\times|d_{\mathrm{RP}}| = 7\times3\times4=84$.

| Parameter | Symbol | Candidate Values |
|---|---|---|
| Top-k features | $k$ | {8, 16, 20, 32, 36, 40, 64} |
| Dropout rate | $p$ | {0.1, 0.3, 0.5} |
| Random Projection Dimension | $d_{\mathrm{RP}}$ | {8, 16, 32, 64} |

**Table 1: Hyper-parameter**

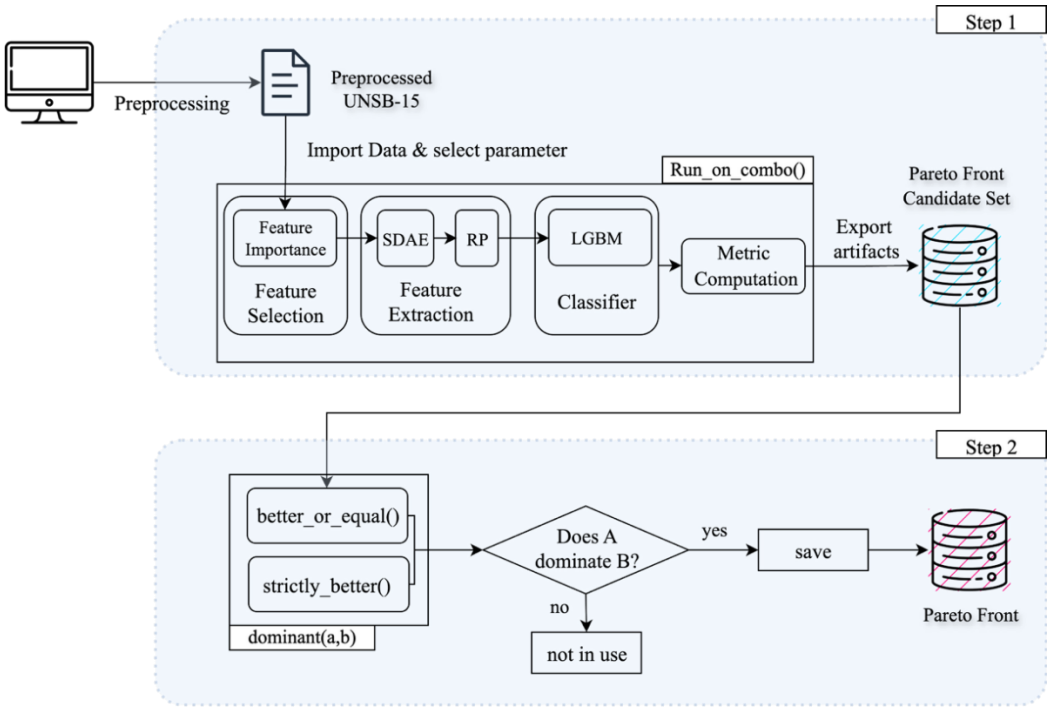


**Figure 1.** Building Pareto Front

**[Step 1] run_on_combo():** The minimum execution unit in this experiment is the function run_on _combo(). This function executes the aforementioned pipeline for each combination, computes performance metrics, and saves resulting artifacts. The performance metrics computed by this function are F1-score, Latency, and Computational Complexity (hereafter referred to as complexity).

Complexity is a score representing the weighted sum of the computational complexity of the model components, and is calculated using the following formula:

$$active = 1 - dropout\_rate$$
$$complexity = 0.6 \times (number\ of\ features \times active) + 0.25 \times (RP\ dimension) + 0.1 \times log(1 + number\ of\ SDAE\ parameters) + 0.05 \times log(1 + number\ of\ classifier\ parameters)$$

In the final stage of the function, a metadata file is generated. This file stores details about the input parameters ($k, p,\ d_{\mathrm{RP}}$), performance metrics (F1-score, latency_ms, complexity), the number of SDAE

parameters, the bottleneck dimension, and the LightGBM threshold value. After each execution of run_on_combo(), the metadata corresponding to the evaluated combination is appended to a file referred to as the Pareto Front Candidate Set, which aggregates the results of all combinations. Once all executions are complete, the Pareto Front Candidate Set contains information on a total of 84 Pareto Front candidates.

