Software Aging in a Real-Time Object Detection System on an Edge Server

Kengo Watanabe
University of Tsukuba
Tsukuba, Japan
watanabe.kengo@sd.cs.tsukuba.ac.jp

Fumio Machida
University of Tsukuba
Tsukuba Japan
machida@cs.tsukuba.ac.jp

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

Roberto Pietrantuono
University of Naples Federico II
Naples, Italy
roberto.pietrantuono@unina.it

Domenico Cotroneo
University of Naples Federico II
Naples, Italy
cotroneo@unina.it

ABSTRACT

Real-time object detection systems are rapidly adopted in many edge computing systems for IoT applications. Since the computational resources on edge devices are often limited, continuous real-time object detection may suffer from the degradation of performance and reliability due to software aging. To provide a reliable IoT applications, it is crucial to understand how software aging can manifest in object detection systems under resource-constrained environment. In this paper, we investigate the software aging issue in a real-time object detection system using YOLOv5 running on a Raspberry Pi-based edge server. By performing statistical analysis on the measurement data, we detected a suspicious trend of software aging in the memory usage, which is induced by real-time object detection workloads. We also observe that a system monitoring process is halted due to the shortage of free storage space as a result of YOLOv5's resource dissipation. The monitoring process fails after 24.11, 44.56, and 115.36 hours (on average), when we set the sizes of input images to 160px, 320px, and 640px, respectively, in our system. Our experimental results can be used to plan countermeasures such as software rejuvenation and task offloading.

CCS CONCEPTS

• **Computer systems organization** → **Reliability**; • **Computing methodologies** → **Computer vision**;

KEYWORDS

Edge computing, Memory degradation, Object detection, Software aging, YOLO

ACM Reference Format:
Kengo Watanabe, Fumio Machida, Ermeson Andrade, Roberto Pietrantuono, and Domenico Cotroneo. 2023. Software Aging in a Real-Time Object Detection System on an Edge Server. In *Proceedings of ACM SAC Conference (SAC'23).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3555776.3577717

https://doi.org/10.1145/3555776.3577717

1 INTRODUCTION

Object detection is becoming an essential task in many application domains such as autonomous vehicle, video surveillance, anomaly detection, and robot visions [34, 36]. In IoT application systems, object detection functions are often deployed on edge devices considering the real-time performance and data privacy [18]. This integration is particularly important for applications requiring real-time performance, such as detection of individuals with suspicious objects in a crowd, prevention of road accidents, and equipment malfunction avoidance. The outcome of object detection systems significantly impacts on our lives and society, and hence object detection function needs to be reliable even running on resource-constrained edge devices.

As object detection functions in IoT applications typically requires continuous operation on edge server, the software aging issue comes to be a reliability concern. Software aging is known as a phenomenon that induces performance degradation and an eventual system failure after a long-running operation due to aging-related software bugs [14]. Common causes of software aging include memory leaks, memory fragmentation, and accumulated numerical errors that are often not easily detected and removed completely in the development phase [11]. Researches on software aging are broadly divided into two categories; model-based study and measurement-based study [9]. In the model-based approach, analytical models (such as stochastic reward nets (SRN) [3] and continuous-time Markov chains (CTMC) [33]) are used to capture system degradation behavior. In the measurement-based approach, which is the approach adopted in this work, system attributes (e.g., memory consumption) are periodically collected to infer signs of software aging [22].

Considering the influences of object detection systems on real-world applications, it is important to quantitatively assess the impacts of software aging on these environments. Previous research has evaluated the impact of software aging on web servers [10], operating systems [8], and cloud computing environments [24]. Software aging issues have also been reported to occur when image processing is performed in an edge computing environment [2]. A recent study reports that object detection algorithms can suffer from software aging in different libraries, implementations, and datasets [23]. However, to the best of our knowledge, there is no

existing work investigating software aging issues in a real-time object detection system running on an edge server with scarce computing resources.

In this paper, we experimentally evaluate the software aging phenomena in a real time object detection system continuously running on edge computing environments. The edge computing environment consists of two main components: a Real-Time Streaming Protocol (RTSP) server and a processing node. The RTSP server is responsible for sending the videos captured by the connected camera to other processing nodes using RTSP distribution. The processing nodes receive the RTSP video and perform real-time object detection using YOLOv5 [17]. YOLO (You Only Look Once) is the most popular object detection method based on deep learning with high speed and accuracy, and it is also used for real-time object detection [5, 17, 25–27]. In our experiment, YOLOv5 [17] was used as the object detection method. More specifically, YOLOv5s.pt is used as the trained model for object detection, as it is the lightest and fastest available. To investigate the phenomenon of software aging, we conducted long-running experiments to collect data during continuous operation of a real-time object detection system using YOLOv5 for 10 days on edge servers under different workload configurations. By using Mann-Kendall test [20] with Sen's slope estimator [29], we confirmed decreasing trends in the free memory and increasing trends in the swap usage under the workload with a larger image size. Besides the aging trend analysis, we observe the 'side-impact' of software aging that may cause another process running on the same server to fail. YOLOv5 is configured by default to save images after inferences (i.e., object detection), and hence the storage space is used up when the process runs continuously for long time. While the YOLOv5 process can survive even encountering excessive storage usage, other processes are affected by lack of storage space. In our testbed, a monitoring process that periodically appends log entries is halted due to this side-impact. We consider such a side-impact event as a failure event and measure the time to failure data under different workload conditions. The results reveal that failures occur on average after 24 hours, when the image size is 160px.

