Energy Bugs in Object Detection Software on Battery-Powered Devices

Ippo Hiroi
Department of Computer Science
University of Tsukuba
Tsukuba, Japan
hiroi.ippo@sd.cs.tsukuba.ac.jp

Fumio Machida
Department of Computer Science
University of Tsukuba
Tsukuba, Japan
machida@cs.tsukuba.ac.jp

Ermeson Andrade
Department of Computing
Federal Rural University of Pernambuco
Recife, Brazil
ermeson.andrade@ufrpe.br

***Abstract*—Object detection software is widely adopted in edge computing systems such as mobile devices, drones, and autonomous robots. These edge devices are often battery-powered and, hence, confront stringent requirements for efficient computation and energy saving. Energy bugs that cause inefficient software execution pose significant risks to object detection systems running on battery-powered edge devices. However, such risks have not been well discussed in the previous research. This paper aims to investigate potential energy bugs in object detection software running on a battery-powered device. First, we searched for reports of known energy bugs from internet forums and bug-tracking systems for YOLOv5 and Mask R-CNN (Mask Region-based Convolutional Neural Network). Although we did not find direct mentions of energy bugs, some reports hinted at energy inefficiencies. Consequently, we conducted experiments to assess the impact of energy bugs by monitoring resource utilization and battery consumption of an object detection system running YOLOv5 on a battery-powered Raspberry Pi. Our experimental results clearly show increased power consumption caused by the omission of the input image size setting and improper deployment of the unquantized model, which can be regarded as potential energy bugs in object detection software.**

***Index Terms*—Object detection, Battery-powered devices, Energy bugs, Energy consumption.**

I. Introduction

In recent years, mobile edge devices, such as Automated Guided Vehicles (AGVs), Autonomous Mobile Robots (AMRs), and drones are increasingly used in various fields, including agriculture, logistics, and disaster response [1] [2]. These mobile edge devices are often powered by batteries. Given the limited capacity of batteries that can be accommodated on such devices, it is crucial to save as much battery power as possible and operate them in an energy-efficient manner [3]. Energy-efficient operation is particularly important in mission-critical applications, such as disaster rescue, in which a long and continuous operation is essential with limited battery resources.

Object detection software is a promising Artificial Intelligence (AI) technology recently introduced in many edge devices. These algorithm enables real-time identification and understanding of objects in the real world from the video input. However, object detection processing is a computation-intensive task that consumes a large amount of resources and power, potentially compromising the device's ability to operate continuously. Inefficient object detection algorithms or their implementation with improper settings can also lead to unnecessary power consumption [4]. As a result, wasteful power usage due to software inefficiencies is expected to become a more prevalent issue in battery-powered edge devices.

In this paper, we focus on the issue of *energy bugs* that can affect the execution of object detection software on battery-powered devices. Energy bugs are software faults that cause a larger power consumption during the operation than the expected power consumption by design. These inefficiencies arise from improper management of CPU and memory resources, inefficient processing, and unnecessary background processes. Existing studies of energy bugs mainly target smartphones, where complex configurations and poor resource management can lead to faster battery drain [5]. However, the issue and impact of energy bugs in object detection software running on edge devices have not been largely discussed in the previous literature. Energy bugs might be a significant risk in object detection software used in a battery-powered edge device.

To address this gap, this study aims to characterize potential risks of energy bugs in object detection software by two approaches. First, we investigated the issue reports posted on internet forums and bug-tracking systems for well-known object detection software. We searched the bug reports related to energy issues by keyword search. While we do not find any reports directly mentioning energy bugs, some reports implied potential causes of energy bugs in object detection systems. Therefore, in the next step, we conducted experiments to reproduce the potential energy bugs in object detection software on battery-powered edge devices and to characterize the impact of such bugs by monitoring battery consumption and resource utilization. The experimental results clearly show the potential risk of energy bugs that are related to improper image size settings and the wrong use of unquantized models on resource-constrained edge devices. We further discuss the impact of these energy bugs, their causes, and possible mitigation strategies.

The organization of this paper is as follows. Section II discusses the background and related work. Section III outlines our motivation for this study and the approaches adopted in the investigation. Section IV presents our investigation into energy bugs in object detection software using internet forums. Section V details the experimental setup, and Section VI presents a discussion on the impact, possible causes, and approaches for addressing energy bugs. Finally, Section VII offers our conclusions.