**[Step 2] Pareto Front:** The generated Pareto front candidates are further refined by applying the concept of Pareto optimality to retain only those combinations that are not dominated by any others. The procedure for building the Pareto front is as follows. First, the Pareto Front Candidate Set is loaded, and metrics requiring numerical operations—namely F1-score, latency_ms, and complexity—are converted into numeric types. Any candidates with missing values in these metrics are excluded, leaving only comparable candidates. Subsequently, dominant(a, b) function is applied to filter and retain candidates that remain undominated by others, thereby building the Pareto front.

**Algorithm 1. Pareto Front Dominance**

```
function dominant(a, b):
    better_or_equal =
        (a.f1 >= b.f1) AND
        (a.latency_ms <= b.latency_ms) AND
        (a.complexity <= b. complexity)
    strictly_better =
        (a.f1 > b.f1) OR
        (a.latency_ms < b.latency_ms) OR
        (a. complexity < b. complexity)
    return (better_or_equal AND strictly_better)
```

The dominant(a, b) function compares performance metrics to determine whether Candidate A dominates Candidate B. The dominance relation is defined as follows.

1. better_or_equal: Candidate A is equal to or better than Candidate B in all metrics.
2. strictly_better: Candidate A is better in at least one metric.

If both conditions are satisfied simultaneously, then Candidate A is considered to dominate Candidate B. If Candidate B dominates Candidate A even once, Candidate A cannot be part of the Pareto front. Only candidates that remain undominated by all other candidates can be part of the Pareto front. In the current experiment, the Pareto Front consisted of 25 solutions.

## 3.3 Operation Process

In this experiment, the battery capacity(100%) is divided into three states using thresholds of 60% and 30%, defined as Normal (≥60%), Limit (30%~60%), and Crisis (≤30%). Once the battery level determines the battery state, a hyperparameter set is selected according to the policy mechanism defined for that state. Each policy mechanism consists of three elements: the Minimum F1-threshold, complexity weight α, and latency weight ß. Although the specific values of these elements differ across states, the overall structure of the policy mechanism remains consistent. The policy mechanism operates as follows.

(1) **Candidate Filtering**

: Only the combinations that satisfy F1-score ≥ Minimum F1-threshold are retained.

(1a). **Relaxing Minimum F1-threshold**

: If no candidate satisfies (1), the Minimum F1-threshold is relaxed by 0.01 up to a maximum of three times.

(2) **Utility Score (U) Calculation**

: Calculate the U for combinations satisfying (1).

The formula for calculating U is as follows.

$$LAT\_norm = 1.0 - normalize\_to\_0\_1(Latency)$$
$$CPX\_norm = 1.0 - normalize\_to\_0\_1(Complexity)$$
$$U = \alpha * (CPX\_norm) + \beta * (LAT\_norm)$$

(3) **Priority Rule**

In this study, the optimal combination is selected based on the following priority rule. First, the combination with the highest U score is selected. If the U scores are identical, the combination with the higher F1-score is selected. If the F1-scores are identical, the combination with the lower latency is selected. Finally, if the latency is also identical, the combination with the lower complexity is selected.

The default settings for a virtual battery simulation are as follows. The total battery capacity (*CAPACITY_mAh*) was set to 1000mAh with a base load(*BASE_mA*) assumed to consume 8mA. The coefficient for converting complexity to mA(*Com_TO_mA*) is set to 0.8. The integration interval (*DT_SET*) is set to 1 second, integrating in seconds to reduce the remaining battery capacity.

$$\text{CAPACITY\_mAh} = 1000.0, \text{BASE\_mA}=8.0, \text{Com\_TO\_mA}=0.8, \text{DT\_SET}=1.0$$
$$ImA = BASE\_mA + Com\_TO\_mA * row["complexity"]$$
$$batt\_mAh -= ImA * (DT\_SEC / 3600.0)$$

The current consumption, denoted as *ImA*, is defined as the sum of the base load and complexity converted to milliamperes(mA). The simulation operates by calculating *ImA* every second and deducting it from the remaining battery capacity. Additional overhead is incorporated during pipeline switching, which is categorized into time overhead and energy overhead. For the time overhead, the battery is drained by the current consumed by the existing pipeline during the switching period. For the energy overhead, the energy cost associated with the switching operation itself is converted to milliampere-hours(mAh) and subtracted from the battery capacity.