The contributions of the paper are summarized as below:

- We detect suspicious software aging trends in memory and swap usage on edge servers running a YOLOv5 process.
- We show that the software aging in YOLOv5 is highly likely caused by the image saving option that continuously save the images after inference.
- We identify an adverse side-impact of software aging that causes a system monitoring process to fail. We present the time-to-failure data of the monitoring process under different workload configurations.

The rest of the paper is organized as follows. Section 2 presents basic concepts. Section 3 describes the related work. Section 4 explains the experimental plan. Section 5 presents the results with some statistical analysis. Finally, Section 6 presents our conclusion and future works.

2 BACKGROUND

2.1 Object detection

Object detection is one category of image processing tasks that involves identifying and locating objects (e.g.: humans, animals, cars, or buildings) in an image or video. Object detection algorithms help systems to accurately and precisely identify and locate objects, just like humans. Object detection can be divided into two categories: two-stage detection and one-stage detection [16]. Two-stage detection is more accurate, but requires long time for image processing. On the other hand, one-stage detection is time-efficient and can detect objects in real time [16]. There are many one-stage detection algorithms such as YOLO, RetinaNet and Single Shot Multibox (SSD). Among others, YOLO is the most popular one-stage object detection method based on deep learning with high speed and high accuracy. Therefore YOLO is extensively used for real-time object detection tasks [5, 17, 25–27] in various domains such as automated driving [28] and drone systems [30].

2.2 Software aging

The software aging is the phenomenon causing performance degradation and an eventual system failure after a long-running operation due to aging-related software bugs [14]. There are two potential causes of the aging phenomena [31]. First, the accumulation of errors due to activation of aging related bugs within the running system (e.g.: like memory leaks or round-off errors) that can lead to performance degradation or even unplanned outages. Second, the activation and/or propagation of system errors that can influence the total system uptime, such as the incident with the Patriot missile defense system [12]. Fundamental concepts of software aging are also found in [11, 31].

Aging effects are the result of the accumulation of errors, which can be detected through aging indicators. Aging indicators refer to system variables that can be measured directly and can be related to software aging phenomena. Examples of aging indicators are service response times, memory consumption and swap usages. Aging indicators can be used to determine the existence of the software aging phenomenon. There are three approaches to estimating this phenomenon: the threshold-based approach, the statistical-based approach, and the machine learning-based approach [22]. In the threshold-based approach, a threshold is determined so that actions are taken when the aging indicator exceeds the pre-determined threshold value. In the statistical-based approach, statistical techniques are used to analyzed the data collected to assess the presence of aging phenomena. In the machine learning-based approach, time series approaches are used to predict when the system is most likely to fail due to the aging effects.

3 RELATED WORK

Software rejuvenation [14] and software life-extension [19] have been known as effective countermeasures for software aging. In order to apply such countermeasures effectively, it is imperative to understand how software aging manifest in a specific execution environment. Statistical techniques, which is the approach adopted in this work, are commonly applied for characterizing software aging from system measurement data [22]. The Mann-kendall test

[20] and the Sen's slope estimator [29] are often used to analyze the trend in the time series of aging indicator. Some alternatives have been proposed for trend detection and estimation (e.g., a modified version of the Cox-Stuart test for trend detection followed by the iterative Hodrick-Prescott Filter for (linear and non-linear) trend estimation [35], or a metric based on Multidimensional Multi-scale Entropy (MMSE) for trend analysis and aging-related failure prediction [7], but the Mann-Kendall test plus Sen's estimator procedure remains largely the most adopted one, mainly because of its simplicity and widespread use in numerous research fields. In contrast to many experimental studies on software aging that analyze the trends in the measurement data [1, 8, 10, 24], our work also attempts to collect the time-to-failure data caused by software aging, since it is also important to model and predict the aging behavior.

We also target a context little investigated so far in the software aging literature, namely object detection algorithms. YOLO-based system [5, 17, 25–27], which can perform object detection from real-time video stream, have been used in various situations such as edge computing [13]. However, because of the limited computational resources of edge computing environments, real-time object detection programs like YOLO potentially confront software aging phenomena. Therefore, this paper investigates software aging phenomenon in real-time object detection using YOLOv5 through experiments on our edge computing testbed.

Researchers in the Computer Vision area have analyzed several Deep Neural Networks (DNNs), including algorithms for object detection, from a performance perspective [4, 6], even on Raspberry Pi devices [32], and on mobile robotics [21]. These studies consider metrics that are also of interest for software aging, including power consumption, inference time, and memory. However, none of them is focused on software aging, namely what happens to performance and resource consumption metrics in the long running, which is the main goal of this work.

4 EXPERIMENTAL PLAN

The objective of our study is to analyze software aging issues in a real-time object detection system using YOLOv5 implemented on an edge server. As the edge server has limited resources that may not cope with heavy workloads, our goal is to collect aging-related statistics such as aging indicators and time-to-failure data to find suspicious aging trends and their consequences. We set up our own experimental platform consisting of edge servers executing YOLOv5 for objective detection in response to the video stream transmitted from a streaming server. We apply a measurement-based strategy using different workloads to characterize software aging phenomena. The experiment details are explained below.