II. Background and related work

An energy bug is a software defect that causes a program to consume more power than anticipated due to design flaws, improper API usage, or inefficient interactions between hardware components [6]. Although the program may run correctly aside from the excessive power consumption, this issue makes the bug difficult to detect. Additionally, identifying the root cause is challenging because the unnecessary power consumption often results from a combination of complex factors. Based on the classification by Pathak et al. [6], energy bugs mainly categorized into four types: hardware-induced bugs, software-induced bugs, externally triggered bugs, and bugs with unknown causes.

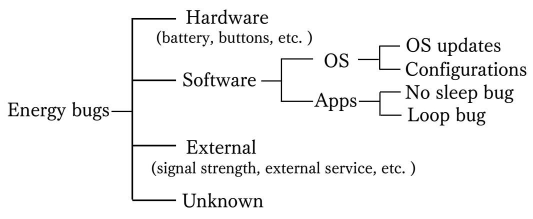

Fig. 1. The common types of energy bugs by Pathak et al. [6]

Many existing studies reported issues with energy bugs in mobile devices like smartphones. Pathak et al. [6] extracted reports related to energy bugs from internet forums and bug tracking systems using keyword searches and clustering techniques. They developed a classification method for these bugs, identifying that energy issues in mobile devices can be attributed to hardware, software, external conditions, or may remain unexplained. Notably, software bugs accounted for 35% of the cases, making them the most common. Zhang et al. [7] also investigated energy bugs in Android through internet forums, comparing them with those found in iOS and Windows. Although these studies found actual energy bugs in mobile devices, the reports did not cover potential energy bugs in object detection software, which is the focus of this study.

Detecting energy bugs is an important challenge addressed in several studies. Jiang et al. [5] proposed a static detection method for energy bugs in Android applications called SAAD (Static Application Analysis Detection). Using SAAD, they successfully detected applications with resource leaks and layout flaws with accuracies of 86.67% and 76.27%, respectively. Kim et al. [8] also presented a static analysis method for identifying wasteful processes in applications with intensive graphics operations. By providing feedback to programmers for corrections, they achieved up to a 44% reduction in energy usage. These methods are designed for detecting specific types of energy bugs in mobile devices. In contrast, this study aims to investigate energy bugs in object detection software, where the specific causes are not yet well understood.

III. Motivation and approach

While energy bugs in mobile devices are well-known and thoroughly investigated, the risks associated with energy bugs in object detection software have not received the same level of attention. Energy inefficiencies caused by software bugs in object detection can have significant consequences, particularly on battery-powered edge devices. For instance, in a scenario where real-time object detection is used on a drone for environmental monitoring, energy bugs could reduce flight endurance and compromise critical missions, such as disaster rescue operations. This highlights the need to investigate the potential risks of energy bugs in object detection software.

To better understand energy bugs in object detection software, this study explores two research questions.

1) Have energy bugs been reported in well-known object detection software?
2) How can energy bugs impact the operation of an object detection system running on a battery-powered device?

To answer the first question, we conducted reviews of online forums, such as GitHub and Stack Overflow, focusing on well-known object detection software. A similar methodology was adopted in the energy bug investigation for mobile devices [6], [7]. We follow the same approach, targeting the object detection software instead. As a result, we identified a few reports about YOLOv5's issues that might be caused by energy bugs. The results are presented in Section IV. To answer the second question, based on the identified bug reports, we conducted experiments to characterize the impact of energy bugs in an object detection system. We consider two energy bug scenarios. The first scenario reproduces an inefficient execution caused by the wrong setting for input image size. The other scenario is related to the improper use of an unquantized model on the edge device. By analyzing the decreasing trends of battery consumption, we demonstrate the significant impacts of those energy bugs in Section V.

IV. Investigation of bug reports

As the first step to investigate potential energy bugs in object detection software, first we conducted reviews of the posts on bug-tracking systems and internet forums. In this study, we choose YOLOv5 and Mask R-CNN as the target object detection software. YOLOv5 and Mask-R CNN are popular object detection algorithms for one-stage and two-stage algorithms employed in many application systems. Since their source codes are available on GitHub, software bugs in these software are actively discussed in the developer forums. From the GitHub and Stack Overflow, we searched for posts related to energy bugs using keywords, such as ”energy,” ”battery,” ”power,” ”wake,” and ”drain,” following the approach used by [6]. The results for YOLOv5 and Mask R-CNN are discussed in the following subsection A and B, respectively.

A. Bug Reports for YOLOv5

YOLO is a one-stage object detection algorithm that can perform object detection and classification simultaneously, making it fast and accurate, which has contributed to its widespread use [9]. In this study, we selected YOLOv5 for examination due to the numerous issue reports available on GitHub. Table I presents the results of our keyword searches in GitHub issue reports and Stack Overflow. Duplicate entries were excluded from the counts for each keyword.

