Artificial Intelligence on the Edge for Space: Benchmarking Quantisation, Latency, and Power Consumption for On-board Inference

Jiarong Zhang[1], Yasir Latif, Tat-Jun Chin

***Abstract*— Limited power and computation budgets available on-board small satellites make space a challenging scenario for intelligent decision making. At the same time, many on-board problems can greatly benefit from the application of modern artificial intelligence techniques. In this work, we explore the use of specialised edge hardware to enable low latency, low-power on-board inference. Commercial hardware developed for edge applications poses constraints on the type of network architectures that can be optimally run on them. In particular, we use an object detection task to explore the various ways of training models for the Google Edge TPU and how they affect task performance during on-board inference. We specifically look at how power consumption, latency, and performance are affected by various training choices such as quantisation. Results are presented for a combination of these parameters, along with their effects and recommendations to facilitate the adoption of new tasks on edge hardware for space applications.**

I. Introduction

From rovers to satellites, many space technologies will greatly benefit from energy efficient, low latency on-board AI as it enables near-real time situational awareness and decision making. This is especially important when communication with ground control is either sparsely available (line of sight to a receiving station) or not guaranteed at all (deep space) due the the enormous distances involved. On-board AI enables satellites to autonomously make mission critical decisions in-situ without operator input.

Recent advancements in space hardware have reduced both the cost and physical size of satellites while ride-share programs have made space more accessible, further emphasising the need for on-board autonomy. Cubesats are a standardised subclass of small satellites with a form factor based on 10 cm cubes units (1U), with each unit weighing less than 2 kilograms. They are preferred over conventional satellite development strategies due to reduced development costs and time. However, a compact size also means limited power generation, and hence, limited computing and communication capabilities. A 3U Cubesat can produce 0.13 kWh per day [1] on average from solar cells, where as each 1U can provide about 0.04 kWh daily. For comparison, the Mars Perseverance rover, which is the size of a car and weighs approximately 1 tonne, is equipped with a radioisotope power generator capable of producing 2.6 kWh per day [2]. While AI has enabled capabilities such as slippage and immobilisation detection [3], clearance and collision detection [4], and energy-aware path planning [5] on the Perseverance rover, on-board machine learning capabilities for small satellites are still sparse. This can largely be attributed to the limited power budgets available on the smaller platforms, making computationally heavy AI payloads difficult to operate in-orbit, as well as the lack of available datasets to train AI models for space applications.

A promising computation paradigm, edge computing, encourages processing data closer to the source for better efficiency, autonomy, and optimised network bandwidth utilisation. It envisions a lot of small, low-powered devices making local inferences and decisions at the source, instead of transmitting raw data elsewhere for remote decision making. This approach is well-suited for the small satellite use case. Hardware developed for machine learning on the edge consists of low-powered application-specific integrated circuits (ASIC) [6], [7] that offer trillions of operations per second (TOPS) per watt of power. Such hardware addresses both the power and computational constraints for in-orbit autonomy on-board a small satellite. However, being application specific hardware, the types of computations that can be run on them is limited, as opposed to a General Purpose Graphical Processing Unit (GPGPU). Therefore, general purpose machine learning algorithms need to be adapted to the specific hardware to conform to underlying architectural choices. These choices affect representation of floating point numbers and the types of neural network layers than can be emulated in hardware.

This work serves as a practical guide for deploying machine learning algorithms, specifically from an object detection perspective, to edge devices for space applications. We outline the training methodology that enables using a general purpose GPU for training and the edge device for inference alone, as previously explored in [8], [9]. We further focus on the comparison of GPU models (running on general purpose hardware without any power considerations) and their edge counterparts in terms of task performance, power consumption and model size. We build upon the work of [10], [11], [12] who investigated real-time object detection on edge hardware but did not comment on the aforementioned metrics. This work aims to address the practical aspects of on-board machine learning through the task of object detection. We report findings and make best practice recommendations for using on-board AI in the space setting.

