Reaching the Edge of the Edge: Image Analysis in Space

Robert Bayer
roba@itu.dk
IT University of Copenhagen

Julian Priest
jucp@itu.dk
IT University of Copenhagen

Pınar Tözün
pito@itu.dk
IT University of Copenhagen

ABSTRACT

Satellites have become more widely available due to the reduction in size and cost of their components. As a result, there has been an advent of smaller organizations having the ability to deploy satellites with a variety of data-intensive applications to run on them. One popular application is image analysis to detect, for example, land, ice, clouds, etc. for Earth observation. However, the resource-constrained nature of the devices deployed in satellites creates additional challenges for this resource-intensive application.

In this paper, we present our work and lessons-learned on building an Image Processing Unit (IPU) for this satellite. We first highlight the resource constraints based on a deployed satellite performing machine learning on satellite imagery in orbit, including the required latency, power budget, and the network bandwidth limitations driving the need for such a solution. We then investigate the performance of a variety of edge devices (comparing CPU, GPU, TPU, and VPU) for deep-learning-based image processing on satellites. Our goal is to identify devices that are flexible when the workload changes while satisfying the power and latency constraints of satellites. Our results demonstrate that hardware accelerators such as ASICs and GPUs are essential for meeting the latency requirements. However, state-of-the-art edge devices with GPUs may draw too much power for deployment on a satellite.

CCS CONCEPTS

• **Hardware → Emerging technologies**; • **Computer systems organization → Real-time systems**; • **Computing methodologies → Machine learning**.

KEYWORDS

earth observation, image data analysis, satellites, internet of things, edge computing, machine learning

ACM Reference Format:

Robert Bayer, Julian Priest, and Pınar Tözün. 2024. Reaching the Edge of the Edge: Image Analysis in Space. In *Workshop on Data Management for End-to-End Machine Learning (DEEM 24), June 9, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3650203.3663330

1 INTRODUCTION

Most innovations in real-world satellite applications used to be only available to the largest countries such as the USA and Russia. These innovations led to significant reductions in the size of a satellite's components and the cost of the manufacturing and deployment process of a satellite. This, in turn, introduced a new CubeSat class of small satellites. Their format is based on a 10cm cube, with the possibility of combining multiple of these to create a larger satellite. This standardization [12] makes a batch deployment of satellites easier as the format affords tight configuration. The reduction in costs that came as a result of this new deployment method led to the advent of satellites owned by small or private organizations.

On the other hand, the size of the CubeSat class satellites poses new complexities, such as the power and thermal constraints and the physical dimensions of components. Therefore, space Internet-of-Things (IoT) applications pose additional challenges compared to terrestrial edge IoT applications. The edge devices deployed on satellites must perform very resource-intensive workloads with high reliability and do so in a highly resource-constrained environment.

Image processing and analysis is one class of possible satellite workloads, especially for Earth observation [2]. Satellites with this type of workload take large-scale images, which they have to store and send back to a ground station. The link between the satellite and a ground station is of limited bandwidth and short-lived. The images must be highly compressed to send all of them or filtered by quality and areas of interest. The images already have a resolution of tens or hundreds of meters per pixel, and lossy compression would lead to even more loss of detail. Filtering by areas of interest preserves the details of the images but is more resource-intensive.

Performing data compression or filtering operations on a satellite before the data is sent to Earth is essentially an initial step, a type of data pre-processing task, for a longer end-to-end image analysis pipeline for data-intensive Earth observation applications. Enabling such data processing on a satellite can lead to smaller satellite sizes, while still delivering data of higher value, i.e. image resolution and relevance. This paper presents in detail what it takes to perform deep-learning-based image processing on small satellites based on our experiences in the DISCO [17] project. DISCO offers university students the opportunity to design and operate a small satellite and to gain space flight experience. It is a collaboration between four universities in Denmark and aims to launch three student CubeSats into Low Earth Orbit. The first launch, including the IPU described in this work, happened in April 2023 [44].

The contributions of this paper are:

- We illustrate the network bandwidth limitations of small satellites in Section 2 by quantifying the maximum number of images these satellites can transmit back to Earth, stressing the need for image filtering and compression to increase the value of the received data.
- We characterize the requirements and constraints of performing satellite imagery analysis on a CubeSat-based satellite in orbit in Section 2. We do this by outlining multiple scenarios based on deep-learning-based image filtering on satellites and showing

DEEM 24, June 9, 2024, Santiago, AA, Chile

ACM ISBN 979-8-4007-0611-0/24/06
https://doi.org/10.1145/3650203.3663330

how each scenario affects the required latency, storage, and power draw of such a system.
- Using the identified scenarios, we investigate the suitability of several edge devices for the IPU on the satellite in Section 3. The devices are based on a variety of hardware architectures, ranging from a microcontroller based on an ARM CPU to system-on-chip (SoC) devices offering more parallelism by leveraging the power of hardware accelerators. Our results demonstrate not only the benefit of using highly specialized hardware but also the need to strike a balance across flexibility, energy efficiency, and image processing latency for the space edge IoT use case.

2 REQUIREMENTS

The requirements and constraints of the system are based on the DISCO's Arctic imaging mission use case. The power and mass constraints of the Image Processing Unit (IPU) are based on the budget and engineering design. These are typical values for a modular 3U Earth-imaging satellite. One third of the satellite (1U) is dedicated for the *payload* to house the IPU as well as the module enclosure, cameras, and corresponding optics. The mass of the whole payload cannot exceed 1.3kg. Table 1 lists the IPU's physical constraints.

Nominal power	Peak power	Mass	Dimensions
2.00 W	5.00 W	0.15 kg	10x70x80 mm

Table 1: List of constraints of the Image Processing Unit (IPU).

In addition to the physical constraints, the IPU should meet soft requirements, such as short time-to-boot, robustness against crashes and failures, and preferably flight heritage. Furthermore, there are use-case-related requirements, such as latency, which we base on a single streaming camera setup that can be extrapolated to multi-camera setups. The IPU must also have low heat dissipation as it has to rely purely on radiation for cooling.

The planned orbit of DISCO is a sun-synchronous polar orbit at 550 km altitude. The camera sensor is based on Alvium 1800 C-2040 with an image resolution of 4, 512 x 4, 512 pixels, and the lens focal length is the maximum that could be considered for the mission. Assuming a 50% overlap[1], i.e., capturing the same land area, we derive a 4.42s latency between consecutive images (T_i) as follows:

$$T_i = \frac{GSD \cdot H_i \cdot \cap_i}{v} = \frac{14.8495m/px \cdot 4,512px \cdot 0.5}{7,585.16m/s} = 4.42s \quad (1)$$

, where GSD (ground sample distance) is the spatial resolution of an image, H_i is the height of the image in pixels, $\cap_i$ is the overlap between images, and v is the satellite's orbital velocity.

Given this latency, we next describe three image processing scenarios with varying levels of difficulty based on the Arctic mission's goals. For multi-camera setups, the substitution of other cameras' GSD and H_i would determine the latencies for each of them. The overall latency requirement for the IPU would then depend on whether the cameras are used simultaneously or not.

Scenario 1: Real-time imaging. Images are taken with a period of 4.42s, as derived in Equation 1. The inference has to be completed before the next image is captured. This allows for the results of an inference to be used in decision-making for the next image capture.