4.1 Setup

We built a system that acquires images from an edge server equipped with a camera and performs object detection processing on the acquired images. The system are implemented on Raspberry Pi 4 model B that is connected with a 1000BASE-T local area network. We use the Raspberry Pi 4 model B with 1.5GHz quad core ARMv8 CPU, 4GB RAM, and 64bit Raspberry Pi OS. The system consists of two types of nodes: an RTSP server node that acquires a video using a camera and distributes the video, and a processing node

Figure 1: Real-time object detection in a video stream

that receives the video via RTSP and performs object detection using YOLOv5. The camera was used to capture a screen showing a video of a cityscape, and object detection was performed on the captured video. Figure 1 shows a screenshot of object detection in the captured video. To detect objects such as cars and people, we installed the pre-trained file "yolov5s.pt", which is a publicly available trained file for YOLOv5. Additionally, we used 12 Raspberry Pi servers; one is for an RTSP server and 11 are for processing nodes running in parallel. Each processing node executes independently and does not have any interactions except the RTSP server.

4.2 Experimental procedure

4.2.1 Aging trend analysis. The first experiments are conducted to analyze the aging trend in system metrics during continuous real-time object detection by YOLOv5 for 10 days. We choose free memory usage and swap usage as the aging indicators to be investigated, since they are common metrics for resource consumption [8, 10, 22, 24]. These metrics are collected using *sysstat* and *smem* at 10-minute intervals. We consider workload impacts by defining different input image sizes that can be controlled with YOLOv5's "–img size" option. In the experiment, we use four categories of workload: no workload, 160px, 320px, and 640px.

For trend analysis in the measurement data, we use the Python module pyMannkendall [15] to compute the Mann-Kendall statistics and to obtain the Sen's slope estimates. The Mann-Kendall analysis tests the null hypothesis (H0) that there is no trend in the time series data, while the alternative hypothesis (H1) indicates an upward or a downward trend in the data. If the p-value of the test is lower than the significance level ($p = 0.05$), then there is statistically significant evidence that a trend is present in the time series data. The Sen's slope estimator assesses the magnitude of the trend. It is computed as the median of all pairwise slopes between each pair of points in the data set so that a positive Sen's slope implies a positive trend, while a negative Sen's slope means a negative trend.

YOLOv5 runs with the image saving mode by default, in which image data is saved to local storage after each object detection. This save mode can be disabled by setting "True" to the option "–nosave", which enforces the process to save only at the final checkpoint.

4.2.2 Time to failure analysis. The second experiment is conducted to collect the time to failure values by running YOLOv5 with the image saving mode. When we run YOLOv5 with image saving mode, images are saved after every object detection, resulting in a shortage of storage space on the edge server that has only 32GB of SD card. Interestingly, YOLOv5 can continue operation even encountering the shortage of storage space. However, it has side-impacts on other running processes requiring write access to the storage. In our experimental system, the system monitoring process is forced to stop due to this side-impact. We consider this phenomenon as a type of failure caused by software aging because the resource depletion event is caused by software aging in another process under high workload. We measure the time to failure data in different workload settings in terms of input data image size (i.e., 160px, 320px, and 640px). As the processing time of each image must depend on the image size, we also measure the image processing times in different workload settings.

5 RESULTS AND ANALYSIS

5.1 Aging trend analysis

5.1.1 Image saving mode. For real-time object detection system using YOLOv5 with image saving mode, we observed symptoms of software aging in both free memory and memory swap usages. Figure 2 plots the amounts of free memory under different workload settings, and Figure 3 plots the free memory in no workload condition. In each figure, the x-axis represents the elapsed time and the y-axis represents the amount of free memory. In Figure 2, for the workloads with 160px and 320px image sizes, the amount of free memory gradually increases after the steep decrease at the beginning. On the other hand, for the workload with 640px image size, the amount of free memory decreases gradually even after the steep decrease at the beginning. The difference implies that the workload with 640px is more memory-intensive than the other workload scenarios. The free memory in no workload case exhibits a decreasing trend of free memory, as shown in Figure 3. However, the amount of free memory is one order of magnitude larger than in other cases where real-time object detection is running.

To confirm the trend in the observed free memory, we conducted the statistical analysis mentioned in Section 4.2.1. Besides, to compare the significance of trends, we applied Sen's slope estimator. Table 1 shows the results of the Mann-Kendall test and Sen's slope estimator. For the Mann-Kendall test, most p-values are less than 0.05. This means that the null hypothesis of the Mann-Kendall test is rejected. For the workload with 640px image size, both the Mann-Kendall test and Sen's slope estimator confirm the decreasing trend in the amount of free memory. It should be noted that this phenomenon can be more severe than the slope estimate appears to be. In fact, the trend is not linear and after a few hours the available memory is close to zero.