TABLE I
SEARCH RESULTS FOR YOLOV5 BUG REPORTS

	Total	Energy	Battery	Power	Wake	Drain
GitHub	8835	3	8	63	1	2
Stack Overflow	495	0	0	3	0	0

After reviewing the reports identified, we filtered potential energy bugs using the following two criteria adopted by Zhang et al. [7]:

- The issue describes a potential cause of battery depletion.
- The issue pertains to the application rather than to hardware or the operating system.

After the filtering, we found zero reports satisfying the above criteria from the Stack Overflow. However, from the GitHub, we found one report related to the keyword “energy” and four reports related to the word ”power”. The details of these reports are summarized as follows.

- **EdgeTPU HIB error / stability (#8147)[1]:** A phenomenon has been reported where the Google Coral Dev board freezes due to a ”HIB error” after performing long inference tasks using the YOLOv5 model. The cause of this issue is attributed to some driver IO/heat. As a countermeasure, it is suggested to either reduce the model input size or use a smaller model, which is expected to reduce the device load and lower the frequency of the error occurrence.
- **Running medium or larger model creates an unkillable zombie process (#11789)[2]:** When running training with the train.py script using a medium-sized YOLOv5 model, an issue has been reported where the process does not terminate even after the training is completed and continues to consume a significant portion of GPU and CPU processing power. After training with the default image size, the number of processes does not change even when using the kill command. This issue persists even when attempting to forcibly terminate the process, leaving server reset as the only solution.
- **Data loading and preparation slow and repeated for each iteration in hyperparameters evolution (#11367)[3]:** A report indicates that when training a model with a small dataset, the speed of scanning and caching is slow, leading to lengthy load times. According to the respondent, one of the possible reasons is that hyperparameter evolution actually adds augmentation during the first iteration. As a solution, it is proposed to add the –noautoanchor and –nosave flags to the command to turn off augmentation and disable caching. However, it is not clear whether this solution effectively resolved the issue in the environment described by the reporter.
- **Consistent and efficient preprocessing for classification and detection model (#9192)[4]:** This report pointed out that the new classification model and object detection model in YOLOv5 use different preprocessing methods. This difference reduces interchangeability between models and reportedly affects performance, particularly in low-power environments. This discrepancy can potentially lead to unnecessary energy consumption when used in embedded applications or on low-power devices. It is proposed to adopt the same preprocessing used in the detection model for the classification model to lighten the preprocessing workload.
- **Why using detect.py use more power than PyTorch Hub? (#9638)[5]:** A report indicates that there is a difference in power consumption when performing video detection using the regular detect.py script versus PyTorch Hub. Users measured that using PyTorch Hub is faster and consumes less power than using detect.py. It is concluded that this difference is due to detect.py being configured to save inference results by default, while PyTorch Hub only saves results on demand.

The last bug report, where using a model without saving inference results reduced power consumption, is considered an example of a resolved energy bug. The other four reports did not explicitly mention power consumption issues, they are regarded as potential energy bug reports due to their implications for energy usage.

B. Bug Reports for Mask R-CNN

Mask R-CNN is a two-stage object detection algorithm [10]. This algorithm first identifies regions where objects might be present, and then performs object detection on each of these regions. Although slower than single-stage algorithms like YOLO, Mask R-CNN provides higher accuracy. It is capable of segmentation and is used in advanced image editing where high accuracy is required. The results of the keyword search, for the Stack Overflow and the GitHub are shown in Table II.

TABLE II
SEARCH RESULTS FOR MASK R-CNN BUG REPORTS

	Total	Energy	Battery	Power	Wake	Drain
GitHub	2726	1	1	8	1	0
Stack Overflow	262	0	0	1	0	0

We filtered the report by the same criteria adopted by Zhang et al. [7]. However, we found no reports meeting these criteria either GitHub or Stack Overflow.

[1]https://github.com/ultralytics/yolov5/issues/8147
[2]https://github.com/ultralytics/yolov5/issues/11789
[3]https://github.com/ultralytics/yolov5/issues/11367
[4]https://github.com/ultralytics/yolov5/issues/9192
[5]https://github.com/ultralytics/yolov5/issues/9638

C. *Results from Internet Forum Investigations*

Similar to the studies by Pathak et al. [6] and Zhang et al. [7], the investigation of internet forums and bug tracking systems related to object detection software identified several reports that could be regarded energy bugs, although the reports did not directly mention the causes as energy bugs. Given the number of identified issues, the risk of energy bug in object detection software does not seem to be widely recognized yet. We did not find any reports specifically related to power consumption in battery-powered edge devices. However, among the bug reports reviewed, there were more indications of potential energy bugs in YOLOv5 compared to Mask R-CNN. Consequently, we decided to conduct further experiments using YOLOv5 to explore these potential energy issues in greater detail.