Scenario 2: Arctic region imaging. Images are taken only while the satellite is over the polar regions relaxing the latency requirements. Images are buffered, and inference is performed in the remainder of the orbit, with the IPU available for inference before the subsequent orbit. This cycle completes in 5, 739s. Given the camera's spatial resolution, this amounts to the capture of 80 images and 71.74s per image.

Scenario 3: Greenland imaging. Images are taken only while over Greenland, further relaxing the latency requirements. Once a day, there are four consecutive passes over Greenland. Each pass takes 80 images (N_i), which can be derived as follows:

$$N_i = \frac{H_G}{GSD \cdot H_i \cdot \cap_i} = \frac{2,670,000m}{14.8495m/px \cdot 4,512px \cdot 0.5} = 79.7 \quad (2)$$

, where H_G is the distance between the northernmost and the southernmost points of Greenland. As a result, $80 \cdot 4 = 320$ images have to be buffered for performing inference on them per day. This yields the maximum per-image inference latency of 270s. While this dramatically decreases the compute requirements, it increases the requirements for the storage capacity and I/O, as the images have to be first buffered to storage.

Space to Earth bandwidth impact. Given the camera's image resolution, we can calculate the theoretical maximum number of images we can transfer back to Earth in a day:

$$T_i(b) = \frac{b \cdot N_p \cdot \overline{t_p}}{H_i \cdot W_i \cdot C_i \cdot 8} \quad (3)$$

, where C_i is number of channels in the image, three in the case of RGB cameras, such as the one used on DISCO satellites. N_p corresponds to the number of passes a day above a ground station and $\overline{t_p}$ is the average length of a pass in seconds. While these numbers vary daily, we can use an estimate of 4 passes with 10 minutes of usable average duration. This total number of images is dependent on the network bandwidth between the satellite and the ground station (b). In the case of DISCO1, only a slower UHF connection with up to 9.6 Kbps bandwidth is available. Finally, H_i and W_i are the height and width of the image in pixels, respectively, and each pixel has an 8-byte representation. This results in:

$$T_i(9,600) = \frac{9,600 \cdot 4 \cdot 600}{4,512 \cdot 4,512 \cdot 3 \cdot 8} = 0.05 \quad (4)$$

The next satellite launch, DISCO2, is expected to have the significantly faster S-band connection that achieves 10 Mbps between the satellite and the Earth. With that, the total number of images that can be transferred in a day increases to 49.12 based on Equation 4.

However, even with this increase, one would need to discard 38.6% of images in the Greenland imaging scenario, which is the least demanding among the three.

Two other factors can increase the number of images sent in a day significantly, which this equation does not account for. First, the equation only calculates this number for a single ground station. The number of the total images transferred would scale linearly with the increase in the number of ground stations. Second, the load on the network is calculated based on the raw, uncompressed, size of the images. The size of the images can be significantly reduced

[1]A conservative estimate by the domain experts in the project.

using compression. While lossy compression could deliver a higher reduction in the size, lossless compression is preferable in order to preserve details of the imagery for further scientific purposes.

Nevertheless, these numbers highlight the need for filtering out irrelevant images or parts of images on the satellite before sending them to Earth. Even the simplest case of discarding cloud-covered images can save ~ 67% of the bandwidth [34]. Cloud discrimination can be combined with other objectives closely aligned with the mission, which can increase the bandwidth use efficiency even further. Therefore, investigating the most suitable computing platform to deploy on the IPU to accomplish such a filtering task is essential.

3 CHOICE OF THE IPU

With the requirements for the IPU identified in Section 2, this section investigates the suitability of a list of edge devices as a choice for the IPU. The list of devices represents different processing types and capabilities at the edge as described in Section 3.1. As a result, there is no one size fits all when it comes to the software setup required for the target use case on these devices. To ensure that we utilize them in the best way possible for the target workload, we first analyze the performance trade-offs and settle on the most optimized setup for each of them in Section 3.2, before we compare the performance of all devices with respect to each other in Section 3.3. Finally, Section 3.4 gives a summary of our investigation.

3.1 Methodology and Setup

3.1.1 Devices under Test. We conduct experiments on seven devices representing different hardware architectures, from general-purpose to ASICs. Table 2 lists the specifications. In addition to the variety of hardware architectures to represent, the devices are chosen based on their size and mass while taking into account the need for high performance and the highly limited power budget.

ARM Cortex-M7 Microcontroller [28] is the most general-purpose device on our list since it only has ARM CPUs. It delivers the lowest performance due to its low specialization. It, however, has a low power consumption, which is an important factor in the choice of the IPU. Furthermore, this chip has an extensive flight history and would, therefore, be a safe choice for deployment.

To perform machine learning inference using this microcontroller (MCU), we leverage the X-CUBE-AI tool [58] to generate an optimized inference library for STM32 chips.

On this device, we emulate the performance and power characteristics of a flight computer available for potential deployment in the DISCO project (OBC-P3) using the Cortex-M7 core of the *STM32H745* chip with clock frequency scaled down to 300MHz.

NVIDIA Jetson Nano [43] is designed for embedded applications built around a power-efficient SoC composed of an ARM CPU and an NVIDIA GPU. GPUs are the most common choice for the acceleration of machine learning thanks to the high degree of parallelism they offer. This device is not only specialized for deep learning but can also be leveraged to accelerate other parallel tasks, such as image pre-processing pipelines (cropping, resizing, etc.). This flexibility, while allowing a more general-purpose use, can have increased power consumption as one of its side effects.

NVIDIA Jetson Nano can operate at up to 10W and was configured to operate at 5W by disabling two of the four cores of the CPU to fit within the power budget of the satellite (see Table 1). This, however, does not affect the GPU due to the lack of finer control over its power, which is only present in the newer architectures.

This device runs Ubuntu 18.04 and can leverage frameworks such as TensorFlow [1], PyTorch [45], or TensorRT [56].

Intel Neural Compute Stick 2 (NCS2) [26] is an ASIC based on the Intel Movidius Myriad X Vision Processing Unit (VPU), which is designed to accelerate convolutional neural network inference. This accelerator needs to be coupled with a host device, which in the case of this study is Raspberry Pi 3 [46].

The deployment of neural network inference on NCS2 is facilitated by the Intel OpenVino framework [27], which can convert the neural networks trained using a majority of popular deep learning frameworks into its intermediate representation. OpenVino further optimizes these models and compiles them for running on the NCS2. These optimizations include layer fusion and quantization to 16-bit floating point values, the only data type supported by NCS2.

Toradex Verdin iMX8M Plus [53] is based on the NXP iMX8M Plus chip, which uses Neural Processing Unit (NPU) for acceleration of machine learning workloads. Similarly to the GPU on the NVIDIA Jetson Nano, the NPU of the NXP iMX8M Plus chip is found on the same die as the CPU, used as the host of the accelerator, allowing the accelerator to take advantage of the fast interconnect within the chip when communicating between the CPU and GPU.

The deployment of machine learning workloads on this device is facilitated by TensorFlow Lite and a custom VX Delegate, which facilitates offloading of the operators onto the NPU, supporting 16-bit floating point and 8-bit integer operations. This device runs custom stipped-down Yocto Linux.

CoralAI TPU [14] is developed by Google to accelerate machine learning inference at the edge. Similar to NCS2, it is not standalone and needs to be coupled with a host. The TPU compiler [15] accepts neural networks in the TensorFlow Lite [23] format, quantized to 8-bit integers, the device's only supported data type. We evaluate this device in three form factors:

(1) *Coral AI Dev Board Micro*, an embedded device coupling the CoralAI TPU with two low-power ARM cores running FreeRTOS [18].
(2) *Coral AI Dev Board Mini*, which combines the CoralAI TPU with a more capable quad-core CPU at the cost of higher power consumption.
(3) *Coral AI USB Accelerator*, hosted on the Raspberry Pi 3, which provides a fair comparison with the NCS2.