As memory usage dynamics are closely related to swap usage, we also looked at swap usage trends. Figure 4 plots the memory swap usages of some representative workload cases. The x-axis represents the elapsed time and the y-axis represents the memory swap usage. As it can be seen, swap usages increase sharply after 200 hours in both results with workloads with an image size of 640px. The results clearly show that memory tends to deplete on

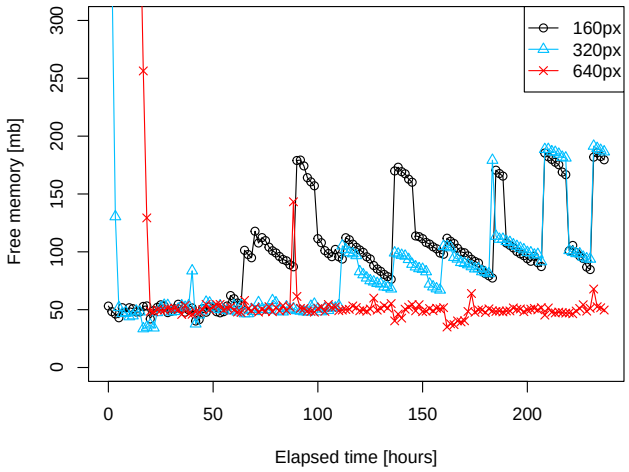

Figure 2: Trends of free memory with different image sizes under the image saving mode

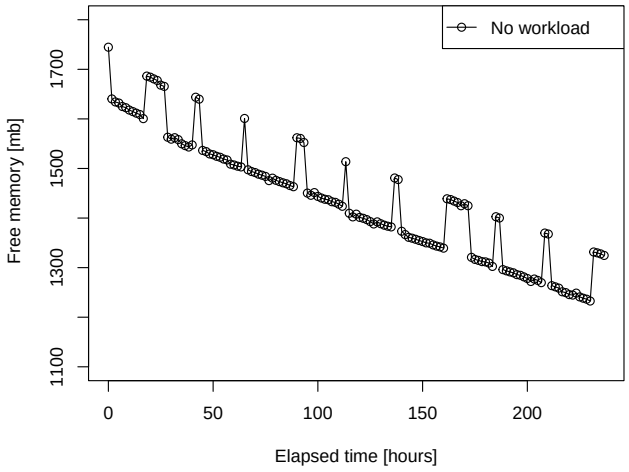

Figure 3: Trend of free memory in no workload scenario

these workloads. On the other hand, for the workloads with 160px and 320px image sizes, the swap usage is less than 5% over the experimental period.

Table 2 shows the results of the Mann-Kendall test and Sen's slope estimator. The Mann-Kendall test and Sen's slope estimator results also confirm increasing trends in all the cases except the no workload case. Some of the results show increasing trends, but with a slope of 0, meaning that the trend exists but is not significant. The results suggest that software aging is likely to occur in real-time object detection systems using YOLOv5 when the system runs for long periods of time. In particular, under the 640px image size setting, all the test results show positive slope estimates in swap usage. Note that for this case the system operates for a long time on low memory and after about 200 hours it starts using swap massively.

Table 1: MKT and slope estimates for free memory under the image saving mode

Workload	MKT p-value	Trend	Sen's estimator Slope [kb/10mins]
None	0.0	decreasing	−283.882
160px(1)	0.0	increasing	49.364
160px(2)	0.0	increasing	38.465
320px(1)	0.0	increasing	56.205
320px(2)	0.0	increasing	61.489
320px(3)	0.0	increasing	73.667
320px(4)	0.0	increasing	59.241
640px(1)	0.077	no trend	−1.091
640px(2)	0.0	decreasing	−2.290
640px(3)	0.0	decreasing	−2.692
640px(4)	0.0	decreasing	−3.996

Table 2: MKT and slope estimates for swap usage under the image saving mode

Workload	MKT p-value	Trend	Sen's estimator Slope [kb/10mins]
None	1.0	no trend	0.0
160px(1)	0.0	increasing	0.0
160px(2)	0.0	increasing	0.0
320px(1)	0.0	increasing	0.0
320px(2)	0.0	increasing	0.0
320px(3)	0.0	increasing	60.41364
320px(4)	0.0	increasing	167.92944
640px(1)	0.0	increasing	483.30912
640px(2)	0.0	increasing	429.03924
640px(3)	0.0	increasing	337.9068
640px(4)	0.0	increasing	269.30148

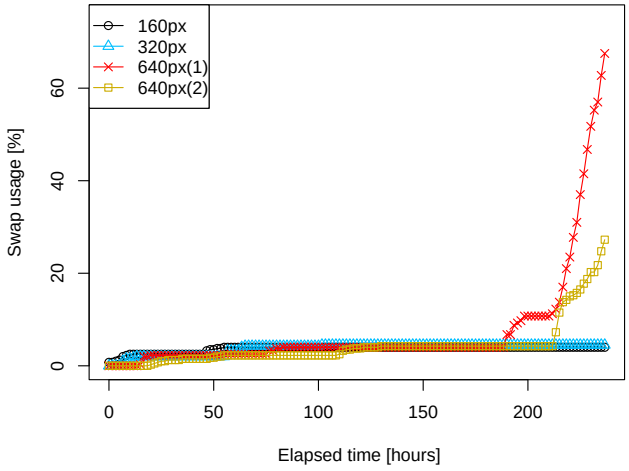

Figure 4: Trends of swap usages with different image sizes under the image saving mode