II. Related Work

AI for space has received a lot of attention recently with emphasis on efficient on-board inference [7], [13]. Furano et al. [8] considered on-board machine learning from the perspective of Earth observation and proposed on-board

All the authors are with the University of Adelaide, Australia.
[1]emily.zhang@student.adelaide.edu.au

image preprocessing to reduce data storage and downlink bandwidth. They found that by identifying images with significant cloud cover, more than 50% of imagery can be discarded and only useful data is sent to ground, saving bandwidth. To update on-board algorithms, rather than learning on-board, models are trained on the ground and updated via satellite uplink. As edge-compatible models are typically compressed, this is computationally more efficient. A later work by Furano et al. [9] discussed hardware challenges such as power budget, memory budget, dependability issues in the harsh space environment, as well as the availability of training data for new instruments. For reliability, applying AI in isolation is recommended so that any failures in local tasks do not propagate to the rest of the satellite. To circumvent the initial lack of training data for on-board models, training with simulated data is recommended until real data can be collected using on-board sensors and models updated. The space environment is harsh on electronics due to cosmic radiation, and unprotected devices are susceptible to random soft errors such as bit-flips. Enabling on-board machine learning for satellites will require additional work towards radiation hardening of the hardware. Efforts towards radiation characterisation of edge-AI microchips such as the the Edge TPU and the Intel Myriad X [7] will enable widespread use on small satellites.

Several recent works have demonstrated applications of machine learning on the edge using various edge accelerators. The Google Edge TPU [6] has been applied for tasks such disease detection in bees [10], fruit yield detection [11], as well as for satellite pose estimation [12]. These works discuss model training, quantisation, and accuracy/latency trade-offs involved. These works utilised quantisation on their detection networks, and the findings generally indicate that the model accuracy on Edge TPU is comparable to full-size state-of-the-art GPU models while achieving higher throughput. Interestingly, the satellite post estimation [12] work reported a slight increase in accuracy for the detection network after converting the floating-point, quantisation-aware trained network to use 8-bit weights and activations. However, there are no results for the non-quantisation-aware trained network for comparison. Additionally, the total power draw of the Dev Board Mini between CPU and TPU inference is reported, but the power consumption of the TPU independent of the board is not referenced. Additionally, the reduction of file size is not reported either.

III. Methods

Objects in orbit around Earth are tracked using two dominant modalities: radar [14] and optical telescopes [15]. In these methods, the sensor is pointed at selected angles toward the sky and collects information about objects which pass through its field of view. While radar measurements can provide information like size, altitude, and orbital inclination of space objects, optical measurements examine light curves to determine the material and density of space objects. Space objects can be tracked by combining data from radar and optical measurements, such as size and material composition. Information about each tracked object is collected and updated frequently to predict collisions with satellites, which would allow time for mission engineers to plan for evasive manoeuvres if required [16].

Ground-based observations are limited by the range of the sensor and objects further from Earth may not be observed at all. Satellites in deep space cannot rely on ground-based observations. Moreover, a satellite must also be able to receive radio frequency communications in order to follow manoeuvring commands. If a satellite has delayed communications with ground control due to distance, frequency band access time, or is unable to establish communications at all due to malfunctioning subsystems, then it has to operate in an autonomous mode, requiring precise awareness of its surroundings. These factors highlight the significance of autonomous object detection and avoidance for satellites, as the reliance on ground-based observations and communication would be eliminated. Due to its practical significance, we explore object detection as the sample task to solve on-board a satellite. In its essence, the problem demonstrates the capabilities of the general class of object detection tasks, and the conclusions from this work generalise to similar on-board object detection tasks such as space-based Earth observation.

Edge accelerators in general and the Google Edge TPU [6] in particular leverages fast multiply-accumulate operations in parallel to provide low-power, high-throughput neural network inference. However, the network weights must be represented as integers rather than the usual floating point numbers. This requires a quantisation step (see Sec. III-A), which needs to be applied to convert the model into an edge accelerator compatible representation. Therefore, training for on-board inference can be addressed in mainly two ways as shown in Fig. 1, which includes three main steps: fine-tuning, quantisation, and compiling the model for Edge TPU. We discuss these steps in the rest of this section.

A. Quantisation

The Google Edge TPU only supports models with 8-bit fixed-point precision due to limited on-chip memory. In deep learning, neural network models typically operate using high-precision (32-bit) floating-point numbers. However, for deployment on edge devices or for efficient inference in general, it is desirable to use lower-precision, fixed-point data types (e.g. 8-bit integers) as low precision inference consumes less memory and computational resources. Quantisation is the process where a model is compressed; model weights are mapped from 32-bit to 8-bit precision, reducing file sizes by up to 75%. Quantisation by its nature is lossy and will reduce the inference accuracy if carried out blindly. However, the effects of quantisation can be minimised by performing quantisation-aware training (QAT) which simulate the quantisation step during the training process, allowing the network to be trained for the conditions that will be observed during inference on the edge device. QAT is only viable when the model can be re-trained (there is enough training data including supervision labels). However, this might not always be possible. In such