# 4 Evaluation and Analysis

The objective of this experiment is to compare and evaluate static detection policies and battery-based dynamic detection policies in a battery-constrained environment such as IoT. To this end, we constructed a virtual battery-based simulation environment and quantitatively analyzed the impact of each policy on detection performance and battery lifetime. The experimental procedure is illustrated in Figure 2.

## 4.1 Evaluation environment

In this experiment, we utilized the training and testing CSV files from the UNSW-NB15 dataset [6]–[10] and converted them into a trainable format through preprocessing. The preprocessing procedure is as follows. The UNSW-NB15 dataset contains two columns: *label* and *attack_cat*. Among them, the

*label* column, which indicates whether the network traffic is normal (0) or anomalous (1), was designated as the target variable. The *attack_cat* column, which specifies the attack type for multi-class classification, was excluded as it was not required for this experiment.

Subsequently, the *label* and *attack_cat* columns were excluded to construct the input feature matrix solely with numerical features. One-hot encoding was applied to represent categorical features as new columns, and to ensure dimensional consistency between the training and test sets. Furthermore, missing and infinite values in the data were replaced with zeros to maintain stability, as such values cannot be processed by the model. The data was then standardized to reduce distributional differences among input feature values and facilitate stable model training. Specifically, a StandardScaler was utilized to set the mean to 0 and the variance to 1, and the same transformation was used for both training and test datasets.

As a result of preprocessing, the dataset was transformed into a form suitable for training. The final training set consisted of 175,341 samples and 192 features, while the test set consisted of 82,332 samples and 192 features.