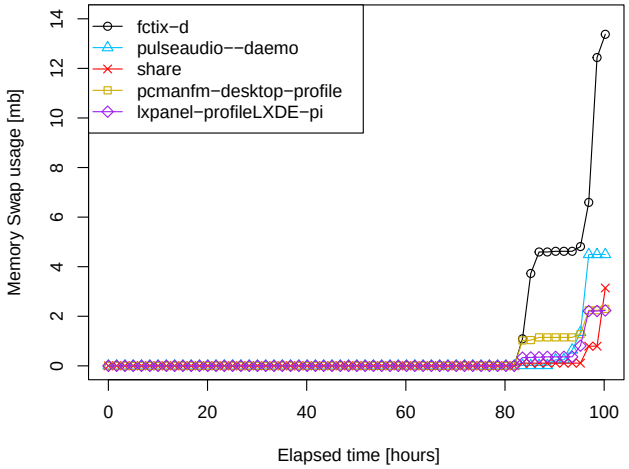

Figure 5: Trends of swap usages for the top 5 processes

To examine the root causes of the increase in swap usage, we conducted additional experiment to collect per process memory information and performed a process analysis to identify the processes that are using the most memory and swap on the edge server. We use *smem* to collect the memory and swap usage information per process at 10 minute intervals. Table 3 shows the 10 processes that consume the most swap. The most swap-consuming process is "fcitx-d" which is a lightweight input method framework providing language support independent of the Linux environment. The process is not directly involved in object detection.

Figure 5 shows the swap usage for the top five processes presented in Table 3. We can observe that the swap usages of these processes increase significantly after 80 hours. When the operating system starts swapping, the choice of the process whose pages are swapped out depends on several factors considered by the page replacement algorithm, but in general the most inactive pages are those gradually moved into the swap space. Thus, the least used processes during the experiment (e.g.: "fcitx-d") are probably the ones to be swapped out. It should be noted that this experiment focuses on analyzing the root causes of software aging presented earlier and are conducted separately from the first set of experiments. Therefore, the start of swap consumption does not coincide with the results shown in Figure 4.

We also investigated the memory usage per process. We found that "detect.py" was the most memory-consuming process, which is the main process of YOLOv5. The memory usage trend of "detect.py" is plotted in Figure 6. The memory usage gradually decreases between 80 and 100 hours. Since this interval corresponds to the same range at which swap usage increases, this event implies YOLOv5 causes swap to be consumed.

5.1.2 No image saving mode. We conducted the same experiments on YOLOv5 with no image saving mode. Figure 7 plots the amount of free memory under different workload settings. The x-axis represents the elapsed time and the y-axis represents the amount of free memory. Unlike Figure 2, the y-axis takes values between 1500mb

Table 3: The top 10 processes consuming high swap usages

Process	Swap usage [kb]	Description
fcitx-d	13940	Linux environment-independent input method framework
pulseaudio–daemon	4600	A general purpose sound server intended to act as middleware
python3/usr/share	3216	Contains architecture-independent data of Python3
pcmanfm–desktop–profile	2352	The standard file manager for Lightweight X11 Desktop Environment (LXDE)
lxpanel–profileLXDE-pi	2288	The task bar file manager for LXDE
-bash	1244	Default shell for Linux
pipewire	1156	A multimedia framework that handles audio and video on Linux
pipewire-media-session	1056	A simple session manager for PipeWire framework
systemd–user	1028	User instance
mozc/mozc_server	668	Japanese input method editor used by fcitx

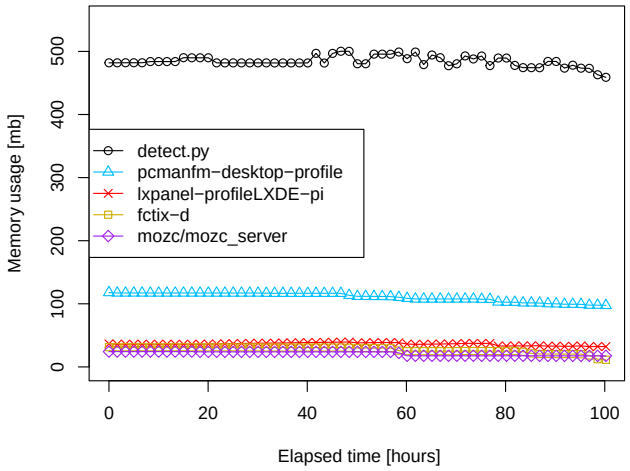

Figure 6: Trends of memory usage for the top 5 processes

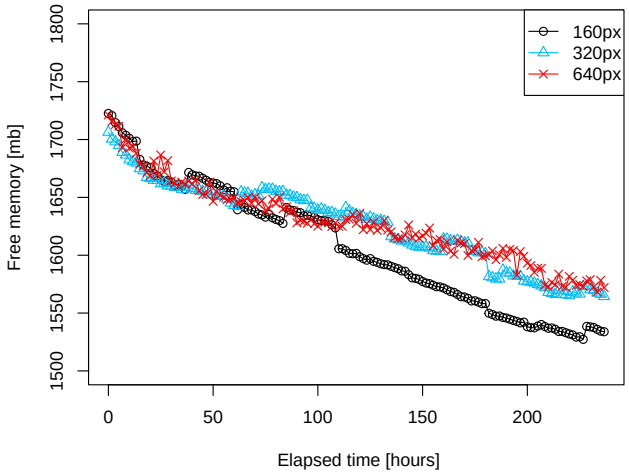

Figure 7: Trends of free memory with different image sizes under no image saving mode

and 1800mb. The amount of free memory is decreasing for all workloads, but unlike the image saving mode, the decrease is smooth and to the same extent as the *no workload* case of the saving-mode experiment.

