Proceedings of the ASME 2022
International Design Engineering Technical Conferences and
Computers and Information in Engineering Conference
IDETC/CIE2022
August 14-17, 2022, St. Louis, Missouri

DETC2022-88459

METAREASONING APPROACHES FOR THERMAL MANAGEMENT DURING IMAGE PROCESSING

Michael K. Dawson Jr.
Jeffrey W. Herrmann
Department of Mechanical Engineering
University of Maryland
College Park, Maryland 20742

ABSTRACT

Resource-constrained electronic systems are present in many semi- and fully-autonomous systems and are tasked with computationally heavy tasks such as neural network image processing. Without sufficient cooling, these tasks often increase device temperature up to a predetermined maximum, beyond which the task is slowed by the device firmware to maintain the maximum. This is done to avoid decreased processor lifespan due to thermal fatigue or catastrophic processor failure due to thermal overstress. This paper describes a study that evaluated how well metareasoning can manage a Raspberry Pi 4B's central processing unit (CPU) temperature while it is performing image processing (object detection and classification) on the Common Objects in Context (COCO) dataset. We developed and tested two metareasoning approaches: the first maintains constant image throughput, and the second maintains constant expected detection accuracy. The first approach switched between the InceptionV2 and MobileNetV2 image classification networks with a Single Shot Multibox Detector (SSD) attached. The second approach was tested on each network for a range of parameter values. The study also considered cases that used the system's built-in throttling method to control the temperature. Both metareasoning approaches were able to stabilize the device temperature without relying on throttling. Keywords: Neural networks; controls; thermodynamics

1 INTRODUCTION

Many autonomous systems rely on image processing for object detection. Tesla autopilot, for example, uses video feeds from cameras around the vehicle to recognize people, stop signs, and other objects critical to the driving experience [1]. Quickly processing images using neural networks, which require millions of computations, increases the processor's temperature [2].

Normally, the processor's temperature will fluctuate as the computational load varies, but when a computationally heavy task such as image processing is performed continuously, the temperature may increase enough to damage the processor [3].

Thus, it is important to have a thermal management approach that can maintain the processor's temperature in an acceptable range. This paper describes a study that considered both a traditional throttling approach and a new metareasoning approach. Metareasoning can monitor the device temperature in order to control the image processing procedure by switching between different neural networks and adjusting the processing frequency to adapt to changes in both the ambient and device temperatures.

Section 2 details background information about the problem of thermal management, reviews related studies, and emphasizes why this study was performed. Section 3 describes the experimental apparatus on which the study was performed, the design of the algorithm that managed the apparatus' temperature, and the variations in parameters between experiments. Section 4 provides key results determined from the performed exper-

iments. Section 5 discusses the differences between notable tests and highlights the effect of test parameters on the corresponding results. Section 6 summarizes the nature of this study and draws relevant conclusions from the results.

2 BACKGROUND

2.1 Thermal Management

The fundamental problem of thermal management arises from two conflicting characteristics of a processing unit: (i) when active, the device generates heat at a rate that is proportional to the rate of performed computations (which increases its temperature), but (ii) the device lifetime is inversely correlated with the mean operating temperature. This has led to the development of thermal throttling strategies as processing units are pushed towards ever-higher computation rates. These strategies fall into two main categories depending on what actuator they are using to control the temperature.

2.1.1 Internal Management Internally-managed systems rely on changes to the input to the processing unit to lower the number of computations performed, thereby lowering the operating temperature. A popular strategy for achieving this is dynamic voltage and frequency scaling (DVFS) [4]. This method adjusts the input frequency to the processing unit to slow computation frequency. The slower computation frequency also leads to a lower operating temperature and power consumption.

Das et al. proposed a reinforcement learning strategy for distributing computational workload across cores via threads while also modifying CPU frequency to thermally throttle electronic devices [5]. They were able to achieve a 100 to 200% increase in mean time to failure (MTTF) by controlling the device temperature.

2.1.2 External Management Alternatively, the temperature of the processor can be controlled by changing the environment in which it operates. Typically, this is done by providing increased cooling through increased conductive or convective heat transfer.

Convection can be increased passively by increasing the surface area of the processor through fins, or actively by increasing the convection coefficient through increased mass flow. Benoit-Cattin et al. [6] showed that dynamic active cooling can increase the efficiency of image processing on a Raspberry Pi. Their approach, which used an external fan that was controlled by the Raspberry Pi 4B, increased the image processing throughput.

2.2 Metareasoning

Metareasoning is a higher-level form of programmatic thinking that can lead to improved performance in autonomous agents. Metareasoning achieves this by monitoring the agent and its decision-making environment to determine how to approach its current decision [7], [8].

We are unaware of any work explicitly detailing metareasoning as a method for thermal management. There is, however, some use of metareasoning in related areas - particularly, image processing.

Nguyen et al. [9] used “frame skipping” to achieve a desired output video quality. Given a set input frame rate, their system was designed to discard certain images if a consistent output frame rate was not maintainable due to limited computational resources on the client-side of the gaming system. Although this was not explicitly considered metareasoning by the authors, their implementation involved a ”decision engine” which would ”decide the optimal frame rate given the current system status.” Thus, this is a type of metareasoning.