V. Investigating energy bugs by experiments

We conducted experiments on a battery-powered device to examine the impact of energy bugs in object detection software. Inspired by the reports from the internet forum, we set up two different experimental configurations. The first configuration aims to replicate the issue of heavy preprocessing for object detection running on a low-powered device (Issue #9192). We omit the input image size setting, resulting in increased preprocessing for resizing the input image. The configuration may cause increased energy consumption due to unnecessary preprocessing on the edge device. The second configuration is inspired by the issue of increased power consumption due to an inappropriate model for the edge device (Issue #9638). We create a quantization model that is suitable for execution on the edge device and compare its energy efficiency with an unquantized model. Using the unquantized model on the edge device potentially consumes more energy than the system with a quantized model.

A. *Experiment Setup*

1) System Configuration: For this experiment, we simulated natural environment monitoring with a UAV, using a battery-powered Raspberry Pi for real-time object detection. To recreate a natural environment, we used the AirSim simulator to create a virtual forest populated with animals. Fig. 2 shows an image captured within this virtual environment.

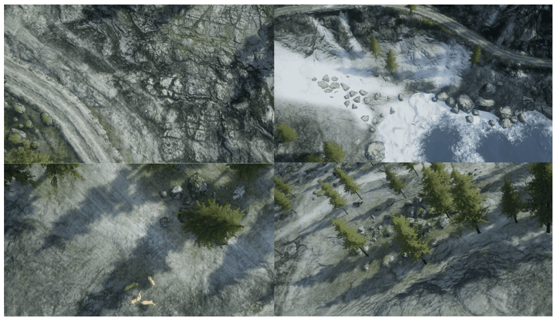

Fig. 2. Landscape of the virtual environment

A UAV in the simulator captures real-time video footage to detect individual animals, such as deer and wolves, in the forest. The generated videos are streamed using an RTMP (Real-Time Messaging Protocol) server, while the object detection software runs on the Raspberry Pi. We used YOLOv5 for object detection, which offers a lightweight model suitable for edge devices with limited computing resources, commonly used in drone systems [11]. Fig. 3 shows the system used in the experiments. The system configuration is as follows:

- Edge Device: Raspberry Pi 4B, Quad-core Cortex-A72 (ARMv8) 64-bit SoC 1.5GHz, 4GB RAM;
- OS: Debian GNU/Linux 11;
- Battery: PiSugar 3 Plus, 5000mAh;
- RTMP server/Video collection equipment: ASUS Vivo-Mini VC65, Ubuntu 18.04.6 LTS;
- Object Detection Software: YOLOv5 v7.0.

Fig. 3. The experiment system using a Raspberry Pi powered by a mobile battery

2) Measurements: We measured the battery consumption and resource utilization of the Raspberry Pi. Battery consumption data were collected every five minutes using the PiSugar 3 Plus app [12]. Additionally, resource utilization data, including CPU and memory usage, was collected every five minutes using the 'sar' command [13]. Each measurement session started with the battery fully charged at 100%. Monitoring of the Raspberry Pi's resource utilization and power consumption started at the beginning of the experiment, with YOLOv5 execution starting after 10 minutes.

3) Premeasurement of battery consumption trend: A preliminary measurement test was conducted to assess a common battery consumption trends without running object detection. The Raspberry Pi was operated in an idle state with a fully charged battery, and the battery level was measured until it depleted to 0%. The results are presented in Fig. 4. The duration of continuous operation was 432 minutes. The observed battery level (blue line) generally matched the theoretical value (dashed green line) $BL(t) = 100 - (100t/432)$, which is the expected battery level at time t given that the battery level reaches to 0 at a constant rate. The standard deviation of measurement errors, σ, was 2.3%, and the confidence interval, $BL(t) \pm \sigma$, is depicted in the green shaded area in the figure. The average of the slope of two consecutive points from the start to the end of the measurement was -0.223 [%/min].