Table 4 shows the results of the Mann-Kendall test and Sen's slope estimator. The results also confirm a decrease in the amount of free memory under all image sizes. For swap, we did not observe swap usage under all workload conditions, thus we omit to show the plots. Therefore, memory-related software aging trend is small in the no image saving mode. These results suggest that the free memory reduction and memory swap caused by YOLOv5 is highly likely caused by the default image saving option.

5.2 Time to failure analysis

As explained in Section 4.2.2, a system monitoring process is halted when we run YOLOv5 with image saving mode. The failure is caused by the shortage of storage space. In this experiments, we collected the time-to-failure data under different workload settings. The obtained results are shown in Table 5. The results show that the monitoring process fails faster when YOLOv5 runs with the smaller image size. Failures were observed between approximately

Table 4: MKT and slope estimates for free memory under no image saving mode

Workload	MKT p-value	Trend	Sen's estimator Slope [kb/10mins]
160px(1)	0.0	decreasing	−146.425
160px(2)	0.0	decreasing	−127.952
160px(3)	0.0	decreasing	−127.952
320px(1)	0.0	decreasing	−106.653
320px(2)	0.0	decreasing	−83.461
320px(3)	0.0	decreasing	−92.863
320px(4)	0.0	decreasing	−87.980
640px(1)	0.0	decreasing	−79.172
640px(2)	0.0	decreasing	−72.496
640px(3)	0.0	decreasing	−72.496
640px(4)	0.0	decreasing	−90.047

22.5 and 26.5 hours for 160px images, which are about half of the case with 320px and a quarter of the case with 640px. The results reveal that the time to failure is highly related to the image size.

Table 5: Time-to-failures observed in different image sizes

Workload	Time-to-failure [hour]	Average
160px(1)	22.49574	
160px(2)	23.22652	
160px(3)	26.59628	24.10618
320px(1)	43.35192	
320px(2)	45.59534	
320px(3)	44.86154	
320px(4)	44.25723	44.55833
640px(1)	120.2252	
640px(2)	101.0385	
640px(3)	102.3448	
640px(4)	137.8333	115.36045

The more severe aging side-impact for the 160px case than for the 320px and 640px cases (unlike the previous experiment) can be explained by considering that the main contribution to memory consumption is likely more related to processing. That is, with smaller images, YOLOv5 can process more images in the same time unit, and this impacts more on memory consumption. To corroborate this hypothesis, we also measured the processing time of each image under different workload settings. The average processing times per an image are 0.294, 0.716, and 2.207 seconds, respectively, for input image sizes of 160px, 320px, and 640px. The larger the image size to be inferred, the more time it takes to detect objects per image, and the longer it takes to put pressure on storage. The processing time per image is faster in 160px case, which increases the number of images to be stored and puts pressure on storage.

We also conducted the Mann-Kendall test and Sen's slope estimator to investigate the effect of aging on processing time. Table 6 shows a slight trend toward a decrease in processing time in most cases. That is, the results showed no negative effects of software aging on processing time. We also computed throughput per workloads at 10 minute intervals and conduct data trend analysis. The throughput per workload are 1566.519, 744.622, and 257.386 [images/10mins], when we set the sizes of the input images to 160px, 320px, and 640px, respectively. Likewise the processing time, throughput in the 160px case is the highest. Table 7 shows that the only case of a decreasing trend in the throughput is the 160px(3) case. As this is the only decreasing case, we present more details about it. Figure 8 shows the throughput over time for this case. The results show that throughput significantly decreased around 8 hours. However, after 14 hours, the throughput increases significantly, followed by a gradual decreasing trend, which is potentially caused by software aging. After that, the process is halted around 26 hours, as presented in Table 5.

Table 6: MKT and slope estimates for the processing time

Workload	MKT p-value	Trend	Sen's estimator Slope [secs/10images]
160px(1)	0.795	no trend	0.0
160px(2)	0.001	decreasing	$-1.289e^{-7}$
160px(3)	0.135	no trend	$-7.867e^{-8}$
320px(1)	0.0	decreasing	$-5.303e^{-7}$
320px(2)	0.648	no trend	0.0
320px(3)	0.0	decreasing	$-3.863e^{-6}$
320px(4)	$3.775e^{-15}$	decreasing	$-1.216e^{-6}$
640px(1)	0.0	decreasing	$-3.933e^{-5}$
640px(2)	0.0	decreasing	$-3.814e^{-6}$
640px(3)	0.0	decreasing	$-3.557e^{-6}$
640px(4)	0.0	decreasing	$-5.846e^{-5}$