Although DVFS is a proven methodology for power and thermal management, it does not perform optimally with tasks that require short bursts of high CPU utilization [10]. Das et al.’s methodology is a welcome improvement, but it is not application specific and is therefore unable to make more context sensitive trade-offs, such as object detection accuracy. Additionally, not all devices can be cooled externally, so the work by Benoit-Cattin et al. cannot be applied to systems that rely only on passive cooling. Finally, the approach used by Nguyen et al. is designed to maintain a desired frame rate, but it does not account for temperature.

Thus, the study described here sought to determine whether a metareasoning approach would be able to control a processor’s temperature, which would be useful in situations where other techniques are infeasible or expensive.

3 EXPERIMENTAL SETUP

Our experimental apparatus performed a range of tests on a set of two algorithms designed to implement a metareasoning approach for managing the temperature of a processor.

3.1 Hardware

We tested the metareasoning approaches on a Raspberry Pi 4B. This device was chosen because of its low price, popularity, and our familiarity with the default operating system, Raspbian. The device was flashed with the 2021-05-07 release of the 32-bit version Raspberry Pi OS firmware.

The Raspberry Pi uses a Broadcom BCM2711 system on a chip (SoC), which contains a quad-core Cortex-A72 processor [11]. All four course of the processor are used during testing. The model variant with 2 GB of random access memory (RAM) was used. The SoC features an internal temperature sensor from which data was collected [11]. The device was powered by a 5.1 volt, 3.5 amp USB-C power supply.

FIGURE 1. RASPBERRY PI 4B IN ITS CASE WITH FAN.

A 3D-printed case was created for the device [12]. A small 5V fan was installed in the case and controlled with the GPIO Python package [13]. The fan vented air out of the case. The final setup is shown in Fig. 1.

3.2 Algorithms

Image processing was performed with openCV based on a real-time object detection program written by Adrian Rosebrock [14, 15]. The neural networks used for image processing were trained on the COCO dataset using the Tensorflow object detection API [16,17]. These networks were then adapted for use with the openCV deep neural network module by the openCV team [18]. The COCO images were resized to 300 x 300 pixels for input into the neural networks.

We used two neural networks that combine object classification and object detection:

1. MobileNetV2 with Single Shot Multibox Detector (SSD) [19, 20]
2. InceptionV2 [2] with SSD

Object detectors are typically rated by the proportion of objects they correctly identify. The detectors provide multiple classifications of an object with probabilities assigned to each. "Top-1 accuracy" is the proportion of objects for which the detector assigned the highest probability the correct classification. By this definition, the InceptionV2 network has an Top-1 accuracy of 73.9% on the ImageNet database [17]. Testing of MobileNetV2 on the same database resulted in a measured Top-1 accuracy 71.81% [21].

There is a known trade-off between classification accuracy and speed in object detector research [17]. Accordingly, image classifications using InceptionV2 require more time than using MobileNetV2. Trials on the Raspberry Pi demonstrated a time of approximately 1.8 seconds for InceptionV2 to process an image. MobileNetV2 processed the same image in approximately 0.8 seconds. Image throughput is the inverse of processing speed. Therefore, MobileNetV2 has a throughput of 1.25 frames per second (FPS), while InceptionV2 has a throughput of approximately 0.55 FPS.

To alleviate computational load, the computational rate must be decreased. Our proposed method does this on a "macro" level by inserting a pause of varying duration during an image processing loop. While paused, the device idles at a low CPU load. On average, this decreases the computational frequency, therefore lowering the SoC heat output and the measured temperature.

We created and tested two metareasoning algorithms. The first method, Algorithm 1, inserts a pause with a duration proportional to the overshoot of the desired temperature. Because only one network is used, average accuracy remains constant while throughput drops over time to maintain a constant temperature. Here, metareasoning manifests as dynamic parameter adjustment during each test. The second method, Algorithm 2, switches between InceptionV2 and MobileNetV2. If the measured temperature is too high, the algorithm switches from InceptionV2 to MobileNetV2. To maintain constant throughput, a one second pause is inserted after MobileNetV2 processes an image. Therefore, a small accuracy trade-off is made to preserve constant temperature and throughput. In this algorithm, metareasoning manifests as both parameter adjustment and algorithm switching.

Note that, if the device were processing incoming video, incoming frames would be dropped when the image processing loop was too long to keep up with the incoming video.

3.3 Design of Experiments

For Algorithm 1, we performed multiple experiments with different values for the pause adjustment coefficient, p_a, and the initial pause duration, p_d. Because these tests depend on the temperature of the SoC, it was important that all tests began at the same temperature and have similar ambient thermodynamic environments. While idle, the temperature of the SoC in a 22°C room and without active cooling is approximately 57°C. Be-

Algorithm 1 Pause Duration Adjustment

Require: $T \leq 50°C$
  $t_0 \leftarrow 0s$ ▷ Set test start time
  $t_s \leftarrow 300s$ ▷ Set test duration
  $T_d \leftarrow 75°C$ ▷ Set desired temperature
  $p_d \leftarrow p_d$ ▷ Initialize pause duration
  $p_a \leftarrow p_a$ ▷ Set pause adjustment coef.
  while $t < t_s$ **do** ▷ t is the program time
    Process image
    Record N_d ▷ The processing duration
    Record T ▷ The current temperature
    $p_d \leftarrow p_d + p_a \times (T - T_d)$
    if $p_d < 0s$ **then**
      $p_d \leftarrow 0s$
    end if
    Wait p_d seconds
    Record L_d ▷ The loop duration
    Record T
  end while