3.1.2 Metrics. The metrics used for the performance characterization of the devices are based on the requirements Section 2 outlines.

Latency is the time-to-inference of an image of size 4512 x 4512 pixels in seconds. It measures the case where the image is already in the device's memory and disregards the latency of image capture, network overhead, setup time, and the model initialization, which will all get amortized over time in a real-world setting.

Nominal power draw is the average power draw of the device multiplied by the device's *duty cycle*, which is the ratio between the achieved and required latency for each scenario outlined in Section 2. This metric is reported in megawatts (mW).

Peak power draw is the maximum power draw, measured in mW, a device reaches during the inference.

	OBC-P3	Jetson Nano	RPi 3 + NCS2	Toradex Verdin iMX8MP	CoralAI Dev Board Micro	CoralAI Dev Board Mini	RPi 3 + CoralAI USB accel.
CPU	ARM Cortex-M7 @ 300MHz	Quad-core ARM Cortex-A57 @ 1.43 GHz	BCM2837 @ 1.2 GHz	Quad-core ARM Cortex-A53 @ 1.8 GHz, ARM Cortex-M7 @ 800 MHz	ARM Cortex-M7 @ 800 MHz, ARM Cortex-M4 @ 400 MHz	Quad-core ARM Cortex-A35 @ 1.5 GHz	BCM2837 @ 1.2 GHz
RAM	384KB SRAM	4GB	1 GB	4GB	64 MB	2GB	1 GB
Storage	2MB Flash, 64GB eMMC	64 GB SD card	32GB eMMC	32 GB SD card	128MB NAND	8GB eMMC	32 GB SD card
Accelerator	-	128-core Maxwell GPU	Intel NCS2	NPU (2.25 TOPS)	CoralAI Edge TPU (4 TOPS)	CoralAI Edge TPU (4 TOPS)	Coral AI USB accel. (4 TOPS)
Dimensions	94 x 94 x 13 mm	100 x 80 x 29 mm	85 x 56 x 20 mm (72.5 x 27 x 14 mm)	120 x 120 x 19 mm	65 x 30 x 6.8 mm	64 x 48 x 14.6 mm	85 x 56 x 20 mm (65 x 30 x 8 mm)
Mass	120 g	141 g	42 g (53.4 g)	181.8 g	10.4 g	25.5 g	42 g (19.2 g)

Table 2: Specifications of the devices. The dimension and mass in parenthesis shows data for the accelerators attached externally.

Power consumption is reported in megawatt-hour (mWh) based on Equation 5, where E is power consumption, $\bar{P}$ is the average power draw in mW, and t is the latency in seconds of inference of a single full-size image. This metric will guide the choice of a more energy-efficient device if multiple devices fulfill the requirements for the use cases.

$$E = \bar{P}(t/3,600) \tag{5}$$

The power draw of the devices was measured using Otii Arc Pro [48], a precision power supply and analyzer.

3.1.3 Workload. To simulate the scenarios outlined in Section 2, we use a 5-class image classification problem aligned with one of the initial tasks the satellite will run - land cover classification. While the number of classes might change after deployment, this change only affects the last layer of the model, which in turn impacts the total number of multiply-accumulate operations insignificantly.

To match this classification problem, we chose MobileNetV1 [25] model pre-trained on the ImageNet [51] dataset and fine-tuned for 5-class classification task. The choice of the model was guided by the high availability of pretrained models as well as its small size and high flexibility in scaling down the size of the model using the *depth multiplier*, a parameter that controls the number of convolutional layers of the network. The scaling down is necessary to fit this model on the smallest of the devices under test, the ARM Cortex-M7 MCU, which only has a little over 2MB of memory.

Inference scenario. The shape of the images is the most important factor for our analysis when quantifying the *latency* and *power*-related metrics since the shape, not the content, of an image affects the computational requirements for the inference given a model. To emulate the computational requirements of the expected inference scenario, we randomly generate a 4512 x 4512 image (~60MB) following the size of the images produced by our camera (Section 2). This image is essentially a matrix with dimensions 4512 x 4512 x 3. The image is then partitioned into tiles of 224 x 224 creating 400 patches from the original image, in order to fit into the memory space of all the devices under test and the model input size.[2] We perform the inference on these patches separately. The reported latency is the latency to perform the inference on all 400 patches. The division into patches furthermore serves as a coarse-grain image segmentation in practice since it can allow us to send only the patches interesting to the mission at hand rather than the whole 4512 x 4512 image, leading to further optimization in bandwidth utilization between the satellite and the ground.

To report latency results for the devices with insufficient memory, i.e., ARM Cortex-M7 MCU and the CoralAI Dev Board Micro, we use images of total sizes of 224 x 224 (single patch) and 2272 x 2272 (100 patches) pixels, respectively. The results of the benchmarks for these subsets are extrapolated to 400 patches by multiplying the latency by 400 and 4, respectively.

Finally, after the division into patches, the inference does not include any image pre-processing, except for the image rescaling to the range of $[0, 1)$ in the case of the NVIDIA Jetson Nano and the NCS2, which operate on floating point values rather than integers.

Runs. The results are the average of ten inference runs on the full-sized images. All devices are tested on three sizes of the network (depth multiplier of 0.25, 0.5, and 1.0), with the exception of the ARM Cortex-M7 MCU, which could only fit the smallest of the models in memory.

[2]The image is not evenly divisible. Therefore, we disregard the last 32 rows and columns of the image matrix.

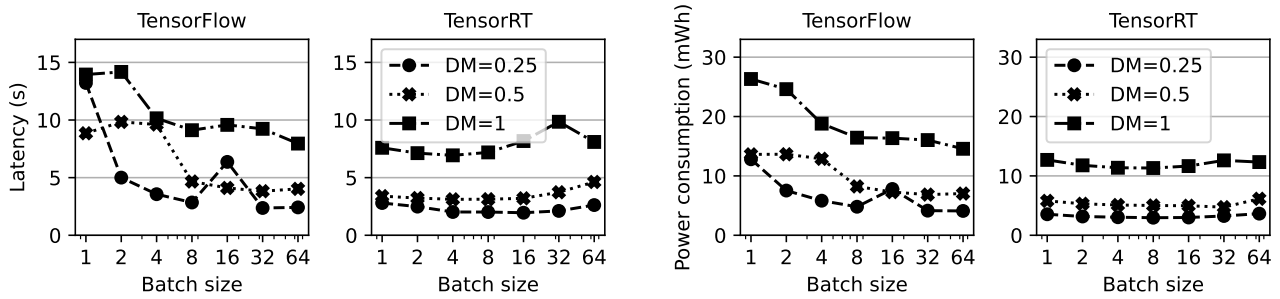

Figure 1: Inference latency (left) and power consumption (right) on full-size image using NVIDIA Jetson Nano as the batch size vary while using TensorFlow and TensorRT. DM is the depth multiplier of MobileNetV1 to scale the model size.

3.2 Optimizations on Each Device

To take full advantage of the hardware architectures at hand, we apply optimizations on each setup prior to comparing them.