Table 7: MKT and slope estimates for the throughput

Workload	MKT p-value	Trend	Sen's estimator Slope [images/10mins]
160px(1)	0.281	no trend	0.12174
160px(2)	0.988	no trend	0.0
160px(3)	0.0458	decreasing	−0.15152
320px(1)	$6.666e^{-06}$	increasing	0.089917
320px(2)	0.825	no trend	0.0
320px(3)	0.002	increasing	0.09615
320px(4)	0.37	no trend	0.02109
640px(1)	0.492	no trend	0.00396
640px(2)	0.0004	increasing	0.0084
640px(3)	0.465	no trend	0.0
640px(4)	$5.354e^{-10}$	increasing	0.07971

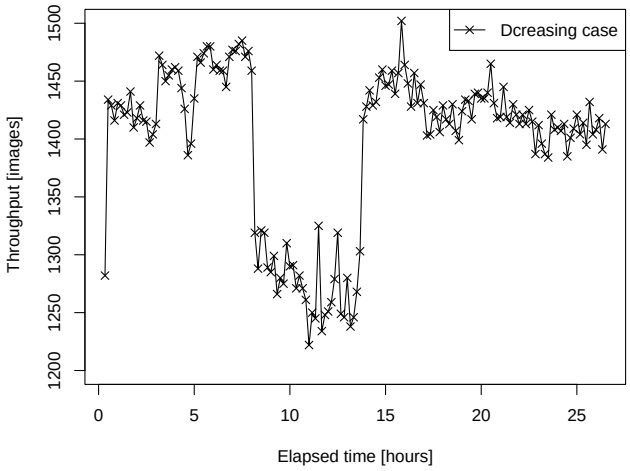

Figure 8: The observed decreasing trend of throughput

6 CONCLUSION

In this study, we observed experimentally suspicious software aging trends in edge computing environments for real time object detection using YOLOv5. The decrease in the amount of free memory, and the increase in the memory swap usage are observed when we set image size to 640px in the image saving mode. However, we did not observe such trend in the no image saving mode. This suggests that conventional software aging problems are likely to occur, especially under the workload with 640px in the image saving mode. We also identified software aging in storage space when YOLOv5

runs with the image saving mode. Although the shortage of storage space did not impact the operation of YOLOv5, it did impact other system process that was halted when data could not be written to storage. Consequently, we collected the time-to-failure of the monitoring process after starting YOLOv5. The results showed that the failures occur in a shorter time for the smallest input image sizes. This may be due to the fact that the low size image is processed faster than larger size images, which increases the amount of images to be stored. In conclusion, real-time object detection with YOLOv5 in an edge computing environment is likely subject to software aging problems, although the degree varies depending on the workload. To be confirmed, however, additional experiments are required. As with most of the software aging literature, our tests are run under a fixed workload in order to accelerate the aging process. Still, the aging behavior under a variable workload intensity should be investigated.

In the future, we plan to work along three directions: *i)* conducting similar experiments with different object detection algorithms and different versions of YOLO to see if the same phenomenon occurs, *ii)* exploiting measurement data to setup a model-based analysis, where we can model the consumption of the involved resources and explore alternative architectural solutions, and *iii)* considering proactive countermeasures such as periodic restart for clearing aging states (i.e., software rejuvenation) or offloading tasks to other computation nodes to perform software life-extension [33].

ACKNOWLEDGEMENTS

This work was supported in part by the grant of University of Tsukuba Basic Research Support Program Type S. This research was also partially supported by Japan Society for the Promotion of Science (JSPS) Invitational Fellowships for Research in Japan.

REFERENCES

[1] Ermeson Andrade, Fumio Machida, Roberto Pietrantuono, and Domenico Cotroneo. 2020. Software Aging in Image Classification Systems on Cloud and Edge. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. 342–348. https://doi.org/10.1109/ISSREW51248.2020.00099

[2] Ermeson Andrade, Roberto Pietrantuono, Fumio Machida, and Domenico Cotroneo. 2021. A Comparative Analysis of Software Aging in Image Classifiers on Cloud and Edge. *IEEE Transactions on Dependable and Secure Computing* (2021).

[3] Ermeson C Andrade, Fumio Machida, Dong-Seong Kim, and Kishor S Trivedi. 2011. Modeling and analyzing server system with rejuvenation through sysml and stochastic reward nets. In *2011 Sixth International Conference on Availability, Reliability and Security*. IEEE, 161–168.

[4] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. 2018. Benchmark analysis of representative deep neural network architectures. *IEEE access* 6 (2018), 64270–64277.

[5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934* (2020).

[6] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. 2016. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678* (2016).

[7] Pengfei Chen, Yong Qi, Xinyi Li, Di Hou, and Michael Rung-Tsong Lyu. 2016. ARF-predictor: Effective prediction of aging-related failure using entropy. *IEEE Transactions on Dependable and Secure Computing* 15, 4 (2016), 675–693.

[8] Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo. 2010. Software aging analysis of the linux operating system. In *2010 IEEE 21st International Symposium on Software Reliability Engineering*. IEEE, 71–80.

[9] Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo. 2014. A survey of software aging and rejuvenation studies. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 10, 1 (2014), 1–34.

[10] Michael Grottke, Lei Li, Kalyanaraman Vaidyanathan, and Kishor S Trivedi. 2006. Analysis of software aging in a web server. *IEEE Transactions on reliability* 55, 3 (2006), 411–420.