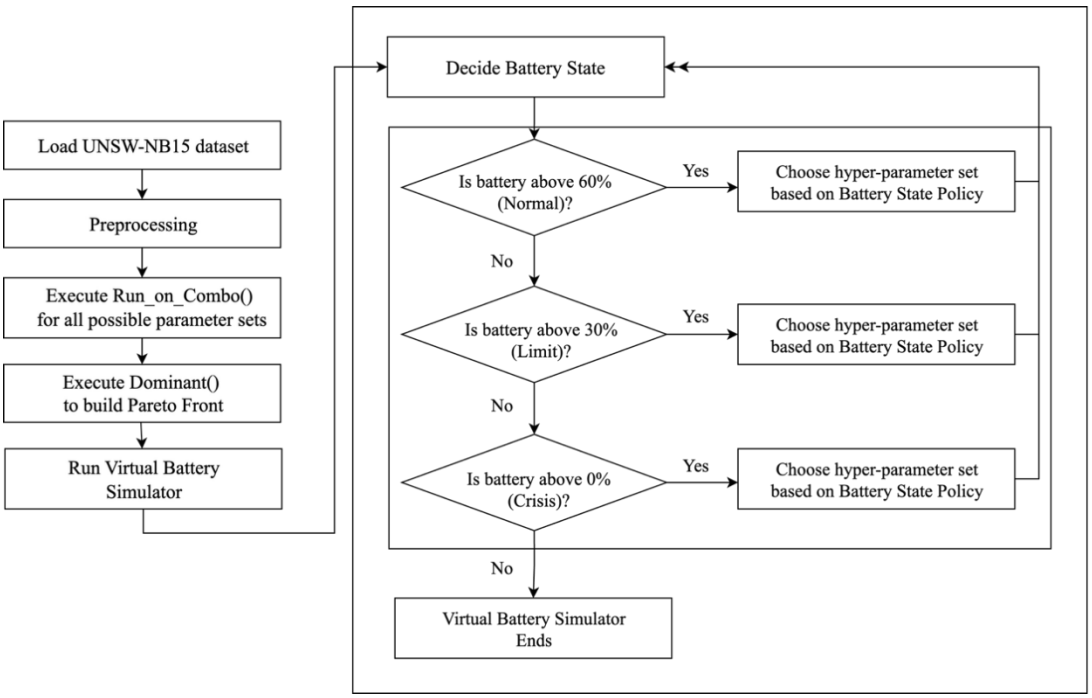


**Figure 2.** Experimental process flowchart

To build a Pareto Front Candidate Set, the function run_on_combo() was executed for all possible combinations of the hyperparameters listed in Table 1, resulting in a total of 84 candidates. By applying Algorithm 1, a Pareto Front consisting of 25 solutions was obtained.

For the static policy experiment, the battery was reduced using only one hyperparameter combination: the one with the highest F1-score from the Pareto Front. In this experiment, the combination with ($k$=64, $p$=0.3, $d_{RP}$=16) was selected as it has the highest F1-score of 0.88.

Dynamic policy experiment consumed battery by selecting an optimal combination based on the operational policy defined for each battery state. In this study, a dynamic policy experiment was

conducted in two configurations: one with a high Minimum F1-threshold (Dynamic V1) and another with a low Minimum F1-threshold (Dynamic V2). The Minimum F1-threshold applied to the normal, limit, and crisis states for Dynamic V1 and Dynamic V2 are summarized in Table 2. In addition, the complexity weight α and latency weight β were set to (0.3, 0.7) in the Normal state, (0.5, 0.5) in the Limit state, and (0.7, 0.3) in the Crisis state, respectively.

| Minimum F1-threshold | Dynamic V1 | Dynamic V2 |
|---|---|---|
| Normal | 0.88 | 0.68 |
| Limit | 0.87 | 0.65 |
| Crisis | 0.86 | 0.63 |

**Table 2: Minimum F1-threshold per experiment**

## 4.2 Evaluation result and analysis

To demonstrate the performance of the proposed method, a comparison with the conventional method was conducted. The experiment results are shown in Table 3.

| Experiment Version | Static | Dynamic V1 | Dynamic V2 |
|---|---|---|---|
| Total_time | 29.761 | 32.636 | 50.942 |
| Average F1-score | 0.880 | 0.877 | 0.779 |
| Average Latency_ms | 0.032 | 0.032 | 0.026 |
| Average Complexity | 32.001 | 28.301 | 14.538 |

**Table 3: Experiment Result**

For the static policy experiment, the time taken to fully deplete the battery was 29.76 hours, with an average F1-score of 0.880, average latency of 0.032, and average complexity of 32.001.

The policy of the pipeline in Dynamic V1 started with the combination of ($k$=64, $p$=0.3, $d_{\mathrm{RP}}$=16) and switched to the combination of ($k$=64, $p$=0.3, $d_{\mathrm{RP}}$=8) when the battery level reached 54%. When the battery level reached 27%, the combination switched to the combination of ($k$=64, $p$=0.5, $d_{\mathrm{RP}}$=8). The time taken to fully deplete the battery was 32.636 hours, with an average F1-score of 0.877, average latency of 0.032, and average complexity of 28.301.

The policy of the pipeline in Dynamic V2 started with the combination of ($k$=40, $p$=0.5, $d_{\mathrm{RP}}$=16) and switched to the combination of ($k$=16, $p$=0.3, $d_{\mathrm{RP}}$=8) when the battery level reached 28%. The time taken to fully deplete the battery was 50.942 hours, with an average F1-score of 0.779, average latency of 0.026, and average complexity of 14.538. The performance comparison graph between the conventional method and the proposed method is shown in Figure 3.