ARM Cortex-M7 Microcontroller. While the MCU has support for floating point operations, we quantized the network to 8-bit precision to increase the performance and, more importantly, decrease the memory requirements of running the models because the memory size is the most limiting factor of this device. X-CUBE-AI [58] generates a library which leverages the CMSIS-NN kernels [36] that are specifically designed and optimized to run on the ARM Cortex-M class of devices, which is what we use in our experiments.

NVIDIA Jetson Nano. Modern GPUs can leverage alternative data types to the default 32-bit floating point. On NVIDIA Jetson Nano, we take advantage of the 16-bit floating point data type, which is the only alternative option on this device, leading to an increase in performance and a decrease in memory requirements.

This device is supported by TensorRT, a framework for building optimized inference engines for NVIDIA GPUs. This is also the only device in our setup that supports batching while performing inference. Therefore, we test the performance of the NVIDIA Jetson Nano using varying batch sizes and leveraging TensorFlow and TensorRT. Figure 1 clearly shows the advantage of using TensorRT.

A separate thread performs the division of the full-size image into patches, subsequent scaling and batching of these patches, and pushing the processed batches into a queue. The batches are then inferred using TensorFlow or TensorRT. The optimal batch size for inference using the TensorRT framework is 16, which achieves the best overall latency and power efficiency. The increase in latency past this batch size is due to the total number of patches not being evenly divisible by these higher batch sizes, rendering a large portion of the computations of the last batch redundant. This problem only arises with TensorRT, as the inference engines are compiled and optimized for a fixed batch size in this framework.

Given the results in Figure 1, we use TensorRT with NVIDIA Jetson Nano in Section 3.3 with batch size 16.

Intel Neural Compute Stick 2. NCS2 supports the submission of more than one inference request concurrently, with four requests being recommended by the manufacturer. This amortizes the time spent on data transfers between the NCS2 and the host, which is non-negligible. As shown in Figure 2, four concurrent requests lead to a 50%-60% lower inference latency compared to one request.

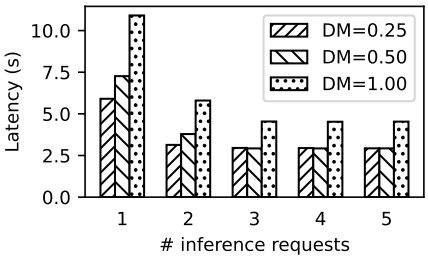

Figure 2: Impact of the number of concurrent inference requests sent to the NCS2 on the inference latency.

To saturate the device with data and, therefore, decrease the latency of inference for the entire image, we produce patches in a separate thread and push the processed images into a queue, as is the case for the NVIDIA Jetson Nano. NCS2 only supports 16-bit floating points. The scaling of the input for this is integrated into the pre-processing steps of the model through OpenVINO. To create the patches from the original image on the host CPU, we leverage the optimizations of *memcpy* to copy a full row of the patch matrix instead of copying every single value one at a time.

The main thread on Raspberry Pi polls the queue for new patches and populates the inference requests. As OpenVINO has support for asynchronous requests, the inference requests can be populated and submitted from the main thread to NCS2. The inference requests accept a callback function, triggered on completion of inference, rendering the inference request available for reuse for the inference request of the next patch.

Toradex Verdin iMX8M Plus. While the on-chip NPU supports both 16-bit floating point values and 8-bit integer values, this work uses the 8-bit integer values due to increased computational efficiency. The use of 8-bit integers also leads to omitting of the otherwise-needed scaling operation of the pre-processing pipeline, as in the case of the NVIDIA Jetson Nano and the Intel NCS2. Furthermore, the creation of the patches leverages the same *memcpy* optimization as the NCS2.

CoralAI TPU. The CoralAI TPU only supports 8-bit integers (Section 3.1.1). The network is, therefore, quantized to this precision. Unlike the NCS2, the TPU only supports synchronous requests.

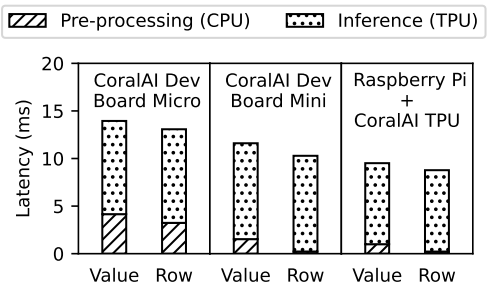

Figure 3: Latency breakdown of the inference pipeline on the CoralAI setups, measured on a single image patch using the model with a depth multiplier of 0.5. *Value* bars show the version where the patches are created by copying one value at a time and the *Row* bars show the optimized version that copies a whole row at a time using *memcpy*.

Since the precision of the network matches that of the image, there is no need to rescale the image after creating the patches. The creation of patches leverages the same *memcpy* optimization as the previous two devices. The optimized version exhibits 1.28 – 6.71x improvement in the latency of the pre-processing step (creation of the patch) of the inference pipeline as Figure 3 demonstrates.

The creation of the patches on the CoralAI Dev Board Mini now accounts for only 91.2 ms of the 4118 ms of overall inference latency of the whole image, which is a small overhead. Therefore, this pre-processing operation is run in the main thread as well since the overhead of thread communication when we add a separate thread may give diminishing returns.

3.3 Comparison of Devices

Figure 4 shows the latency and the power consumption of inference using models with increasing size, i.e., with increased depth multiplier (DM) value. Together with the nominal power draw in Figure 5 and peak power draw in Figure 6, Figure 4 guides our analysis of the suitability of these devices for use in the imaging scenarios Section 2 outlines. After the analysis with respect to each scenario, we show the impact of the depth multiplier parameter of the model and discuss its implications for storage requirements.

Scenario 1: Real-time imaging. This is the most compute-intensive use case among the three scenarios. The low latency requirement also leads to a high duty cycle percentage, causing the nominal power draw to approach the peak power draw of the devices that can fulfill the latency requirements of this scenario.

As mentioned in Section 2, this scenario has to perform inference of the full-size image in 4.42s. The ARM Cortex-M7 MCU and the CoralAI Dev Board Micro are the only devices that cannot fulfill the latency requirements of this scenario for any model size in Figure 4. This is caused by the lack of computing power of the CPU in these devices. Even though the CoralAI Dev Board Micro shares the same accelerator with the CoralAI Dev Board Mini and the Coral AI TPU USB accelerator, its CPU is unable to match the feed rate of the larger devices, underutilizing its TPU. Overall, the Toradex Verdin device with the on-die NPU achieves the best latency due to the high degree of specialization and the fast interconnect between its CPU and NPU. While the latency achieved by NVIDIA Jetson Nano comes the closest to the latency achieved by the Toradex Verdin in the two smaller models, the inference latency using the largest model does not scale well. In comparison to the rest, the devices utilizing the Coral Edge TPU exhibit a lower increase in latency with the increase in model size (i.e., DM). This is due to the high overhead of the communication between the host CPU and the TPU overshadowing the impact of the model sizes.

As mentioned earlier, the high active duty cycle leads to a high nominal power draw, causing multiple devices to exceed the power budget, highlighted in Figure 5. Only the CoralAI Dev Board Mini and the CoralAI USB accelerator fulfill the nominal power draw requirement for the smaller model configurations, while the Toradex Verdin is the only device to fulfill the nominal power draw requirement across all tested model configurations.

Scenario 2: Arctic region imaging. By relaxing the latency constraints by processing the images of only the areas above the Arctic Polar Circle, more configurations pass the requirements.