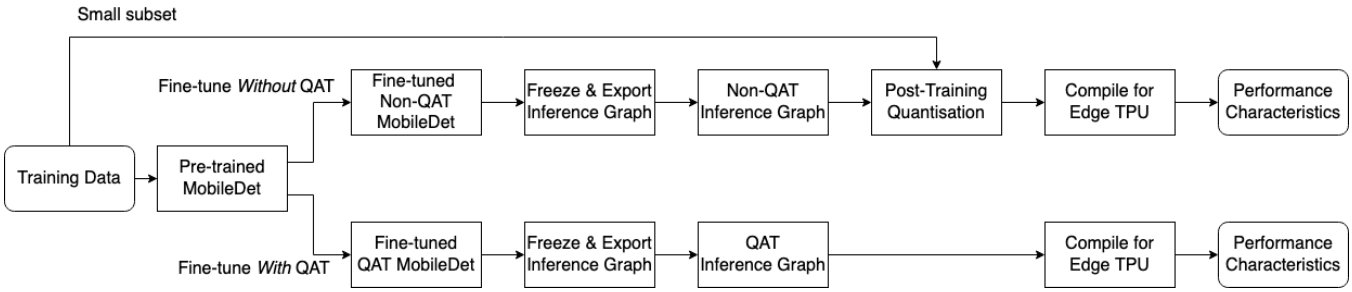

Fig. 1: Visualisation of training configurations for `MobileDet` architecture.

situation, a pretrained model can be quantised through post-training quantisation (PTQ) using a representative subset of input data (no supervision) so that intermediate quantisation parameters can be learned from the distribution of data and the subsequent weight parameters. PTQ can be performed on frozen model inference graphs with as little as 100 sample images. We explore the effect of both these quantisation techniques on the performance of the object detection task.

1) Quantisation-aware Training: During quantisation-aware training, "fake" quantisation nodes are added to the model graph around computational operations to simulate the effect of reduced numeric precision in model weights and parameters. They are called fake because the inputs and outputs are still floating-point numbers. In 8-bit quantisation, floating-point numbers (r) are reduced to 8-bit representations (q) using scale (S) and zero-point (Z) parameters:

$$q = \frac{r}{S} + Z \tag{1}$$

The zero-point is a parameter that represents the quantised value that equates to 0 in floating point, while scale is the factor by which the lowest and highest values in the floating-point range (f_{min}, f_{max}) are mapped to the lowest and highest values in 8-bit representation i.e. $[q_{min}, q_{max}] = [-128, 127]$,

$$S = \frac{f_{max} - f_{min}}{q_{max} - q_{min}} \tag{2}$$

In the fake quantisation nodes, floating-point (r) parameters and converted to 8-bit integers (q), calculations are performed in 8-bit precision and the result is converted back into floating-point. This leads to reduction in the prediction accuracy, due to a loss in numeric precision and rounding (quantisation error); this process is illustrated in Fig. 2. The quantisation error encourages the network to learn parameters that are robust to quantisation. The network must adjust parameters to make accurate predictions while considering lower precision calculations.

2) Post-training Quantisation: In cases where the model cannot be re-trained due to lack of supervision labels or access to model checkpoints, models can be quantised through post-training quantisation (PTQ). This method is performed on the frozen model inference graph, that is the network weights do not change in the process, as illustrated in the *Fine-tune Without QAT* path on Fig. 1. PTQ estimates the minimum and maximum possible values of variables in the network (i.e., weights, inputs, biases) by running inference through a set of example inputs, known as the representative dataset; inputs in the representative dataset must be of the same format and size as the original training dataset, though only a small sample is required (around 100 images are considered here). PTQ converts floating-point values to 8-bit integers using the previously mentioned conversion equation, using f_{min} and f_{max} obtained from inference on the representative dataset to calculate the scale factor (S). Unlike QAT models, PTQ models cannot compensate for quantisation error which results in a less accurate model, especially for larger networks.

TABLE I: Model summary for `MobileDet` and `SpaghettiNet` architectures.