Algorithm 2 Network Switching

Require: $T \leq 50°C$
  $t_0 \leftarrow 0s$
  $t_s \leftarrow 300s$
  $T_d \leftarrow 75°C$
  $p_d \leftarrow 0s$
  Network ← InceptionV2
  while $t < t_s$ **do**
    Process image using Network
    Record N_d
    Record T
    Wait p_d seconds
    if $T > T_d$ **then**
      Network ← MobileNetV2
      $p_d \leftarrow 1s$
    else
      Network ← InceptionV2
      $p_d \leftarrow 0s$
    end if
    Record L_d
    Record T
  end while

tween tests, the device was cooled via convective heat transfer by the 5V fan attached to the case. The cooling cycle, which ran before each test if the temperature was greater than the set starting temperature, activated the fan until a specific temperature was reached. So this cycle was activated even if the device was near its idle temperature, the starting temperature was chosen to be 50°C, which is below the idle temperature. The fan was not active during tests.

TABLE 1. VARIANTS OF P_D AND P_A USED TO CREATE EACH SET OF 25 TESTS.

p_d [s]	p_a [s/°C]
0	0
0.5	0.050
1.0	0.100
1.5	0.150
2.0	0.200

We performed 25 tests for each network. We incremented the values of both p_a and p_d to create the set of tests, shown in Tab. 1.

We tracked multiple variables during each test, measured once after the image was processed and again after the pause was inserted. The most important of these were SoC temperature, SoC CPU use, loop length, and loop length goal.

For each value of p_d, we performed a test where p_a was zero. This represents a program where the loop length is static and metareasoning is inactive. In these tests the temperature reaches a maximum of approximately 82°C.

This occurs because, whenever the temperature of the Raspberry Pi 4B is above 75°C, the CPU operating speed is lowered via DVFS [22]. In practice, the observed throttling temperature is 82°C, seen in Fig. 2.

For Algorithm 2, fewer tests need to be performed. In these tests, p_a was not varied because it were not used in the program, while p_d was not varied between tests because a static value was assigned to each network. Additionally, only the switch from InceptionV2 to MobileNetV2 needs to be tested. Switching from MobileNetV2 to InceptionV2 would have an effect opposite to what is intended. Therefore, only one test is performed for Algorithm 2.

4 RESULTS

4.1 Algorithm 1

When metareasoning was active ($p_a > 0$), the SoC temperature stabilized around the desired temperature T_d, as shown in Fig. 3. For each network, only tests 1, 5, 21, and 25 are shown for visual clarity.

In Tests 5 and 25, p_a = 0.2 s/°C. In both tests, the SoC temperature successfully stabilized around T_d. Figure 4 shows the corresponding loop length and pause length in matching colors. Although Tests 1 and 21 have a constant set loop length, the loop length in Test 1 began to increase once the temperature reached approximately 82 °C, corresponding with the Raspberry Pi 4B's thermal throttling temperature. In Tests 5 and 25, the loop length

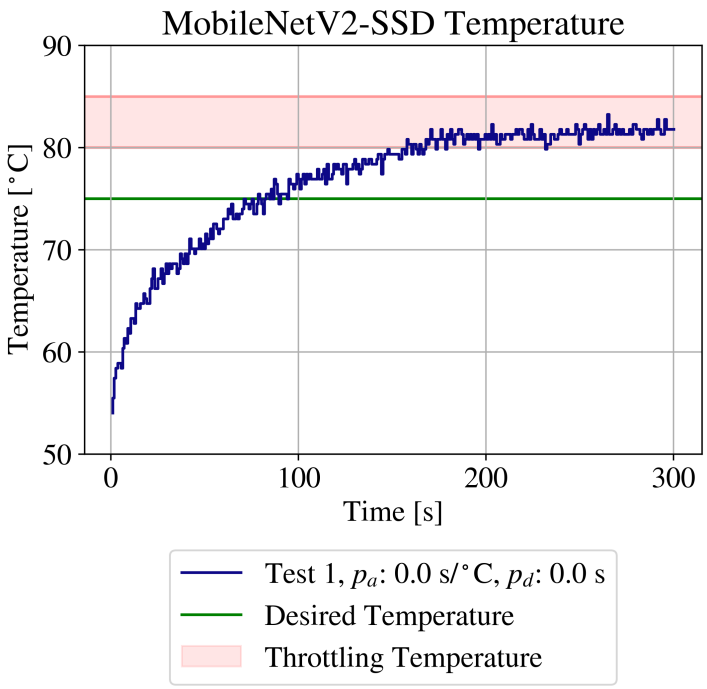

FIGURE 2. EXAMPLE OF RASPBERRY PI 4B SELF THERMAL THROTTLING VIA DVFS WHEN METAREASONING IS INACTIVE.