As Figure 4 shows, the relaxation of the compute requirements leads to all the devices except for the ARM Cortex-M7 MCU to fulfill the latency requirements, which is 71.74s per image for this scenario (Section 2). On the other hand, the power consumption or efficiency at which these devices can perform the inference varies (Figure 4). The lowest power consumption was measured on the Toradex Verdin for the two smaller model configurations, while the flat latency scaling of the Coral Edge TPU devices led to the highest efficiency of the largest model configuration on the two Coral Dev Boards. The highest power consumption was measured on the NVIDIA Jetson Nano and the Raspberry Pi using the NCS2 accelerator, depending on the model size configuration.

The power consumption of the Toradex Verdin is 73 – 78% and 53 – 75% lower than the power consumption of the NVIDIA Jetson Nano and the Raspberry Pi with the NCS2 accelerator, respectively. For the CoralAI Dev Boards this difference is 26 – 84% and 30 – 67%.

As all the devices, except for ARM MCU, either meet or are close to meeting the latency requirements of Scenario 1, the active portion of the duty cycle in the devices decreases significantly, causing the nominal power draw to follow in Figure 5. All devices, therefore, fulfill the nominal power draw requirements for all model sizes.

Scenario 3: Greenland imaging. The least compute-demanding scenario corresponds to taking images of only Greenland. The low constraints mean that all of the devices pass the requirements. This is the only scenario that can also be performed using the low-power ARM Cortex-M7 since the latency requirement is 270s per image (Section 2). Even though ARM MCU has a very low power draw compared to the rest of the devices, the long latency to finish an inference makes it the least efficient choice, having a power consumption 3.67 – 9.31x higher than its counterparts (Figure 4).

The majority of time in this scenario is spent idle, and therefore, the nominal power draw of the devices is minimal.

With respect to the peak power draw, Figure 6 shows that three of the configurations do not meet the requirement of less than 5W: (1) NVIDIA Jetson Nano, whose power draw significantly exceeds the satellite's power budget in all model configurations, (2) the Raspberry Pi with NCS2, which fits within the power budget of the satellite only when using the smallest model, and (3) the Toradex

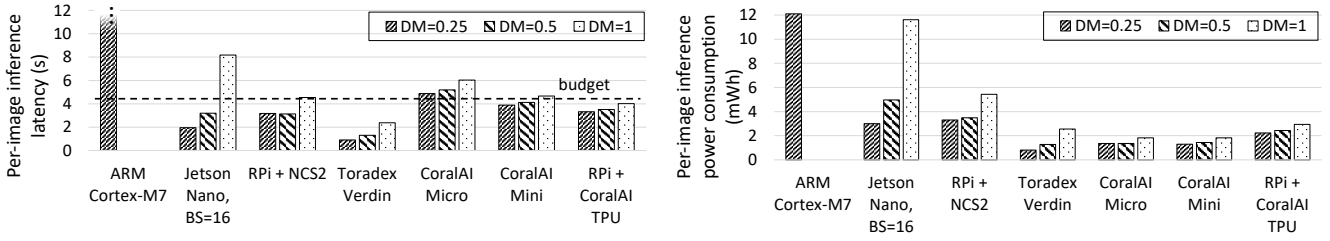

Figure 4: Inference latency (left) and power consumption (right) on full-size image (400 patches) with different scaling factors (DM) of MobileNetV1 on each device. Batch size (BS) is 1 unless stated otherwise.

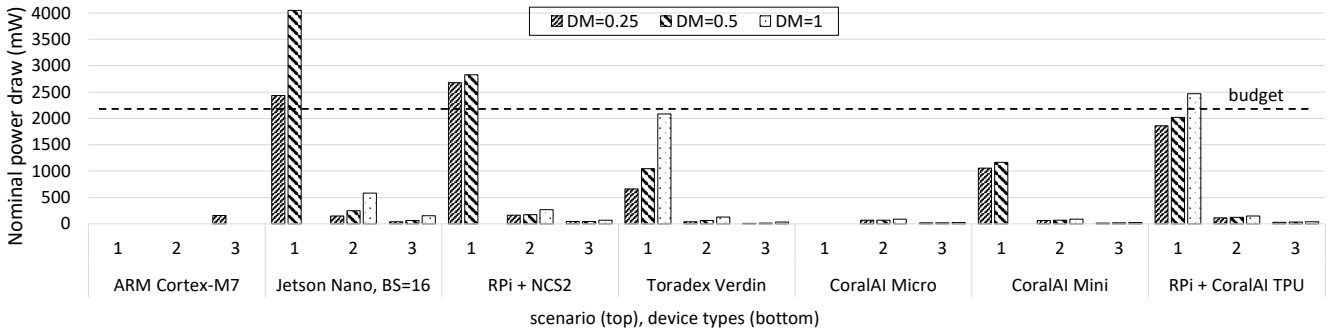

Figure 5: Nominal power draw of each device with the MobileNetV1 model of different sizes (DM) for each scenario. The combinations of the devices and model sizes that do not fulfil the latency requirements are omitted as their active duty cycle is greater than 100%. The *budget* line marks the maximum nominal power allowed for the IPU as reported in Table 1.

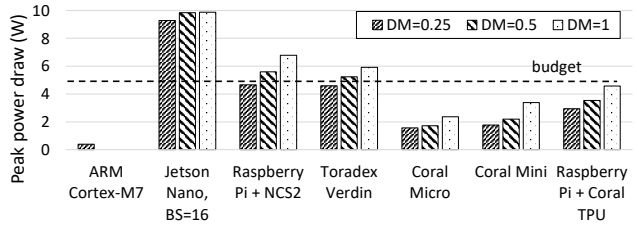

Figure 6: Peak power draw of each device with MobileNetV1 of different sizes (DM). Batch size (BS) is 1 unless stated otherwise. The *budget* line indicates the maximum peak power allowed for the IPU as reported in Table 1.

Verdin, which also fulfills this requirement only using the smallest model configuration. Even though decreasing the batch size of the NVIDIA Jetson Nano decreases its peak power draw by up to 25.8%, it does not lead to reductions significant enough to make this device eligible for deployment on DISCO.

Effects of depth multiplier. Our goal in this work is to characterize the performance of a variety of the edge devices for the DISCO use cases rather than coming up with the most accurate model for such a use case. The size of the model affects not only the accuracy of the model but also, as was shown in this section, latency and power draw of inference. Furthermore, the increased size of the model also increases the storage and memory requirements of the device and also stresses the uplink bandwidth (ground to satellite), which is often lower than the downlink bandwidth (satellite to ground).

MobileNetV1 with a depth multiplier (DM) of 0.25, 0.5, and 1.0 have 219,829, 832,101, and 3,233,989 parameters, respectively. This leads to file sizes of roughly 0.393KB, 1.1MB and 3.6MB, respectively, for the model itself, while these sizes change from device to device based on the deployment toolchain. These sizes can also significantly impact the time to update the model after deployment.

Based on our analysis, the model with DM=0.5 provides a middle ground between the lower latency and accuracy of the smaller model with DM=0.25 and the higher latency, power draw, and similar accuracy of the larger model with DM=1.0.

Effects of quantization. Some devices require or benefit from quantization to 16-bit or 8-bit values. In our experiments, the impact of this quantization on accuracy was negligible. Such impact can be mitigated even further with a quantization-aware training [29].