Model Architecture	Multiply-Accumulate Operations (MACs)	Quantised File Size (MB)	Pixel 4/6 Edge TPU Speed (ms)[1]
`MobileDet-EdgeTPU`	1.53 billion	4.3	6.9
`SpaghettiNet-S`	1.00 billion	3.7	1.3
`SpaghettiNet-M`	1.25 billion	4.4	1.4
`SpaghettiNet-L`	1.75 billion	6.0	1.7

[1] `MobileDet` was evaluated on Pixel 4, while `SpaghettiNet` variations were evaluated on Pixel 6. Adapted from [17]

B. Model Architectures

To evaluate the effects of the different quantisation techniques, two edge-compatible object detection models are considered in this work: `MobileDet-EdgeTPU` and `SpaghettiNet-EdgeTPU-S`. `MobileDets` [18] are a family of object detection networks developed for optimised inference on low-power hardware using neural architecture search (NAS). Previously, [11] and [12] among others, have used `MobileDet` for real-time object detection on edge, with [11] reporting better performance than hand-designed edge-compatible architectures.

Similarly, `SpaghettiNet` was also developed using NAS specifically for the Edge TPU, but with different computation budget allocations and connection patterns than MobileDet; there are three versions of `SpaghettiNet` - *S, M,* and *L*, alluding to the amount of multiply-accumulate operations (MACs) used by each variation. We use the *S* variation as it has the smallest file size and better latency than its larger counterparts. Model details are summarised

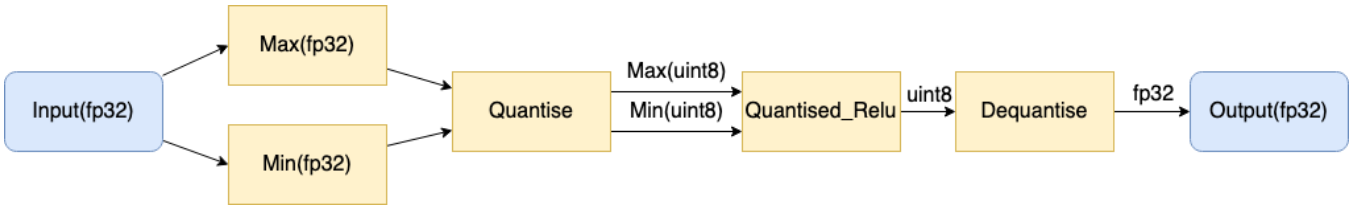

Fig. 2: Visualisation of fake quantisation nodes.

in Table I, outlining computational details and performance comparisons on mobile phone platforms.

C. Porting to the Edge TPU

A quantised model must be compiled specifically for the Edge TPU to map all compatible operations in the network to run on the accelerator. This creates one amalgamated operation that will execute on the TPU upon inference. Models with unsupported operations can still be compiled, but the compiler will partition the model graph at the first point where an unsupported operation occurs. This means that only certain model architectures are able to take advantage of accelerated performance on the Edge TPU. As `MobileDet` and `SpaghettiNet` were designed specifically for the Edge TPU, both models were able to be fully compiled except post-processing steps such as generating detection boxes. However, for porting a general model, care should be taken in considering the types of layers supported by the particular edge accelerator to guarantee reasonable performance.

D. Dataset

The training split of the *Spacecraft PosE Estimation Dataset* (SPEED) [19] was used as the source of training and evaluation data for the object detection task. The training split SPEED consists of 12,000 synthetic grayscale images of the Tango satellite captured under diverse conditions, including varying distances, lighting conditions, poses, and background content, as seen in Fig. 3. The original dataset was designed for the task of satellite pose estimation. However, training regime, quantisation mechanism and final benchmarking can be generalised to any object detection task. The dataset is used mainly because of its size and relevance to the in-orbit detection task.

Ground truth bounding boxes were acquired from the code repository developed as part of [20]. The 12,000 images were randomly split into training/evaluation sets using 80:20 ratio, leading to 9,600 and 2,400 images in each set respectively.

IV. Evaluation

In this section, we evaluate the various training configurations and report performance not only in terms of the task accuracy, but also the power consumption and model size, which are all relevant parameters for in-orbit inference.

Both `MobileDet` and `SpaghettiNet` have originally been trained on the COCO dataset [21] for 400,000 steps. For our detection task, we further trained them an additional 40,000 steps on the 9,600 training images from the SPEED training split. Random horizontal flips and random crops were applied as data augmentation. Two variations of the `MobileDet` model are available: one trained with quantisation-aware training (uint8), and one without (fp32). We used the fp32 checkpoint to fine-tune two models for the detection task, with (QAT) and without quantisation-aware training (baseline). The baseline model is tested on a general purpose GPU to report "best-case" performance and then quantised using the post-training-quantisation (PTQ) technique. Both the quantised models are then ported to Edge TPU.