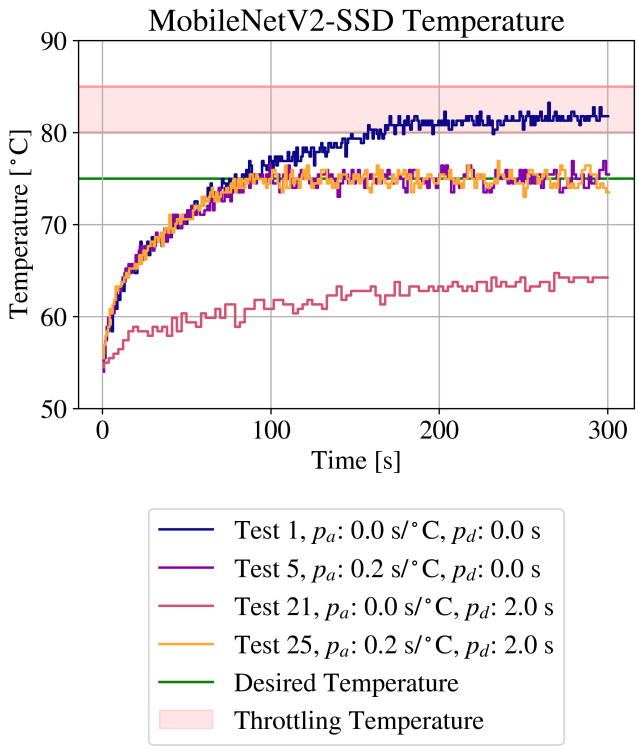

FIGURE 3. COMPARISON OF MEASURED TEMPERATURE BEFORE METAREASONING FOR ACTIVE AND INACTIVE METAREASONING FOR TWO VALUES OF P_D.

initially increased linearly to maintain the desired temperature. Figure 5 shows the CPU usage for all four tests. Tests 1 and 21 had a constant CPU usage, but, in Tests 5 and 25, the CPU usage decayed when the loop length was adjusted to maintain T_d.

The results for the program using Inception V2 with the SSD object detector demonstrated a faster rate of temperature increase, as well as a rapid increase in pause duration once the the temperature was greater than T_d, as seen in Fig. 6 and 7. Additionally, the CPU usage decreased asymptotically, as seen in Fig. 8.

4.2 Algorithm 2

Algorithm 2 was able to stabilize the temperature of the CPU as seen in Fig. 9. Additionally, the loop duration remained in a 0.2 second range of the 1.8 duration of InceptionV2, as seen by the consistent loop length shown in Fig. 10. The CPU utilization shown in Fig. 11 portrays a similar trend to Algorithm 1. Finally, the network switching is visualized in Fig. 12.

5 DISCUSSION

5.1 Algorithm 1

When metareasoning was not active ($p_a = 0$), the longer-than-needed loops continued, and the temperature remained low. When metareasoning was active ($p_a = 0.2$), metareasoning quickly reduced the loop length and maintained that until the temperature increased. At that point, metareasoning inserted pauses to maintain the temperature at the desired level.

The results with the InceptionV2 network were much different: because it required approximately 1.8 seconds, tests in which the loop length goal was 1.5 seconds led to running the network as quickly as possible. Although metareasoning inserted pauses to maintain the temperature when it increased, the temperature overshot the desired value. Because the loop length and pause length are adjusted only once per loop and the InceptionV2 processing time was much greater, the pause length changed more slowly than it did with the other networks.

5.1.1 Operational Cases In testing for Algorithm 1, four use operational cases become apparent, as shown in Tab. 2. In the table, "metareasoning" is shortened to "MR." Each test where metareasoning was inactive falls entirely into one of two categories depending on the value of the pause duration, p_d. When $p_d, p_a = 0$, thermal throttling occurs. This is the control test. When $p_a = 0, p_d > 0$, there is a static pause and no adjustments to the pause duration are made. When $p_a > 0, p_d = 0$, the

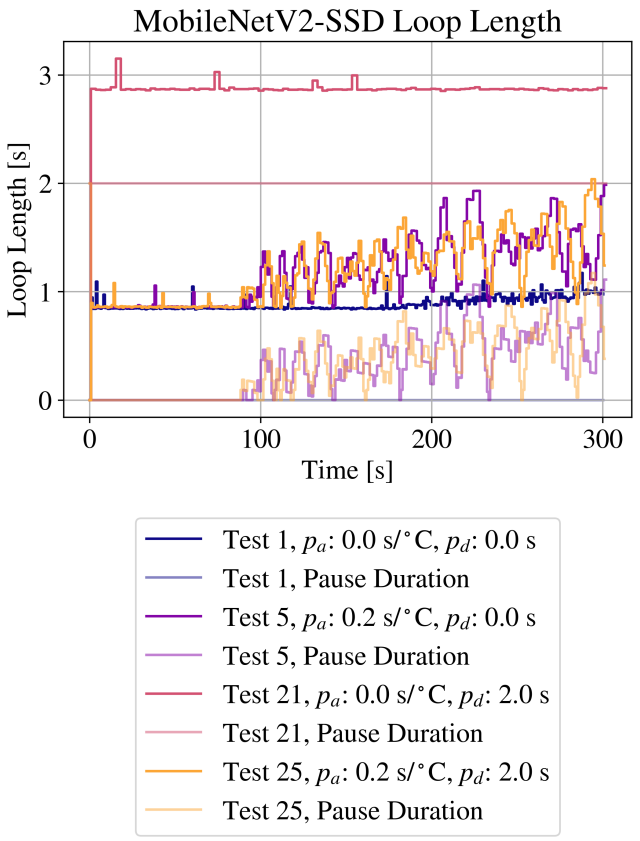

FIGURE 4. COMPARISON OF LOOP AND PAUSE DURATION FOR ACTIVE AND INACTIVE METAREASONING FOR TWO VALUES OF P_D.