Storage. Even though the benchmarking results do not consider storage capacity and the I/O speed, they are very important, mainly for scenarios 2 and 3. In order to increase the required latency compared to the real-time scenario of 4.42 seconds, captured images need to be streamed to storage, to be later read and inferred. In order to buffer the 80 or 320 images for scenarios 2 and 3, respectively (Section 2), the devices have to have $80 \cdot 60MB = 4,800MB$ and $320 \cdot 60MB = 19,200MB$ of storage, respectively. This is fulfilled by all the devices, except for CoralAI Dev Board Mini (fulfills only scenario 2 requirement) and Micro, both of which have support for SD card expansion that can easily solve the storage space issue.

3.4 Discussion of Results

Based on Section 3.3, Toradex Verdin has the lowest latency and highest efficiency in the two smaller model sizes. It can, however, fulfill the peak power draw requirements of only the smallest configuration. We could only analyze this device after the launch of DISCO1. Therefore, it was not considered for integration in DISCO1, but it will be considered for the DISCO2 satellite given its efficiency.

Coral-TPU-based devices are next when it comes to satisfying the requirements for all the scenarios. Although they show higher latency than some of the other devices, such as the NVIDIA Jetson Nano or the Raspberry Pi accelerated by NCS2, they have superior efficiency and significantly lower peak power draw. Both the Coral Edge TPU and the NPU of the Toradex Verdin device owe their efficiency to the highly specialized systolic array architecture, which is built to perform fast multiply-accumulate operations in a highly parallel fashion without the need to load/store intermediate values.

At the other end of the spectrum was the ARM Cortex-M7. Due to the lack of parallelism, the device could only fulfill the latency requirements of the least challenging scenario, scenario 3. The low peak power draw did not lead to increased efficiency compared to the other devices due to the high latency. Furthermore, this device and the CoralAI Dev Board Micro have far less memory, prohibiting the whole image to fit in memory. This can be overcome with in-camera cropping and heavy use of buffering to storage, which would decrease efficiency and increase the complexity of the solution.

Finally, this work tested the performance of the systems using their development board variants. While this is a good option for fast characterization of their relative performance, it does not provide the full picture of the final product, especially with regards to their power draw, physical dimensions, and mass. These development boards are built to showcase all of their capabilities, including connectivity, most of which will not be used after integration. The removal of the unused peripherals, such as Ethernet or HDMI, will reduce the size and the power draw of the integrated devices. Therefore, minimal exceeding of the peak power draw, as is the case for the Toradex Verdin development board, is likely acceptable for the deployment on the future iterations of the DISCO satellites.

Based on our results and the timing of the satellite launch, Coral Dev Board Mini was chosen to be deployed as the IPU of DISCO1.

While this work was conducted in the context of the DISCO project, the presented results have wider applicability. Any project that strive to maximize the value of the imagery taken at challenging environments such as space and under-water while minimizing cost of deployment (e.g., small-sized setups) and operations (e.g., fewer communication channels) would benefit from our findings.

4 RELATED WORK

Influx of specialized hardware at the edge such as NVIDIA Jetsons [10], Google CoralAI [14] or Intel Neural Stick [26], coupled with increase in computing resources, make data processing at the edge viable. As a result, multiple works have investigated on-satellite data post-processing, such as image classification or segmentation, on the limited power budget such environments require.

While microcontrollers or more complex CPUs have extensive flight heritage (have already been deployed in satellites before [3, 54]) and low power footprint, they were not built with running deep learning workloads in mind. In contrast, system-on-chip (SoC)-based devices using small GPUs, such as the NVIDIA Jetsons, have been explored more in the past decade in satellites [11, 39, 52] based on the success of GPUs in terrestrial use cases of machine learning. As shown in Section 3.3, the resource-constrained deployments highly benefit from high degree of specialization. Some deployments utilize these architectures, such as multiple versions of Intel's Movidius Myriad vision processing unit (VPU) [20, 21, 40] and the newer CoralAI Edge TPUs [22]. Nevertheless, to the best of our knowledge, there has not been a thorough benchmarking of this variety of devices in the context of image processing on satellites like the one we perform in Section 3.

In addition to investigating the possibilities and challenges for space edge IoT [16], there have been efforts to tackle the challenges of terrestrial edge IoT. These works include benchmarking machine learning performance for the edge [5, 6, 10, 30, 61], optimization of deep learning inference on resource-constrained devices by leveraging sparsity [42, 55], quantization [7, 13, 50, 57], resource management [31, 33, 38] or application-specific optimizations [35, 60, 62]. Furthermore, there has been work on overcoming the network bandwidth limitations of edge devices using compression [32], collaborative data processing using both edge devices and the cloud [4, 24, 59], managing and processing data in resource-constrained environments [37, 41, 47, 49], and managing IoT data in the cloud [2, 8, 19]. We complement these works by having a very specific data-intensive application focus deployed on satellites.

5 CONCLUSION

This paper presented our work and lessons-learned when building an Image Processing Unit (IPU) for the first satellite in the DISCO project. We characterized the performance of seven systems that were possible candidates for accelerating deep-learning-based image processing on board of a small satellite. The latency and power requirements were only met by the most specialized hardware. Coral Dev Board Mini was chosen as the best candidate to serve as an IPU on DISCO1. The end-product, the DISCO1 satellite, was launched in space in April 2023 and serves as a testbed for student projects and future satellite designs for similar Earth observation cases. All code used for evaluation of devices is available open-source[3]. In an extended version of this paper [9], we outline our experience of integrating the IPU on board of DISCO1, including the steps we took to increase its robustness and fault-tolerance.

As for the ongoing and future research directions in the DISCO project, in addition to an IPU design using the Toradex Verdin device, we work on a power-aware scheduler and a modular image processing pipeline. The former aims at orchestrating the workloads on the satellite without depleting its power reserves, while the latter aims at enabling incremental and partial updates to the image tasks.

ACKNOWLEDGEMENTS

The work is funded by the Independent Research Fund Denmark's Inge Lehmann program under grant agreement number 0171-00062B and the Novo Nordisk Foundation Natural and Technical Sciences program under grant agreement number NNF22OC0079398.

[3] https://github.com/Resource-Aware-Data-systems-RAD/DISCO-IPU-Benchmark

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *OSDI*. 265–283.

[2] Ahmet Kerem Aksoy, Pavel Dushev, Eleni Tzirita Zacharatou, Holmer Hemsen, Marcela Charfuelan, Jorge-Arnulfo Quiané-Ruiz, Begüm Demir, and Volker Markl. 2022. Satellite Image Search in AgoraEO. *PVLDB* 15, 12 (2022), 3646–3649.

[3] L. Alminde, M. Bisgaard, D. Vinther, T. Viscor, and K. Ostergard. 2003. Educational value and lessons learned from the AAU-CubeSat project. In *International Conference on Recent Advances in Space Technologies.*

[4] RJ Atwal, Peter Boncz, Ryan Boyd, Antony Courtney, Till Döhmen, Florian Gerlinghoff, Jeff Huang, Joseph Hwang, Raphael Hyde, Elena Felder, Jacob Lacouture, Yves Le Maout, Boaz Leskes, Yao Liu, Alex Monahan, Dan Perkins, Tino Tereshko, Jordan Tigani, Nick Ursa, Stephanie Wang, and Yannick Welsch. 2024. MotherDuck: DuckDB in the cloud and in the client. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, CA, USA, January 14-17, 2024*. www.cidrdb.org. https://www.cidrdb.org/cidr2024/papers/p46-atwal.pdf

[5] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. 2021. MLPerf Tiny Benchmark. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks* (2021).