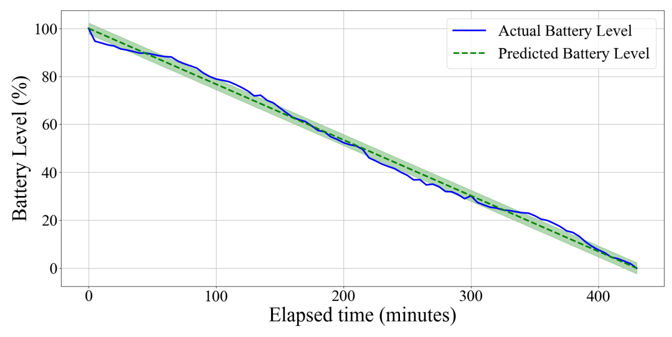

Fig. 4. Typical battery consumption trend of an idle Raspberry Pi

B. Energy bug due to faulty image settings

In YOLOv5, if an image size is not specified, the input image is automatically resized to 640x640 pixels for processing. For example, if YOLOv5 version 7.0 receives a video with a size of 160x160 without setting the image size, the OpenCV library's cv2.resize function enlarges the image to 640x640 pixels for processing. When YOLOv5 is used in an object detection system without properly setting the image size, the resizing of images leads to unnecessary processing, which can increase the power consumption of the object detection system.

1) Experiments on Energy Bugs due to Image Size Settings: To investigate whether omitting image size settings can lead to energy bugs in object detection applications, we conducted experiments comparing battery consumption trends from the systems with or without image size setting. For this experiment, we used the pretrained file yolov5s.pt provided by YOLOv5, which is available from the official YOLOv5 GitHub repository maintained by Ultralytics. The YOLOv5 program was executed ten minutes after starting measurements and continued the operation for 130 minutes. We used a video with an image size of 160x160 pixels. The measurement results were compared for YOLOv5 executions with image size settings (With Size Setting) and without the settings (Without Size Setting). Each configuration was tested in three trials.

2) Experimental Results: Fig. 5 shows the battery consumption trends for 130 minutes with and without image size settings. It can be observed that the battery level decreases rapidly in the Without Size Setting (depicted in orange lines). The results imply that an omission of size setting causes an increased amount of preprocessing. The power consumption trends of Without Size Setting significantly fluctuated, which were notably larger than the standard deviation of $\sigma = 2.3\%$ reported in the preliminary measurements described in Section V-A3. The fluctuation is likely due to variations in load and an unstable power supply. Table III shows the average slope of battery level between 15 and 130 minutes for all the trials. The results show that the slope of the battery level of With Size Setting is significantly smaller than the slope of Without Size Setting. This indicates that when the image size is not properly set, more power is consumed. Specifically, the average battery consumption trend without image size settings is -0.610 [%/min], compared to -0.532 [%/min] with image size settings.

Fig. 6 illustrates the time variation of CPU utilization for Trial 1. We observe that CPU utilization increased sharply after starting the object detection algorithm and kept the same usage level during the remaining period. The CPU utilization of Without Size Setting is consistently larger than that of With Size Setting. Table IV summarizes the average CPU utilization during the stable state, excluding outliers. The comparison of the average CPU utilization further demonstrates that improper image size settings lead to unnecessary computational processing, resulting in increased power consumption.

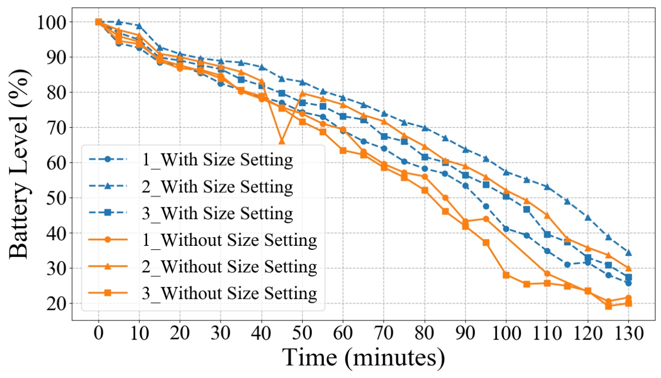

Fig. 5. Battery consumption trends observed in the systems with and without image size setting

TABLE III
COMPARISON OF BATTERY LEVEL DEPLETION RATES [%/MIN]

	Trial 1	Trial 2	Trial 3	Average
With Size Setting	-0.546	-0.507	-0.543	-0.532
Without Size Setting	-0.672	-0.554	-0.603	-0.610