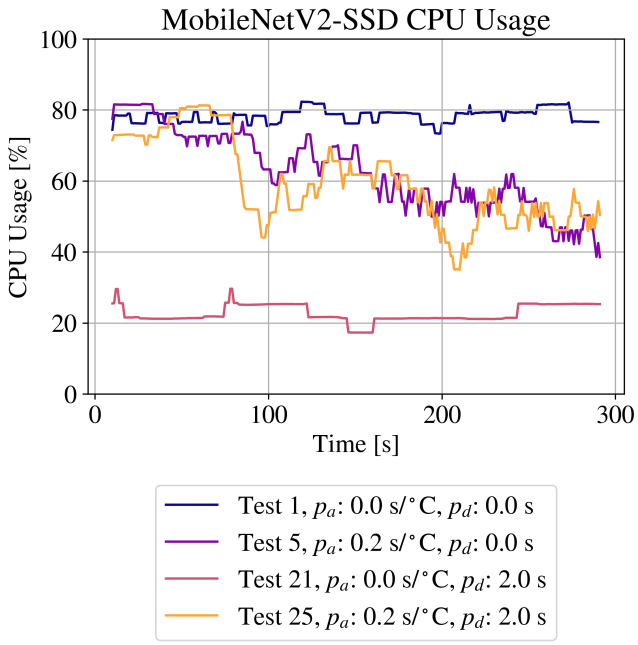

FIGURE 5. COMPARISON OF 20-SECOND MOVING AVERAGE OF CPU USAGE FOR ACTIVE AND INACTIVE METAREASONING FOR TWO VALUES OF P_D USING ALGORITHM 1. NOTE THAT THE LOOP DURATIONS ARE SHOWN IN SATURATED COLORS, WHILE THE PAUSE DURATIONS ARE SHOWN IN MATCHING FADED COLORS.

pause duration starts at zero and only increases once the temperature is higher than T_d. Finally, when $p_a, p_d > 0$, the pause duration is decreased to zero, then increases once the temperature is greater than T_d.

5.1.2 Adjustment Coefficient Effect

The pause duration adjustment coefficient, had different affects on the temperature trajectory depending on its value, p_a, and the neural network processing duration, N_d. Generally, larger values of p_a resulted in faster convergence of the temperature towards the desired value, T_d. Additionally, because the pause duration was only adjusted once per loop, the pause duration was updated less frequently for InceptionV2 then for MobileNetV2. Therefore, despite using the same value for p_a, the temperatures for the InceptionV2 tests with active metareasoning in Fig. 6 show a slight overshoot at approximately 70 seconds. In contrast, the MobileNetV2 temperature for the same test cases in Fig. 3 has no visible overshoot. Values of p_a which are greater than those tested would likely cause overshooting oscillatory behavior. Therefore, there is an ideal value of p_a for each network and thermal environment.

TABLE 2. COMPARISON OF THE FOUR OPERATIONAL CASES WHICH OCCUR IN TESTING.

Condition	No MR ($p_a = 0$)	MR ($p_a > 0$)
$p_d = 0$	The network runs as fast as possible, the temperature increases, and thermal throttling occurs.	The network runs as fast as possible, the temperature increases to T_d, and MR increases p_d such that $L_d \geq N_d$.
$p_d > 0$	The network runs slowly and the temperature increases slowly.	MR reduces p_d until the network runs as fast as possible, and when T_d is reached, MR increases p_d such that $L_d \geq N_d$.

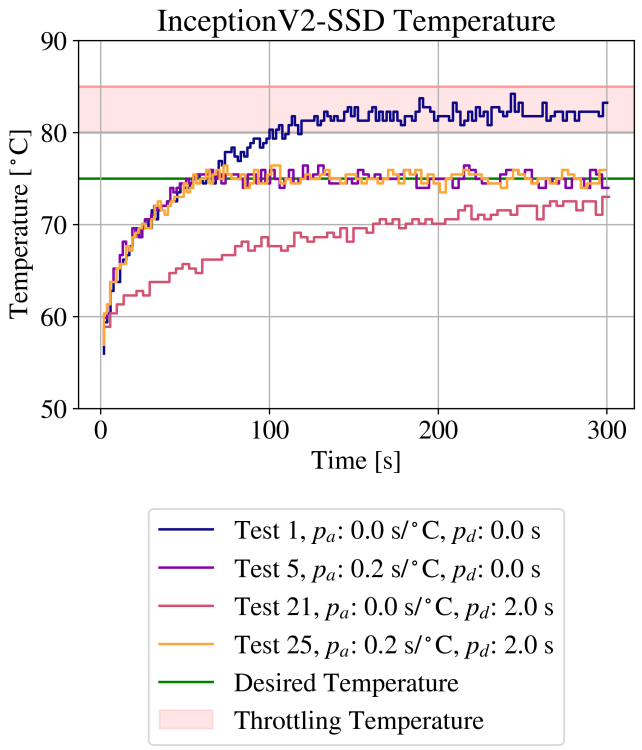

FIGURE 6. COMPARISON OF MEASURED TEMPERATURE PRIOR TO PAUSING FOR ACTIVE AND INACTIVE METAREASONING FOR TWO VALUES OF P_D.