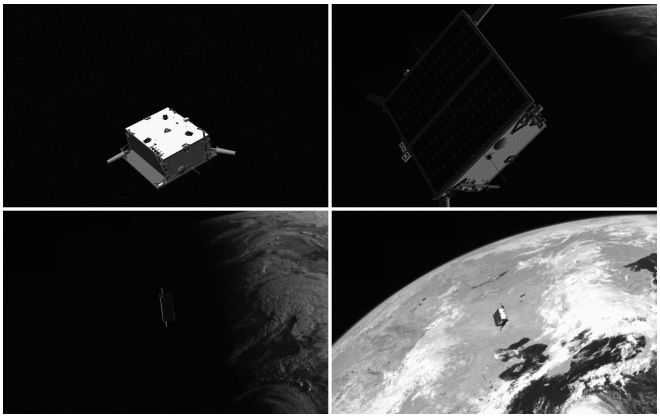

Fig. 3: Examples from the SPEED dataset, composed of synthetic images of the Tango satellite with varying distances, lighting conditions, and backgrounds.

However, the same approach is not applicable to the `SpaghettiNet` as only the quantisation-aware trained model is publicly available. Results are therefore presented for this variation only.

Each model was evaluated on the 2,400 images test on the Coral Dev Board Mini using the PyCoral API on top of TensorFlow Lite API. Model accuracy, inference time, power consumption and file size reduction are considered for both `MobileDet` and `SpaghettiNet` architectures.

A. Model Accuracy

The performance of each model is measured using the COCO object detection evaluation metrics. The key indicator of model accuracy is the mean Average Precision (mAP) score, which in COCO metrics is calculated as the mean of each detection class using averaged AP scores across 10 Intersection-over-Union (IoU) thresholds ranging from 0.5 to 0.95, with a step size of 0.05. AP summarises the area

under the curve of precision (P) and recall (R), which are calculated as

$$P = \frac{TP}{TP + FP} \tag{3}$$

$$R = \frac{TP}{TP + FN} \tag{4}$$

where TP, FP, and FN represent true-positive, false-negative, and false-positive results, respectively. An AP score is calculated for each IoU threshold. AP is calculated as the mean of precision values across 101 recall values between 0 and 1, with a step size of 0.01; the precision at each recall value (R) is found by taking the maximum precision value for any recall level greater than or equal to the current recall level.

$$AP = \frac{1}{101} \sum_{R \in (0, 0.01, 1)} P_{interp}(R) \tag{5}$$

$$P_{interp}(R) = max_{\widehat{R}:\widehat{R} \geq R} P(\widehat{R}) \tag{6}$$

The results are presented in Table II. The "Float" results were collected directly after training on the floating-point model, on a single NVIDIA RTX 2060, while the "8-bit" results were collected after compilation for Edge TPU, on the Coral Dev Board Mini. The Coral Dev Board Mini is a single board computer with an integrated Google Edge TPU.

There is a significant loss in model accuracy if post-training quantisation is applied to `MobileDet`. The accuracy loss in 8-bit PTQ `MobileDet` is expected as the network must work with lower precision than seen during training, and was not trained further to compensate for this loss. There is a 0.002 accuracy improvement for the QAT `MobileDet` model, and a 0.057 improvement for `SpaghettiNet` when converted to 8-bit and executed on the Edge TPU; this is similar to the findings in the previous satellite post estimation work [12]. We hypothesise that this improvement comes from denoising due to dimensionality reduction introduced by quantisation.

Quantisation-aware trained networks leads to a smaller accuracy loss between the floating-point baseline and 8-bit QAT `MobileDet` models. As the network parameters have been trained to expect 8-bit precision values, this results in better accuracy for the 8-bit QAT model. The 8-bit QAT `MobileDet` result has a small accuracy drop compared to the baseline floating-point model, but is considerably more accurate than the 8-bit PTQ model. The same comparison is not possible for `SpaghettiNet`, however one can deduce that it follows a similar pattern; the floating-point QAT model has lower accuracy than the non-QAT baseline, however 8-bit quantised QAT models have significantly better accuracy than PTQ models. See Fig. 4 for a visualisation of predicted bounding boxes by the three models.

TABLE II: Comparison of model accuracy before and after quantisation.