In the conventional method, the battery was depleted in 29.7 hours while consistently maintaining F1-score. Compared to the conventional method, Dynamic V1 maintained an F1-score similar to that of the static baseline. It also gradually reduced power consumption, thereby extending battery lifetime by approximately 1.1 times. Dynamic V2, which relaxed the Minimum F1-threshold, exhibited slightly degraded performance in terms of F1-score; however, it significantly reduced power consumption, extending battery life by 1.71 times compared to the conventional method and 1.56 times compared to Dynamic V1. Therefore, Dynamic V1 can be interpreted as a balanced policy between performance and battery efficiency, whereas Dynamic V2 represents a policy that prioritizes battery life over performance.

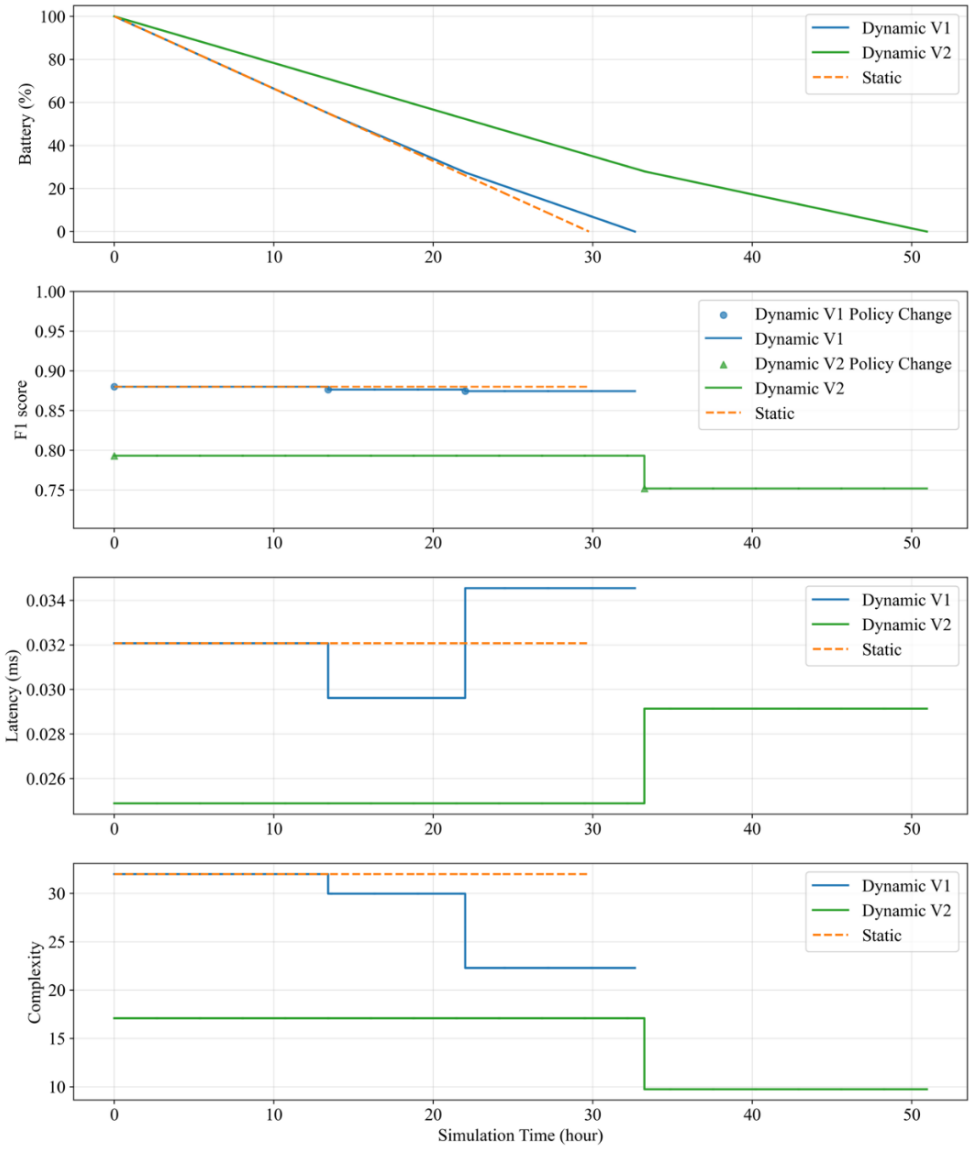


**Figure 3. Evaluation result:**
**(a) battery over time (2) F1-score over time (3) Latency over time (4) Complexity over time**