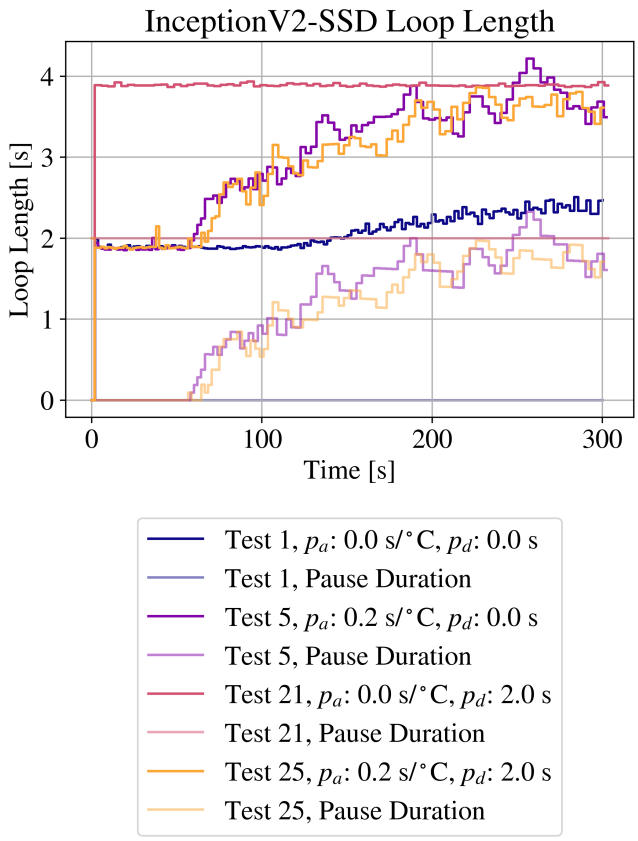

FIGURE 7. COMPARISON OF LOOP LENGTH AND PAUSE DURATION FOR ACTIVE AND INACTIVE METAREASONING FOR TWO VALUES OF P_D.

5.2 Algorithm 2

Figure 12 shows how InceptionV2 is used in 100% of loops at the beginning of the program, but this precentage decreases until MobileNetV2 becomes used in nearly 100% of the processing loops. It is notable that this transition follows a similar, inverted trend as the temperature in Fig. 2. This is likely because the temperature increases towards an asymptote representing the CPU's equilibrium temperature. Therefore, while switching to MobileNetV2 is not advantageous in the beginning of the test, it becomes increasingly advantageous as the test progresses because the processor is idling for more than half of each loop.

These results demonstrate that, because the temperature is controlled via software, the system can be designed to a reliability standard independent of the hardware. To achieve a longer mean time to failure (MTTF), the desired temperature can be lowered to 75 or 70°C [3].

This approach may be useful for managing the temperature on devices besides the Raspberry Pi 4B. A desired temperature can be set to below the device thermal throttling threshold, and the program will maintain the desired temperature.

Although these results demonstrate successful completion of their goal, the implementation of metareasoning has a small negative effect on performance. There is a small amount of time, no more than approximately 70 milliseconds, added to each program loop. This accounts for no more than approximately 4% of processing time for InceptionV2, or 8% for MobileNetV2. During this time, image processing is not occurring, so the overall throughput achieved by either algorithm is inherently lower than an algorithm without metareasoning using the same neural network.

6 CONCLUSION

This paper described a study that evaluated metareasoning algorithms for managing the temperature of a Raspberry Pi 4B by dynamically inserting pauses and switching neural networks in the image processing loop. The SoC's temperature was successfully stabilized around a desired reference using metareasoning regardless of the starting loop duration.

Inserting pauses of varying duration via metareasoning was shown to have a stabilizing effect on temperature, with increasing values of p_a providing a more robust solution by quickly stabi-

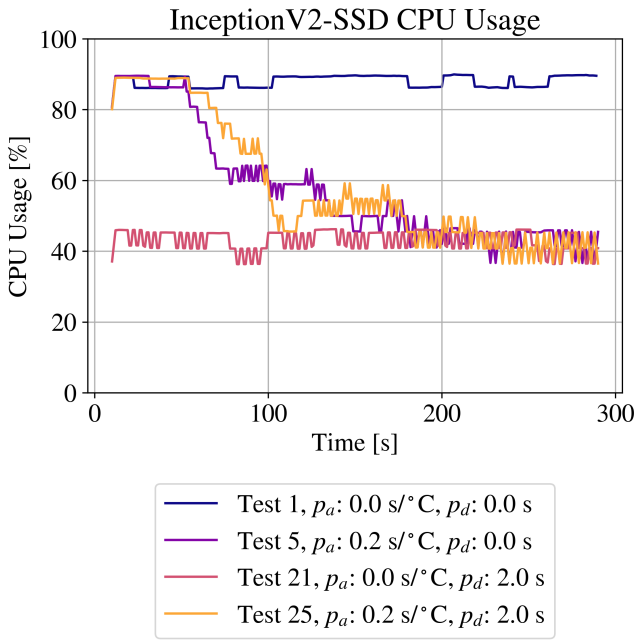

FIGURE 8. COMPARISON OF 20-SECOND MOVING AVERAGE OF CPU USAGE FOR ACTIVE AND INACTIVE METAREASONING FOR TWO VALUES OF P_D USING ALGORITHM 1. NOTE THAT THE LOOP DURATIONS ARE SHOWN IN SATURATED COLORS, WHILE THE PAUSE DURATIONS ARE SHOWN IN MATCHING FADED COLORS.