Model Architecture	mAP			
	Baseline (Float)	**QAT (Float)**	**PTQ (8-bit)**	**QAT (8-bit)**
`MobileDet-EdgeTPU`	0.875	0.868	0.724	0.870
`SpaghettiNet-S`	-	0.850	-	0.907

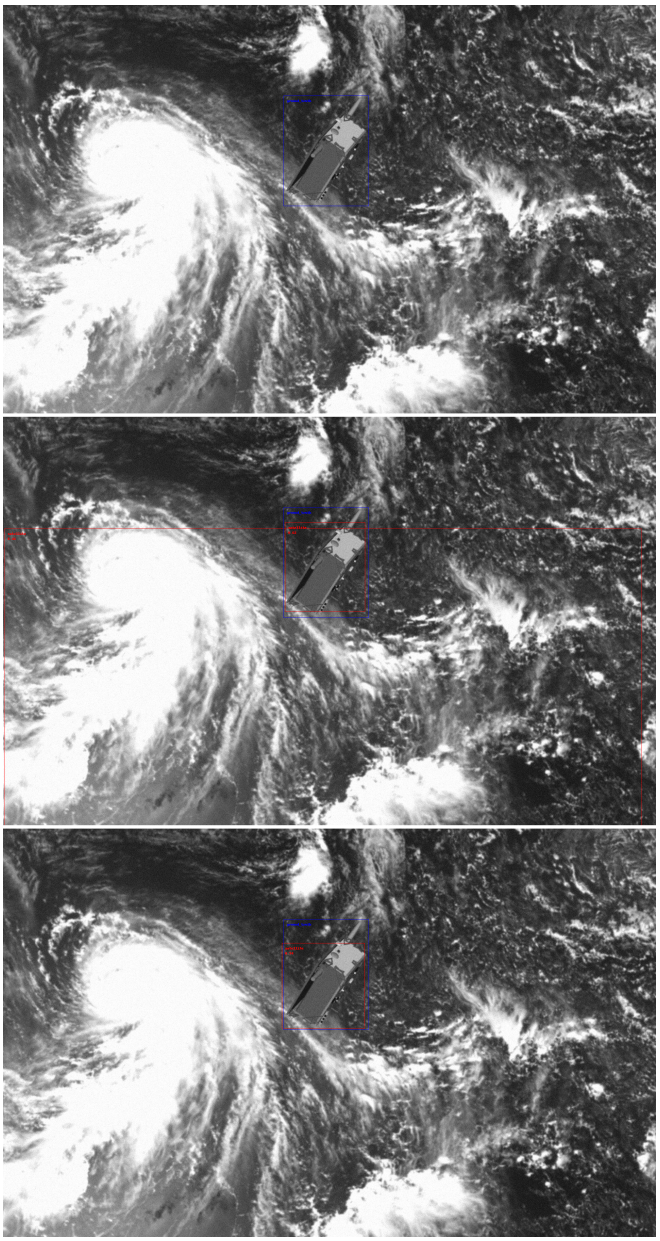

Fig. 4: Example of detection results; ground truth box is outlined in blue and prediction is red. Top: PTQ `MobileDet`, middle: QAT `MobileDet`, bottom: QAT `SpaghettiNet`.

B. Inference Time

Inference times includes time taken to load the image into RAM in addition to making bounding box predictions. The inference times presented in Table III are averaged over the 2,400 images of the test set, each image is of size 1920 x 1200 pixels. Inference times were collected on the Coral Dev Board Mini. Even though both model architectures were optimised for the Edge TPU, the uncompiled 8-bit quantised models were run and timed on the CPU of the Dev Board Mini (Quad-core ARM Cortex-A35) for completeness.

The ARM CPU represents an alternative low-power edge computing device that might be present on-board a satellite.

TABLE III: Inference times for detections.

Model Architecture	Inference Time on ARM CPU (ms)		Inference Time on Edge TPU (ms)	
	PTQ	**QAT**	**PTQ**	**QAT**
`MobileDet-EdgeTPU`	936	939	33.7	33.0
`SpaghettiNet-S`	-	654	-	37.2

The difference in inference time between PTQ and QAT `MobileDet` is negligible on both processors, this indicates there is no strong relationship between quantisation method and inference time. Average inference time on `SpaghettiNet` was measured to be slightly longer than on `MobileDet` using the Edge TPU, which conflicts with the official results provided by TensorFlow [17]. This may be due to the difference in hardware setup for the official benchmarks; `MobileDet-EdgeTPU` was evaluated on Google Pixel 4, while `SpaghettiNet` variations were evaluated on Google Pixel 6 as noted in Table I. The two phones have different processor designs, and it would be inappropriate to draw comparisons unless both models were assessed on the same hardware.