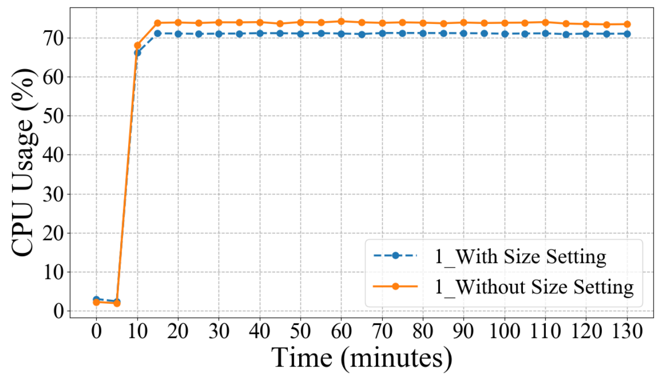

Fig. 6. CPU utilization over time observed in the systems with and without image size setting

TABLE IV
AVERAGE CPU UTILIZATION DURING STABLE STATE [%]

	Trial 1	Trial 2	Trial 3	Average
With Size Setting	71.0857	71.1491	71.0239	71.09
Without Size Setting	73.8265	73.2335	73.3596	73.47

C. Energy bugs due to inappropriate model use

Edge devices have limited computing resources. To perform object detection efficiently and minimize power consumption, it is advisable to use lightweight models. One optimization method is quantization, which is commonly applied when deploying neural networks on edge devices [14] [15]. Quantization reduces the model size by converting weights and activation functions from 32-bit floating-point numbers to 8-bit or 16-bit integers. This approach decreases memory usage, improves the inference speed, and reduces the energy consumption. Consequently, using unquantized models on resource-constrained edge devices can lead to increased computational load and higher power consumption.

1) Experiments on Energy Bugs due to inappropriate model use: To investigate if using unquantized models can cause energy bugs, we conducted experiments to compare resource utilization and power consumption in the systems using quantized and unquantized models. For this experiment, the YOLOv5 model was converted into a TensorFlow Lite format [16]. TensorFlow Lite is an open-source machine learning library optimized for mobile and embedded devices. We prepared two versions of the model: a quantized model (Quantized Model) and an unquantized model (Unquantized Model). The quantized TensorFlow Lite model uses int8, while the unquantized model uses float32 from yolov5s.pt. A 640x640 video was streamed using an RTMP server, and inference was performed with both models. Resource utilization and power consumption were collected over a period of 130 minutes. We conducted three experimental trials for each configuration.

2) Experimental Results: Fig. 7 shows the battery consumption trends for both the quantized and unquantized models over time. It can be observed that the Unquantized Model consistently consumes more battery than the Quantized Model. Table V presents the average slope of two consecutive data points between 15 and 130 minutes for all models and trials. The average slope of two consecutive points was smaller for the Quantized Model than that for the Unquantized Model. The battery depletion rate was -0.427 [%/min] for the the Quantized Model and -0.568 [%/min] for the Unquantized Model, both significantly lower than the -0.223 [%/min] observed in preliminary measurements.

Fig. 8 illustrates the CPU utilization over the 130-minute period for Trial 1. Similar to the previous experiments for faulty image setting, the CPU utilization increases sharply after starting the object detection algorithm and then it remains stable until the end of the experiment. The CPU utilization of the Unquantized Model is significantly larger than that of the Quantized Model, revealing the effectiveness of quantization. Table VI shows the average CPU utilization during the stable period. The Unquantized Model can result in excessive power consumption due to intensive CPU processing.

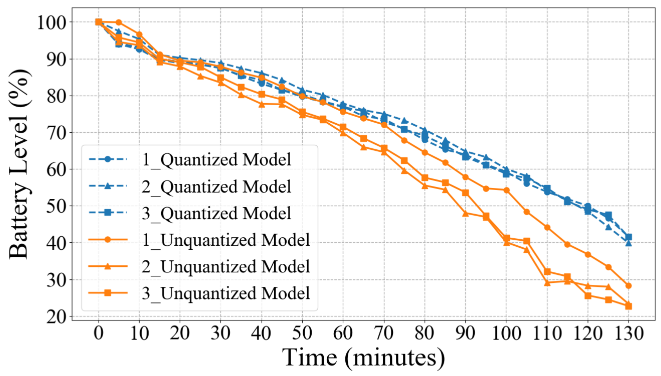

Fig. 7. Battery consumption trends observed in the systems with unquantized model and quantized model

TABLE V
COMPARISON OF BATTERY LEVEL DEPLETION RATES [%/MIN]

	Trial 1	Trial 2	Trial 3	Average
Quantized Model	-0.418	-0.445	-0.418	-0.427
Unquantized Model	-0.547	-0.572	-0.584	-0.568