[6] Colby R. Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, David Patterson, Danilo Pau, Jae sun Seo, Jeff Sieracki, Urmish Thakker, Marian Verhelst, and Poonam Yadav. 2020. Benchmarking TinyML Systems: Challenges and Direction. *CoRR* abs/2003.04821 (2020).

[7] Tom Bannink, Adam Hillier, Lukas Geiger, Tim de Bruin, Leon Overweel, Jelmer Neeven, and Koen Helwegen. 2021. Larq Compute Engine: Design, Benchmark and Deploy State-of-the-Art Binarized Neural Networks. *Proceedings of Machine Learning and Systems* 3 (March 2021), 680–695.

[8] Ronald Barber, Christian Garcia-Arellano, Ronen Grosman, René Müller, Vijayshankar Raman, Richard Sidle, Matt Spilchen, Adam J. Storm, Yuanyuan Tian, Pinar Tözün, Daniel C. Zilio, Matt Huras, Guy M. Lohman, Chandrasekaran Mohan, Fatma Özcan, and Hamid Pirahesh. 2017. Evolving Databases for New-Gen Big Data Applications. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2017/papers/p123-barber-cidr17.pdf

[9] Robert Bayer, Julian Priest, and Pınar Tözün. 2023. Reaching the Edge of the Edge: Image Analysis in Space. (2023). arXiv:2301.04954 [cs.LG]

[10] Robert Bayer, Jon Voigt Tøttrup, and Pinar Tözün. 2022. TPCx-AI on NVIDIA Jetsons. In *TPCTC (Lecture Notes in Computer Science, Vol. 13860)*. 49–66.

[11] Adam D. Braun. 2018. Investigation of Deep Neural Network Image Processing for CubeSat Size Satellites. *MSc Thesis, Morehead State University* (2018).

[12] Cal Poly 2022. *CubeSat Design Specification.* Cal Poly. rev 14.1.

[13] Sivakumar Chidambaram, Pierre Langlois, and Jean-Pierre David. 2020. PoET-BiN: Power Efficient Tiny Binary Neurons. *Proceedings of Machine Learning and Systems* 2 (March 2020), 160–171.

[14] CoralAI. 2020. CoralAI TPU Technology. https://coral.ai/technology/.

[15] CoralAI. 2022. Edge TPU Compiler. https://coral.ai/docs/edgetpu/compiler/.

[16] Bradley Denby and Brandon Lucia. 2020. Orbital Edge Computing: Nanosatellite Constellations as a New Class of Computer System. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 939–954. https://doi.org/10.1145/3373376.3378473

[17] DISCO 2022. Danish Student CubeSat Program. https://discosat.dk/.

[18] FreeRTOS. 2023. FreeRTOS. https://www.freertos.org/index.html.

[19] Christian Garcia-Arellano, Hamdi Roumani, Richard Sidle, Josh Tiefenbach, Kostas Rakopoulos, Imran Sayyid, Adam Storm, Ronald Barber, Fatma Ozcan, Daniel Zilio, Alexander Cheung, Gidon Gershinsky, Hamid Pirahesh, David Kalmuk, Yuanyuan Tian, Matthew Spilchen, Lan Pham, Darren Pepper, and Gal Lushi. 2020. Db2 Event Store: A Purpose-Built IoT Database Engine. *Proceedings of the VLDB Endowment* 13, 12 (Aug. 2020), 3299–3312. https://doi.org/10.14778/3415478.3415552

[20] Gianluca Giuffrida, Lorenzo Diana, Francesco de Gioia, Gionata Benelli, Gabriele Meoni, Massimiliano Donati, and Luca Fanucci. 2020. CloudScout: A Deep Neural Network for On-Board Cloud Detection on Hyperspectral Images. *Remote Sensing* 12 (07 2020), 2205.

[21] Gianluca Giuffrida, Luca Fanucci, Gabriele Meoni, Matej Batič, Léonie Buckley, Aubrey Dunne, Chris van Dijk, Marco Esposito, John Hefele, Nathan Vercruyssen, Gianluca Furano, Massimiliano Pastena, and Josef Aschbacher. 2022. The Φ-Sat-1 Mission: The First On-Board Deep Neural Network Demonstrator for Satellite Earth Observation. *IEEE Transactions on Geoscience and Remote Sensing* 60 (2022), 1–14. https://doi.org/10.1109/TGRS.2021.3125567

[22] Justin Goodwill, Gary Crum, James Mackinnon, Cody Brewer, Michael Monaghan, Travis Wise, and Christopher Wilson. 2021. NASA SpaceCube Edge TPU SmallSat Card for Autonomous Operations and Onboard Science-Data Analysis. In *Proceedings of the Small Satellite Conference.*

[23] Google. 2022. TensorFlow Lite. https://www.tensorflow.org/lite/guide.

[24] Philipp M. Grulich and Faisal Nawab. 2018. Collaborative Edge and Cloud Neural Networks for Real-Time Video Processing. *Proceedings of the VLDB Endowment* 11, 12 (Aug. 2018), 2046–2049. https://doi.org/10.14778/3229863.3236256

[25] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR* (2017).

[26] Intel. 2017. Intel Neural Compute Stick 2 (NCS2). https://www.intel.com/content/www/us/en/developer/articles/tool/neural-compute-stick.html.

[27] Intel. 2023. OpenVino. https://docs.openvino.ai/latest/home.html.

[28] Space Inventor. 2020. OBC-P3. https://www.satcatalog.com/component/obc-p3/.

[29] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2704–2713. https://doi.org/10.1109/CVPR.2018.00286

[30] Vijay Janapa Reddi, David Kanter, Peter Mattson, Jared Duke, Thai Nguyen, Ramesh Chukka, Ken Shiring, Koan-Sin Tan, Mark Charlebois, William Chou, Mostafa El-Khamy, Jungwook Hong, Tom St John, Cindy Trinh, Michael Buch, Mark Mazumder, Relja Markovic, Thomas Atta, Fatih Cakir, Masoud Charkhabi, Xiaodong Chen, Cheng-Ming Chiang, Dave Dexter, Terry Heo, Guenther Schmuelling, Maryam Shabani, and Dylan Zika. 2022. MLPerf Mobile Inference Benchmark: An Industry-Standard Open-Source Machine Learning Benchmark for On-Device AI. *Proceedings of Machine Learning and Systems* 4 (April 2022), 352–369.

[31] Fucheng Jia, Deyu Zhang, Ting Cao, Shiqi Jiang, Yunxin Liu, Ju Ren, and Yaoxue Zhang. 2022. CoDL: Efficient CPU-GPU Co-Execution for Deep Learning Inference on Mobile Devices. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '22)*. Association for Computing Machinery, New York, NY, USA, 209–221. https://doi.org/10.1145/3498361.3538932

[32] Pu Jiao, Sheng Di, Hanqi Guo, Kai Zhao, Jiannan Tian, Dingwen Tao, Xin Liang, and Franck Cappello. 2022. Toward Quantity-of-Interest Preserving Lossy Compression for Scientific Data. *Proceedings of the VLDB Endowment* 16, 4 (Dec. 2022), 697–710. https://doi.org/10.14778/3574245.3574255

[33] Youngsok Kim, Joonsung Kim, Dongju Chae, Daehyun Kim, and Jangwoo Kim. 2019. μLayer: Low Latency On-Device Inference Using Cooperative Single-Layer Acceleration and Processor-Friendly Quantization. In *Proceedings of the Fourteenth EuroSys Conference 2019 (EuroSys '19)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3302424.3303950

[34] Michael D. King, Steven Platnick, W. Paul Menzel, Steven A. Ackerman, and Paul A. Hubanks. 2013. Spatial and Temporal Distribution of Clouds Observed by MODIS Onboard the Terra and Aqua Satellites. *IEEE Transactions on Geoscience and Remote Sensing* 51, 7 (July 2013), 3826–3852. https://doi.org/10.1109/TGRS.2012.2227333

[35] Venkatesh Kodukula, Alexander Shearer, Van Nguyen, Srinivas Lingutla, Yifei Liu, and Robert LiKamWa. 2021. Rhythmic Pixel Regions: Multi-Resolution Visual Sensing System towards High-Precision Visual Computing at Low Power. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 573–586. https://doi.org/10.1145/3445814.3446737

[36] Liangzhen Lai, Naveen Suda, and Vikas Chandra. 2018. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. *CoRR* abs/1801.06601 (2018).