C. Power Consumption

Edge TPU power consumption during inference time was monitored using a power metre over the USB-C power connection to the Dev Board Mini. For reference, the single-board computer consumes about 0.5 watts when idle. The power consumption of the Edge TPU during inference for different models are presented in Table IV.

TABLE IV: Comparison of Edge TPU power consumption.

Model Architecture	Power Consumption (watts)	
	PTQ	**QAT**
`MobileDet-EdgeTPU`	0.50	0.50
`SpaghettiNet-S`	-	0.30

According to official benchmarks [22], `SpaghettiNet` consumed less than 70% of the energy used by `MobileDet` on the COCO object detection task. This is corroborated by the results of our evaluation. Power consumption was the same for the PTQ and QAT `MobileDet` models, which suggest that the quantisation method does not affect power consumption.

D. File Size

As model weights are compressed from 32-bits floating-point numbers to 8-bit integers, the model file size is also reduced. Table V summarises the file size reduction for both model architectures.

The QAT float model is slightly larger than the baseline non-QAT float model as the QAT model has additional fake quantisation nodes. 8-bit PTQ models are slightly larger than 8-bit QAT models because PTQ models have "dequantise" nodes after compilation to Edge TPU, as the PTQ model's post-processing operators expect floating-point values. On average, the file size after compilation is about one-third of the original model. Quantised models have smaller sizes to better fit on memory-limited devices. The Edge TPU has an 8MB SRAM to fit the model's inference executable and model parameters; storing parameter data in Edge TPU RAM results in quicker inference speeds compared to retrieving the parameter data from an external memory source, so small models are more desirable for inference on Edge TPU.

TABLE V: Effect of quantisation on file size.

Model Architecture	File Size (MB)			
	Baseline (Float)	**QAT (Float)**	**PTQ (8-bit)**	**QAT (8-bit)**
`MobileDet-EdgeTPU`	13.7	14.0	4.71	4.48
`SpaghettiNet-S`	-	13.9	-	4.70

E. Discussion

For the object detection task, the `MobileDet` architecture produced a smaller, faster model for the Edge TPU, while the `SpaghettiNet` model was more accurate and consumed less power. Quantisation-aware trained models lead to better detection accuracy, but requires a sufficient amount of training data and training time for the network to adapt to the low precision arithmetic. Post-training quantisation results in more noticeable accuracy loss, however only requires a small representative dataset to calibrate the model.

Based on these findings, the first decision point for on-board AI deployment should be data availability. For a new task where supervision data is available, the most fruitful approach is to fine-tune a quantisation-aware network using `MobileDet` if latency is mission critical, or `SpaghettiNet` to prioritise detection accuracy and power efficiency. Post-training quantisation might be the last resort when no supervision data is available but a small representative dataset exists for a task that the network is already trained for. It should be noted that PTQ does not fine-tune the network for the new task, but only learns the quantisation parameters based on the distribution of the input samples.

V. Conclusion

This paper looks at the practical considerations in fine-tuning an object detection model for use on-board a satellite. To cater for the Size, Weight and Power (SWaP) constraints on-board satellites, the use of edge accelerators has been explored. We have compared two quantisation techniques for two network architectures, and presented how they affected model accuracy, inference time, power consumption, and file size for the task of in-orbit object detection. The results demonstrate that quantisation-aware training is a must to ensure model performance on the TPU accelerator, providing performance comparable to full precision model running on a general purpose GPU. Furthermore, network architecture design leads to variations in power consumption and latency, which might be dictate the choice of one over the other depending on the power budget available on-board.

REFERENCES

[1] D. Selva and D. Krejci, "A survey and assessment of the capabilities of Cubesats for Earth observation," *Acta Astronautica*, vol. 74, pp. 50–68, May 2012.

[2] mars.nasa.gov, "Learn About the Rover - NASA." [Online]. Available: https://mars.nasa.gov/mars2020/spacecraft/rover/

[3] R. Gonzalez, D. Apostolopoulos, and K. Iagnemma, "Slippage and immobilization detection for planetary exploration rovers via machine learning and proprioceptive sensing," *Journal of Field Robotics*, vol. 35, no. 2, pp. 231–247, 2018.

[4] K. Otsu, G. Matheron, S. Ghosh, O. Toupet, and M. Ono, "Fast approximate clearance evaluation for rovers with articulated suspension systems," *Journal of Field Robotics*, vol. 37, no. 5, pp. 768–785, 2020.

[5] A. Lopez Arreguin, S. Montenegro, and E. Dilger, "Towards in-situ characterization of regolith strength by inverse terramechanics and machine learning: A survey and applications to planetary rovers," *Planetary and Space Science*, vol. 204, p. 105271, Sept. 2021.