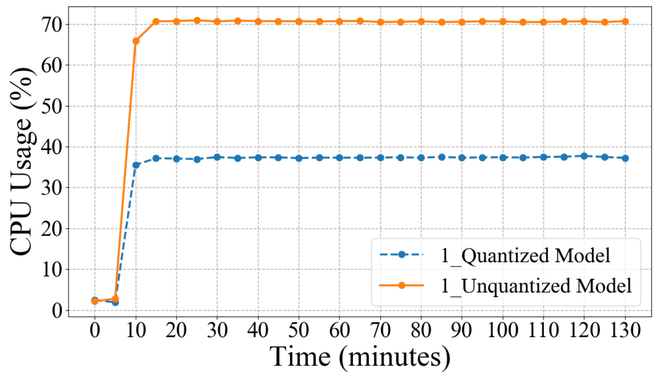

Fig. 8. CPU utilization over time observed in the systems with unquantized model and quantized model

TABLE VI
AVERAGE CPU UTILIZATION DURING STABLE STATE[%]

	Trial 1	Trial 2	Trial 3	Average
Quantized Model	37.3535	38.197	37.2126	37.59
Unquantized Model	70.6926	71.4074	71.4713	71.19

VI. Discussion

This section explores the implications of energy bugs in object detection systems, particularly those running on battery-powered devices. We discuss the potential impacts of these bugs, their causes, and possible mitigation strategies, as well as the relevance of these issues to other systems and models.

A. Impact of Energy Bugs on Object Detection Systems

This study examined the potential impacts of energy bugs in object detection algorithms on battery-powered devices. Several issues can arise from these energy bugs:

- **Operational time reduction:** Unexpected power consumption caused by energy bugs can significantly reduce the operational time of battery-powered devices. This is particularly critical in emergency scenarios, such as disaster response or rescue operations, where drones and robots are expected to operate a longer period.
- **Battery overload:** Excessive power consumption due to energy bugs increases the load on batteries, accelerating degradation and potentially causing a failure. This not only affects the reliability and lifespan of the devices but also leads to increased economic burdens due to frequent battery replacements and maintenance.
- **Increased operational costs:** The need for frequent battery replacements and maintenance driven by energy bugs lead to higher operational costs. Furthermore, poor energy efficiency increases long-term operational expenses, particularly in large-scale systems, making energy bugs a significant economic concern.

B. Causes of Energy Bugs in Object Detection systems

In addition to the energy bugs discussed several other potential causes of energy inefficiency in object detection systems should be considered:

- **Resource leak:** If resources such as memory or CPU are not properly released during the execution of an object detection task, they may continue to be consumed unnecessarily. This can lead to increased energy consumption and a shorter battery life.
- **Wasteful background processes:** If the object detection software runs unnecessary background processes, it can result in wasted energy. For example, an excessively large data cache that is not regularly cleared can cause inefficient use of memory and CPU, increasing power consumption.
- **Unnecessary Data Transfer:** Inefficient data transfers can lead to higher energy consumption. For example, frequently sending unnecessarily large data sets to the cloud or other devices can significantly increase power consumption, particularly in environments with limited bandwidth or high communication costs, where these inefficiencies can further exacerbate energy consumption.

C. Mitigation methods

To mitigate the negative effects of energy bugs in object detection algorithms, several strategies can be employed:

- **Offloading:** Offloading involves transferring computational tasks from local devices to the cloud or dedicated servers, which can reduce energy consumption and extend battery life [17]. However, it is crucial to balance this approach with considerations of network latency and communication costs.
- **Algorithm optimization:** Optimizing algorithms can help reduce energy consumption and address energy bugs [18]. Further energy savings can be achieved by fine-tuning model parameters to minimize unnecessary calculations.
- **Cache and memory management optimization:** Effective cache and memory management can lower data access costs and enhance energy efficiency. Implementing data caching strategies and optimizing memory access patterns can improve both system performance and energy efficiency.

D. Energy bugs in other systems

The energy bugs identified in YOLOv5 may also apply to other object detection systems and machine learning models. For example, while YOLOv5 is a single-stage object detection algorithm, multi-stage algorithms like Mask R-CNN and other tasks such as depth estimation and semantic segmentation may experience similar energy inefficiencies if not properly configured and optimized. Although different types of energy bugs may exist in other models, the mitigation methods discussed above are generally applicable to these issues.