# 5 Conclusion

This study investigates a battery-aware tuning technique that dynamically selects parameter combinations for a pipeline based on the current battery status. The proposed method selects a hyperparameter combination from a Pareto Front that satisfies a Minimum F1-threshold and achieves the highest weighted score, based on the real-time remaining battery level. Experimental results demonstrate that the dynamic tuner can significantly extend battery life while maintaining the F1-score

at a level comparable to that of the static baseline by flexibly adjusting parameter combinations as battery consumption progresses. These findings suggest that the proposed approach can serve as a practical operating policy for real-world IoT devices with constrained power resources.

However, this study has limitations as experiments were performed on a virtual battery simulation, necessitating further validation on real hardware environments. Furthermore, as the Pareto Front Candidate Set in this study was restricted to a total of 84, future work will focus on building a more efficient Pareto Front by adding more hyperparameter candidate values.

**Acknowledgement.** This work is supported by the Ministry of Trade, Industry and Energy (MOTIE) under Training Industrial Security Specialist for High-Tech Industry [grant number RS-2024-00415520] supervised by the Korea Institute for Advancement of Technology (KIAT), the Ministry of Science and ICT (MSIT) under the ICAN (ICT Challenge and Advanced Network of HRD) program [grant number IITP-2022-RS-2022-00156310] and National Research Foundation of Korea (NRF) grant [RS-2025-00518150], and the Information Security Core Technology Development program [grant number RS-2024-00437252] supervised by the Institute of Information & Communication Technology Planning & Evaluation (IITP).

# References

1. Guo, Hongtai, et al. "EGNN: Energy-efficient anomaly detection for IoT multivariate time series data using graph neural network." *Future Generation Computer Systems* 151 (2024): 45-56.
2. Kuaban, Godlove Suila, et al. "Modelling of the energy depletion process and battery depletion attacks for battery-powered internet of things (iot) devices." *Sensors* 23.13 (2023): 6183.
3. Umar, Hafiz Gulfam Ahmad, et al. "Energy-efficient deep learning-based intrusion detection system for edge computing: a novel DNN-KDQ model." *Journal of Cloud Computing* 14.1 (2025): 32.
4. Halgamuge, Malka N., and Dusit Niyato. "Adaptive edge security framework for dynamic IoT security policies in diverse environments." *Computers & Security* 148 (2025): 104128.
5. Altulaihan, Esra, Mohammed Amin Almaiah, and Ahmed Aljughaiman. "Anomaly detection IDS for detecting DoS attacks in IoT networks based on machine learning algorithms." *Sensors* 24.2 (2024): 713.
6. Moustafa, Nour, and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." Military Communications and Information Systems Conference (MilCIS), 2015. IEEE, 2015.
7. Moustafa, Nour, and Jill Slay. "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 dataset and the comparison with the KDD99 dataset." Information Security Journal: A Global Perspective (2016): 1-14.
8. Moustafa, Nour, et al. "Novel geometric area analysis technique for anomaly detection using trapezoidal area estimation on large-scale networks." IEEE Transactions on Big Data (2017).
9. Moustafa, Nour, et al. "Big data analytics for intrusion detection system: statistical decision-making using finite dirichlet mixture models." Data Analytics and Decision Support for Cybersecurity. Springer, Cham, 2017. 127-156.
10. Sarhan, Mohanad, Siamak Layeghy, Nour Moustafa, and Marius Portmann. NetFlow Datasets for Machine Learning-Based Network Intrusion Detection Systems. In Big Data Technologies and Applications: 10th EAI International Conference, BDTA 2020, and 13th EAI International Conference on Wireless Internet, WiCON 2020, Virtual Event, December 11, 2020, Proceedings (p. 117). Springer Nature.