[6] Google Cloud, "Edge TPU." [Online]. Available: https://cloud.google.com/edge-tpu

[7] J. Goodwill, G. Crum, J. Mackinnon, C. Brewer, M. Monaghan, T. Wise, and C. Wilson, "NASA SpaceCube Edge TPU SmallSat Card for Autonomous Operations and Onboard Science-Data Analysis," Virtual, Aug. 2021, nTRS Author Affiliations: Goddard Space Flight Center, GRC LERCIP NTRS Report/Patent Number: SSC21-VII-08 NTRS Document ID: 20210019764 NTRS Research Center: Goddard Space Flight Center (GSFC). [Online]. Available: https://ntrs.nasa.gov/citations/20210019764

[8] G. Furano, A. Tavoularis, and M. Rovatti, "AI in space: applications examples and challenges," in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct. 2020, pp. 1–6, iSSN: 2377-7966.

[9] G. Furano, G. Meoni, A. Dunne, D. Moloney, V. Ferlet-Cavrois, A. Tavoularis, J. Byrne, L. Buckley, M. Psarakis, K.-O. Voss, and L. Fanucci, "Towards the Use of Artificial Intelligence on the Edge in Space Systems: Challenges and Opportunities," *IEEE Aerospace and Electronic Systems Magazine*, vol. 35, no. 12, pp. 44–56, Dec. 2020, conference Name: IEEE Aerospace and Electronic Systems Magazine.

[10] D. Mrozek, R. Grny, A. Wachowicz, and B. Małysiak-Mrozek, "Edge-Based Detection of Varroosis in Beehives with IoT Devices with Embedded and TPU-Accelerated Machine Learning," *Applied Sciences*, vol. 11, no. 22, p. 11078, Jan. 2021, number: 22 Publisher: Multidisciplinary Digital Publishing Institute.

[11] E. Assunção, P. D. Gaspar, K. Alibabaei, M. P. Simões, H. Proença, V. N. G. J. Soares, and J. M. L. P. Caldeira, "Real-Time Image Detection for Edge Devices: A Peach Fruit Detection Application," *Future Internet*, vol. 14, no. 11, p. 323, Nov. 2022, number: 11 Publisher: Multidisciplinary Digital Publishing Institute.

[12] A. Lotti, D. Modenini, P. Tortora, M. Saponara, and M. A. Perino, "Deep Learning for Real Time Satellite Pose Estimation on Low Power Edge TPU," June 2022, arXiv:2204.03296 [cs].

[13] R. Bayer, J. Priest, and P. Tözün, "Reaching the edge of the edge: Image analysis in space," 2023.

[14] NASA, "ARES | Orbital Debris Program Office | Radar Measurements." [Online]. Available: https://www.orbitaldebris.jsc.nasa.gov/measurements/radar.html

[15] ——, "ARES | Orbital Debris Program Office | Optical Measurements." [Online]. Available: https://www.orbitaldebris.jsc.nasa.gov/measurements/optical.html

[16] ——, "ARES | Orbital Debris Program Office | Debris Modeling." [Online]. Available: https://www.orbitaldebris.jsc.nasa.gov/modeling/

[17] TensorFlow, "TensorFlow 1 Detection Model Zoo." [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md

[18] Y. Xiong, H. Liu, S. Gupta, B. Akin, G. Bender, Y. Wang, P.-J. Kindermans, M. Tan, V. Singh, and B. Chen, "MobileDets: Searching for Object Detection Architectures for Mobile Accelerators," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 3824–3833, iSSN: 2575-7075.

[19] S. Sharma, T. H. Park, and S. D'Amico, "Spacecraft Pose Estimation Dataset (SPEED)," 2022. [Online]. Available: https://purl.stanford.edu/dz692fn7184

[20] B. Chen, J. Cao, A. Parra, and T.-J. Chin, "Satellite Pose Estimation with Deep Landmark Regression and Nonlinear Pose Refinement," Aug. 2019, arXiv:1908.11542 [cs].

[21] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common Objects in Context," Feb. 2015, arXiv:1405.0312 [cs]. [Online]. Available: http://arxiv.org/abs/1405.0312

[22] S. Gupta and M. White, "Improved On-Device ML on Pixel 6, with Neural Architecture Search," Nov 2021. [Online]. Available: https://blog.research.google/2021/11/improved-on-device-ml-on-pixel-6-with.html