VII. Conclusion

We investigated energy bugs in object detection software for battery-powered edge devices. Although no explicit mentions of energy bugs were found in internet forums, several reports suggested issues that may be equivalent to energy bugs. Therefore, we conducted experiments to measure resource utilization and battery consumption of the object detection systems with two energy bug settings. The experimental results confirmed that incorrect image size settings and the use of unsuitable models can lead to unnecessary increases in power consumption, clearly indicating that energy bugs can occur in such software. Predicting energy bugs during program development is challenging, and once deployed, they are difficult to eliminate. Therefore, developing methods to detect and avoid energy bugs is essential. Accurate estimation of battery consumption is also crucial for vital the impact of energy bugs.

Acknowledgements

This work was supported in part by a grant from the Telecommunications Advancement Foundation.

References

[1] F. Greenwood, E. L. Nelson, and P. G. Greenough, "Flying into the hurricane: A case study of UAV use in damage assessment during the 2017 hurricanes in Texas and Florida," PLoS ONE, vol. 15, 2020.

[2] A. Rejeb, A. Abdollahi, K. Rejeb, and H. Treiblmaier, "Drones in agriculture: A review and bibliometric analysis," Computers and Electronics in Agriculture, vol. 198, pp. 107017, 2022.

[3] M. S. Rajabi, P. Beigi, and S. Aghakhani, “Drone Delivery Systems and Energy Management: A Review and Future Trends,” in Handbook of Smart Energy Systems, Springer International Publishing, pp. 1273–1291, 2023.

[4] G. Tsoumplekas, V. Li, I. Siniosoglou, V. Argyriou, S. K. Goudos, I. D. Moscholios, P. Radoglou-Grammatikis, and P. Sarigiannidis, “Evaluating the Energy Efficiency of Few-Shot Learning for Object Detection in Industrial Settings,” arXiv preprint arXiv, 2024.

[5] H. Jiang, H. Yang, S. Qin, Z. Su, J. Zhang, and J. Yan, “Detecting energy bugs in android apps using static analysis,“ IEEE International Conference on Formal Engineering Methods, pp. 192–208, 2017.

[6] A. Pathak, Y. C. Hu, and M. Zhang, “Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices,“ Proceedings of the 10th ACM Workshop on Hot Topics in Networks, 2011.

[7] J. Zhang, A. Musa, and W. Le, “A comparison of energy bugs for smartphone platforms,“ 1st International Workshop on the Engineering of Mobile-Enabled Systems, pp. 25–30, 2013.

[8] C. H. P. Kim, D. Kroening, and M. Kwiatkowska, “Static program analysis for identifying energy bugs in graphics-intensive mobile apps,“ IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 115–124, 2016.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788, June 2016.

[10] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,“ IEEE International Conference on Computer Vision, pp. 2980–2988, 2017.

[11] X. Zhu, S. Lyu, X. Wang, and Q. Zhao, “TPH-YOLOv5: Improved YOLOv5 Based on Transformer Prediction Head for Object Detection on Drone-captured Scenarios,“ IEEE/CVF International Conference on Computer Vision Workshops, pp. 2778–2788, 2021.

[12] ”PiSugar Power Manager (Software) · PiSugar/PiSugar Wiki”, https://github.com/PiSugar/PiSugar/wiki/PiSugar-Power-Manager-(Software), [Accessed: September 1, 2024].

[13] ”sar(1) - Linux manual page”, https://man7.org/linux/man-pages/man1/sar.1.html, [Accessed: September 1, 2024].

[14] A. V. Demidovskij and E. Smirnov. “Effective post-training quantization of neural networks for inference on low power neural accelerator,“ International Joint Conference on Neural Networks, pp. 1–7, 2020.

[15] S. K. Macha, O. Oza, A. Escott, F. Caliva, R. M. Armitano, S K. Cheekatmalla, S. H. K. Parthasarathi, and Y. Liu. “Fixed-point quantization aware training for on-device keyword-spotting,“ IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 1–5, 2023.

[16] ”TensorFlow Lite overview — Google AI Edge — Google AI for Developers”, https://ai.google.dev/edge/lite, [Accessed: September 1, 2024].

[17] E. Andrade and F. Machida. “Assuring Autonomy of UAVs in Mission-critical Scenarios by Performability Modeling and Analysis,“ ACM Trans. Cyber-Phys. Syst., vol. 8, no. 3.

[18] H.-S. Suh, J. Meng, T. Nguyen, V. Kumar, Y. Cao, and J.-S. Seo, “Algorithm-hardware Co-optimization for Energy-efficient Drone Detection on Resource-constrained FPGA,” ACM Trans. Reconfigurable Technol. Syst., vol. 16, no. 2, pp. 1–25, 2023.