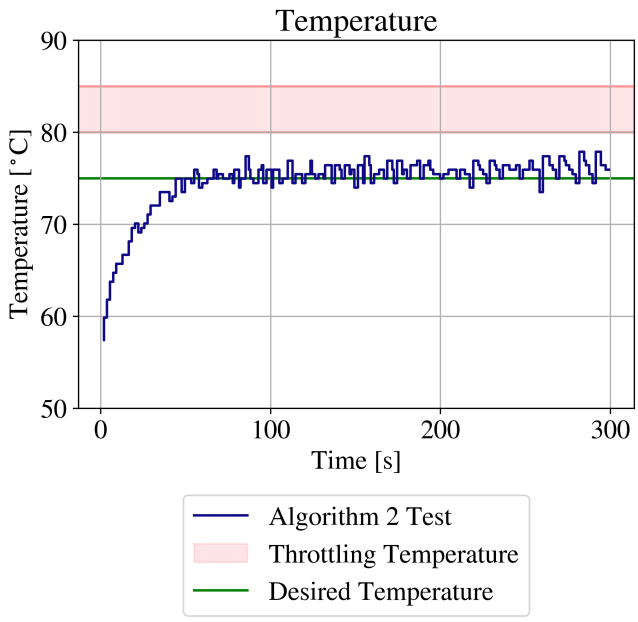

FIGURE 9. MEASURED TEMPERATURE OF THE CPU RUNNING ALGORITHM 2.

lizing SoC temperature for MobileNetV2 and InceptionV2.

Switching between neural networks with differing processing times and corresponding pause lengths to maintain a constant throughput also proved effective in stabilizing the SoC temperature. In this process, the expected average accuracy decreases over the course of the algorithm from that of InceptionV2 to that of MobileNetV2.

This work demonstrates the utility of metareasoning as a software-based approach for thermal management that can be used on other hardware platforms with minimal changes. Additional work is needed to consider how well this approach would work when the processor is performing other tasks at the same time and to determine whether controlling only some of the tasks is necessary for maintaining an acceptable temperature. In addition, it may be possible to use metareasoning to control the temperature of a processor that semi-periodically performs high-priority tasks by slowing or halting the execution of selected low-priority tasks. Finally, the temperature overshoot in Algorithm 1 might be reduced by framing the scenario as a control problem, where p_a is a proportional gain. Additional terms, such as a derivative gain, might then contribute to improved tracking or stability.

We are currently developing a metareasoning policy that will adjust both throughput and expected accuracy as the processor temperature increases.

ACKNOWLEDGMENT

This study is supported by the U.S. Army Research Laboratory (Award W911NF2120076).

REFERENCES

[1] Tesla Inc., 2022. Autopilot. https://www.tesla.com/AI.

[2] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z., 2016. "Rethinking the inception architecture for computer vision". In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2818–2826.

[3] Lakshminarayanan, V., and Sriraam, N., 2014. "The effect of temperature on the reliability of electronic components". In 2014 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), pp. 1–6.

[4] Choi, K., Dantu, K., Cheng, W.-C., and Pedram, M., 2002. "Frame-based dynamic voltage and frequency scaling for a mpeg decoder". In Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design, ICCAD '02, Association for Computing Machinery, p. 732–737.

[5] Das, A., Shafik, R. A., Merrett, G. V., Al-Hashimi, B. M., Kumar, A., and Veeravalli, B., 2014. "Reinforcement

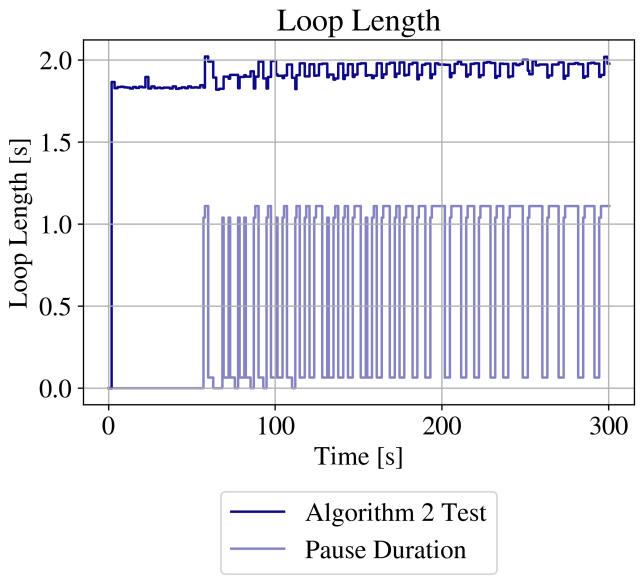

FIGURE 10. LOOP DURATION OF EACH IMAGE PROCESSING CYCLE DURING AN ALGORITHM 2 TEST. NOTE THAT THE LOOP DURATION IS SHOWN IN SATURATED BLUE, WHILE THE PAUSE DURATION IS SHOWN IN A MATCHING FADED BLUE.