[11] Michael Grottke, Rivalino Matias, and Kishor S Trivedi. 2008. The fundamentals of software aging. In *2008 IEEE International conference on software reliability engineering workshops (ISSRE Wksp)*. Ieee, 1–6.

[12] Michael Grottke and Kishor S Trivedi. 2007. Fighting bugs: Remove, retry, replicate, and rejuvenate. *Computer* 40, 2 (2007), 107–109.

[13] Byung-Gil Han, Joon-Goo Lee, Kil-Taek Lim, and Doo-Hyun Choi. 2020. Design of a scalable and fast yolo for edge-computing devices. *Sensors* 20, 23 (2020), 6779.

[14] Yennun Huang, Chandra Kintala, Nick Kolettis, and N Dudley Fulton. 1995. Software rejuvenation: Analysis, module and applications. In *Twenty-fifth international symposium on fault-tolerant computing. Digest of papers*. IEEE, 381–390.

[15] Md Hussain and Ishtiak Mahmud. 2019. pyMannKendall: a python package for non parametric Mann Kendall family of trend tests. *Journal of Open Source Software* 4, 39 (2019), 1556.

[16] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. 2019. A survey of deep learning-based object detection. *IEEE access* 7 (2019), 128837–128868.

[17] Glenn Jocher, A Chaurasia, A Stoken, J Borovec, Y Kwon, and othres. 2022. YOLOv5. https://github.com/ultralytics/yolov5

[18] Gaolei Li, Kaoru Ota, Mianxiong Dong, Jun Wu, and Jianhua Li. 2019. DeSVig: Decentralized swift vigilance against adversarial attacks in industrial artificial intelligence systems. *IEEE Transactions on Industrial Informatics* 16, 5 (2019), 3267–3277.

[19] Fumio Machida, Jianwen Xiang, Kumiko Tadano, and Yoshiharu Maeno. 2012. Software life-extension: a new countermeasure to software aging. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering*. IEEE, 131–140.

[20] Henry B Mann. 1945. Nonparametric tests against trend. *Econometrica: Journal of the econometric society* (1945), 245–259.

[21] Ricardo Pereira, Tiago Barros, Luís Garrote, Ana Lopes, and Urbano J Nunes. 2020. An Experimental Study of the Accuracy vs Inference Speed of RGB-D Object Recognition in Mobile Robotics. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 588–595.

[22] Roberto Pietrantuono, Javier Alonso, and Kalyan Vaidyanathan. 2020. Measurements for Software Aging. *Handbook of Software Aging and Rejuvenation, Chapter 4* (2020).

[23] Roberto Pietrantuono, Domenico Cotroneo, Ermeson Andrade, and Fumio Machida. 2022. An Empirical Study on Software Aging of Long-Running Object Detection Algorithms. In *2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security*.

[24] Roberto Pietrantuono and Stefano Russo. 2020. A survey on software aging and rejuvenation in the cloud. *Software Quality Journal* 28, 1 (2020), 7–38.

[25] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.

[26] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7263–7271.

[27] Joseph Redmon and Ali Farhadi. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).

[28] Abhishek Sarda, Shubhra Dixit, and Anupama Bhan. 2021. Object Detection for Autonomous Driving using YOLO algorithm. In *2021 2nd International Conference on Intelligent Engineering and Management (ICIEM)*. IEEE, 447–451.

[29] Pranab Kumar Sen. 1968. Estimates of the regression coefficient based on Kendall's tau. *Journal of the American statistical association* 63, 324 (1968), 1379–1389.

[30] Li Tan, Xinyue Lv, Xiaofeng Lian, and Ge Wang. 2021. YOLOv4_Drone: UAV image target detection based on an improved YOLOv4 algorithm. *Computers & Electrical Engineering* 93 (2021), 107261.

[31] Kishor S. Trivedi, Michael Grottke, and Ermeson Carneiro de Andrade. 2010. Software fault mitigation and availability assurance techniques. *International Journal of System Assurance Engineering and Management* 1 (2010), 340–350.

[32] Delia Velasco-Montero, Jorge Fernández-Berni, Ricardo Carmona-Galán, and Ángel Rodríguez-Vázquez. 2018. Performance analysis of real-time DNN inference on Raspberry Pi. In *Real-Time Image and Video Processing 2018*, Vol. 10670. SPIE, 115–123.

[33] Kengo Watanabe and Fumio Machida. 2022. Availability Analysis of a Drone System with Proactive Offloading for Software Life-extension. In *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE, 1–6.

[34] Yanzhao Wu, Ling Liu, and Ramana Kompella. 2021. Parallel Detection for Efficient Video Analytics at the Edge. In *2021 IEEE Third International Conference on Cognitive Machine Intelligence (CogMI)*. IEEE, 01–10.

[35] Pengfei Zheng, Yong Qi, Yangfan Zhou, Pengfei Chen, Jianfeng Zhan, and Michael Rung-Tsong Lyu. 2014. An Automatic Framework for Detecting and Characterizing Performance Degradation of Software Systems. *IEEE Transactions on Reliability* 63, 4 (2014), 927–943. https://doi.org/10.1109/TR.2014.2338255

[36] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. 2019. Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055* (2019).