[37] Aljoscha P. Lepping, Hoang Mi Pham, Laura Mons, Balint Rueb, Philipp M. Grulich, Ankit Chaudhary, Steffen Zeuch, and Volker Markl. 2023. Showcasing Data Management Challenges for Future IoT Applications with NebulaStream. *Proc. VLDB Endow.* 16, 12 (2023), 3930–3933. https://doi.org/10.14778/3611540.3611588

[38] Daniyal Liaqat, Silviu Jingoi, Eyal de Lara, Ashvin Goel, Wilson To, Kevin Lee, Italo De Moraes Garcia, and Manuel Saldana. 2016. Sidewinder: An Energy Efficient and Developer Friendly Heterogeneous Architecture for Continuous Mobile Sensing. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. Association for Computing Machinery, New York, NY, USA, 205–215. https://doi.org/10.1145/2872362.2872398

[39] Martina Lofqvist and José Cano. 2020. Accelerating Deep Learning Applications in Space. In *Proceedings of the Small Satellite Conference.*

[40] Gonzalo Mateo-Garcia, Josh Veitch-Michaelis, Cormac Purcell, Nicolas Longepe, Simon Reid, Alice Anlind, Fredrik Bruhn, James Parr, and Pierre Philippe Mathieu. 2023. In-Orbit Demonstration of a Re-Trainable Machine Learning Payload for Processing Optical Imagery. *Scientific Reports* 13, 1 (June 2023), 10391. https://doi.org/10.1038/s41598-023-34436-w

[41] Adrian Michalke, Philipp M. Grulich, Clemens Lutz, Steffen Zeuch, and Volker Markl. 2021. An Energy-Efficient Stream Join for the Internet of Things. In *Proceedings of the 17th International Workshop on Data Management on New Hardware, DaMoN 2021, 21 June 2021, Virtual Event, China*, Danica Porobic and Spyros Blanas (Eds.). ACM, 8:1–8:6. https://doi.org/10.1145/3465998.3466005

[42] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-based Weight Pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 907–922. https://doi.org/10.1145/3373376.3378534

[43] NVIDIA. 2019. Jetson Nano Developer Kit. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[44] IT University of Copenhagen. 2023. Students launch a satellite to test artificial intelligence in space. https://en.itu.dk/About-ITU/Press/News-from-ITU/2023/Students-launch-a-satellite-to-test-artificial-intelligence-in-space.

[45] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NIPS*. 8024–8035.

[46] Raspberry Pi. 2016. Raspberry Pi 3 Model B. https://www.raspberrypi.com/products/raspberry-pi-3-model-b/.

[47] Philippe Pucheral, Luc Bouganim, Patrick Valduriez, and Christophe Bobineau. 2001. PicoDBMS: Scaling down Database Techniques for the Smartcard. *The VLDB Journal* 10, 2–3 (2001), 120–132.

[48] Qoitech. 2022. Otii Arc Pro. https://www.qoitech.com/otii-arc-pro/.

[49] Mark Raasveldt and Hannes Mühleisen. 2020. Data Management for Data Science Towards Embedded Analytics. In *CIDR*.

[50] Manuele Rusci, Alessandro Capotondi, and Luca Benini. 2020. Memory-Driven Mixed Low Precision Quantization for Enabling Deep Network Inference on Microcontrollers. *Proceedings of Machine Learning and Systems* 2 (March 2020), 326–335.

[51] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV* 115, 3 (2015), 211–252.

[52] Bronco Space. 2021. BroncoSat1. https://www.broncospace.com/broncosat-1.

[53] Toradex. 2023. NXP® i.MX 8M Plus Computer on Module. https://www.toradex.com/computer-on-modules/verdin-arm-family/nxp-imx-8m-plus.

[54] Yuichi Tsuda, Nobutada Sako, Takashi Eishima, Takahiro Ito, Yoshihisa Arikawa, Norihide Miyamura, Akira Tanaka, and Shinichi Nakasuka. 2001. University of Tokyo's CubeSat Project: Its Educational and Technological Significance. In *Proceedings of the Small Satellite Conference*.

[55] Yaman Umuroglu, Yash Akhauri, Nicholas James Fraser, and Michaela Blott. 2020. LogicNets: Co-Designed Neural Networks and Circuits for Extreme-Throughput Applications. In *IEEE FPL*. 291–297.

[56] Han Vanholder. 2016. Efficient inference with tensorrt. In *GPU Technology Conference*, Vol. 1.

[57] Jaeyeon Won, Jeyeon Si, Sam Son, Tae Jun Ham, and Jae W. Lee. 2022. ULPPACK: Fast Sub-8-bit Matrix Multiply on Commodity SIMD Hardware. *Proceedings of Machine Learning and Systems* 4 (April 2022), 52–63.

[58] X-CUBE-AI 2022. AI expansion pack for STM32CubeMX. https://www.st.com/en/embedded-software/x-cube-ai.html.

[59] Steffen Zeuch, Ankit Chaudhary, Bonaventura Del Monte, Haralampos Gavriilidis, Dimitrios Giouroukis, Philipp M. Grulich, Sebastian Bress, Jonas Traub, and Volker Markl. 2020. The NebulaStream Platform for Data and Application Management in the Internet of Things. In *CIDR*. 1–11.

[60] Xiaofan Zhang, Haoming Lu, Cong Hao, Jiachen Li, Bowen Cheng, Yuhong Li, Kyle Rupnow, Jinjun Xiong, Thomas Huang, Honghui Shi, Wen-Mei Hwu, and Deming Chen. 2020. SkyNet: A Hardware-Efficient Method for Object Detection and Tracking on Embedded Systems. *Proceedings of Machine Learning and Systems* 2 (March 2020), 216–229.

[61] Xiaofan Zhang, Hanchen Ye, and Deming Chen. 2021. Being-ahead: Benchmarking and Exploring Accelerators for Hardware-Efficient AI Deployment. In *Proceedings of Machine Learning and Systems (Workshop on Benchmarking Machine Learning Workloads on Emerging Hardware)*.

[62] Yifan Zhao, Hashim Sharif, Peter Pao-Huang, Vatsin Shah, Arun Narenthiran Sivakumar, Mateus Valverde Gasparino, Abdulrahman Mahmoud, Nathan Zhao, Sarita Adve, Girish Chowdhary, Sasa Misailovic, and Vikram Adve. 2023. ApproxCaliper: A Programmable Framework for Application-aware Neural Network Optimization. *Proceedings of Machine Learning and Systems* 5 (March 2023).