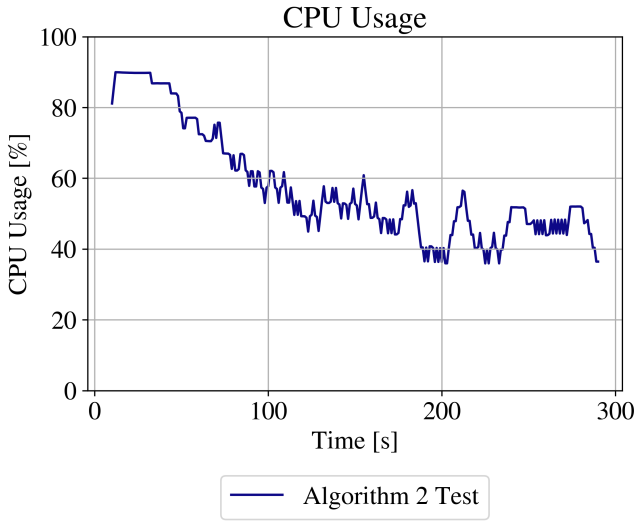

FIGURE 11. A 20-SECOND MOVING AVERAGE OF CPU USAGE DURING AN ALGORITHM 2 TEST.

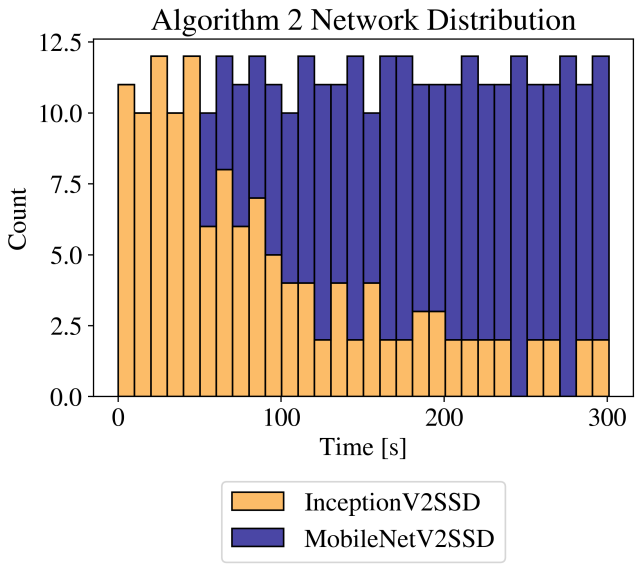

FIGURE 12. NETWORK USAGE OVER 10 SECOND INTERVALS DURING AN ALGORITHM 2 TEST.

learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems". In 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6.

[6] Benoit-Cattin, T., Velasco-Montero, D., and Fernández-Berni, J., 2020. "Impact of thermal throttling on long-term visual inference in a cpu-based edge device". *Electronics,* ***9***(12).

[7] Cox, M. T., and Raja, A., eds. *Metareasoning: Thinking About Thinking*. MIT Press. OCLC: ocn611551144.

[8] Langlois, S. T., Akoroda, O., Carrillo, E., Herrmann, J. W., Azarm, S., Xu, H., and Otte, M., 2020. "Metareasoning structures, problems, and modes for multiagent systems: A survey". *IEEE Access,* ***8***, pp. 183080–183089.

[9] Nguyen, D. V., Tran, H. T. T., and Thang, T. C., 2020. "A delay-aware adaptation framework for cloud gaming under the computation constraint of user devices". In MultiMedia Modeling: 26th International Conference, MMM 2020, Daejeon, South Korea, January 5–8, 2020, Proceedings, Part II, Springer-Verlag, p. 27–38.

[10] Wang, Q., Kanemasa, Y., Li, J., Lai, C. A., Matsubara, M., and Pu, C., 2013. "Impact of dvfs on n-tier application performance". In Proceedings of the First ACM SIGOPS Conference on Timely Results in Operating Systems, TRIOS '13, Association for Computing Machinery.

[11] Raspberry Pi Ltd. Raspberry Pi Documentation - Processors.

[12] Malolo, 2019. Malolo's screw-less / snap fit customizable Raspberry Pi 4 Case and Stands. https://www.thingiverse.com/thing:3723561.

[13] Croston, B. RPi.GPIO: A module to control Raspberry Pi GPIO channels. http://sourceforge.net/projects/raspberry-gpio-python/.

[14] OpenCV team, 2021. Opencv. https://opencv.org/.

[15] Rosebrock, A. Real-time object detec-

tion with deep learning and OpenCV. https://www.pyimagesearch.com/2017/09/18/real-time-object-detection-with-deep-learning-and-opencv/.
[16] COCO Consortium, 2017. COCO - Common Objects in Context. https://cocodataset.org/home.
[17] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., and Murphy, K., 2017. "Speed/accuracy trade-offs for modern convolutional object detectors". In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3296–3297.
[18] OpenCV team, 2018. TensorFlow Object Detection API · opencv/opencv Wiki. https://github.com/opencv/opencv/wiki/TensorFlow-Object-Detection-API.
[19] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C., 2019. Mobilenetv2: Inverted residuals and linear bottlenecks.
[20] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C., 2016. "Ssd: Single shot multibox detector". *Lecture Notes in Computer Science*, p. 21–37.
[21] Bianco, S., Cadene, R., Celona, L., and Napoletano, P., 2018. "Benchmark analysis of representative deep neural network architectures". *IEEE Access,* ***6***, pp. 64270–64277.
[22] Raspberry Pi Ltd. Raspberry Pi Documentation - Frequency Management and Thermal Control.