The not-so-easy task of taking heavy-lift ML models to the edge: a performance-watt perspective

Lucas Meireles[1], Bruna Guterres[1], Kauê Sbrissa[1], Amanda Mendes [1], Francisca Vermeulen [2], Lisl Lain [3], Marié Smith [3], Javier Martinez [4], Paulo Drews [1], Nelson Duarte Filho[1], Vinicus Oliveira[1], Silvia Botelho[1], Marcelo Pias [1]
[1]Federal University of Rio Grande (FURG), Computer Science Centre. Rio Grande, Brazil. [2] SAMS. The Scottish Association for Marine Science. Oban, UK. [3] CSIR. South Africa's Council for Scientific and Industrial Research. South Africa. [4] LEITAT. Acondicionamiento Tarrasense Association. Barcelona, Spain.
caetano02117@gmail.com

ABSTRACT

Edge computing is a new development paradigm that brings computational power to the network edge through novel intelligent end-user services. It allows latency-sensitive applications to be placed where the data is created, thus reducing communication overhead and improving security, mobility and power consumption. There is a plethora of applications benefiting from this type of processing. Of particular interest is emerging edge-based image classification at the microscopic level. The scale and magnitude of the objects to segment, detect and classify are very challenging, with data collected using order of magnitude in magnification. The required data processing is intense, and the wish list of end-users in this space includes tools and solutions that fit into a desk-based device. Taking heavy-lift classification models initially built in the cloud to desk-based image analysis devices is a hard job for application developers. This work looks at the performance limitations and energy consumption footprint in embedding deep learning classification models in a representative edge computing device. Particularly, the dataset and heavy-lift models explored in the case study are phytoplankton images to detect Harmful Algae Blooms (HAB) in aquaculture at early stages. The work takes a deep learning model trained for phytoplankton classification and deploys it at the edge. The *embedded model*, deployed in a base form alongside optimised options, is submitted to a series of system stress experiments. The performance and power consumption profiling help understand system limitations and their impact on the microscopic grade image classification task.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; • **Applied computing** → *Computers in other domains*;

KEYWORDS

Edge Computing, Artificial Intelligence, Edge-based Deep Learning, Performance and Energy Efficiency

1 INTRODUCTION

The development of novel machine learning (ML) applications has followed the needs of modern society. The increase in data volume and heterogeneous systems pushes computing power to a new level. On one side, cloud computing could be more optimal when complying with service level agreements of real-time, privacy-first and low-power end-user services [4]. On the other side, edge computing emerges as a better strategy to collect and process data (entirely or partially) at the device where it is created. This improves performance and power consumption, avoiding unnecessary costly data communication transfer. Artificial Intelligent (AI) applications have grown in recent years, both in number and complexity [5, 12, 15, 21]. As a result, more data is available for advanced big data processing to produce timely and relevant insights. More specifically, deep learning has experienced the most remarkable breakthroughs among many AI subareas [10]. Deep Neural Networks (DNNs), including canonical forms of Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and Generative Adversarial Networks (GANs), have been explored in practical deployments such as autonomous driving, voice assistants, predictive maintenance, and many others.

Edge computing brings the processing power and storage closer to the data source at the end-user device [13]. This feature allows for fast data processing and real-time response time [9]. The edge computing application communicates still over the internet to the services running on a cloud server, building a system where processing power is accessible from everywhere via the web [11]. Internet of Things (IoT) applications are exemplary use cases for edge computing. Leveraging the communication network capabilities offers opportunities to lower the overhead along the dimensions of power consumption and real-time performance (*performance-watt*).

Edge devices have grown in popularity, reaching many industrial sectors, including precision agriculture and aquaculture, high-value manufacturing, and home automation. In addition, as we expand the data availability, machine learning models can solve very specific yet practical tasks of the real world. The digital twinning in the upcoming Industry 5.0 [2] is tasked with processing data to boost innovation towards sustainable production that obeys the planet's limits. As planetary twins scale up, increased growth of IoT sensors will collect substantial amounts of unstructured data. To make sense of this data promptly, edge-device twins can deliver expected outcomes in climate-resilient production applications.

Precision agriculture and aquaculture applications require integrating data, edge processing and embedded machine learning for deploying low-cost, real-time IoT and digital twin systems. Aquaculture consists of farming aquatic organisms, including fish, molluscs, crustaceans, and aquatic plants. The farming process implies a degree of real-time control and actuation to enhance production, such as water quality control, uniform stocking, feeding, protection from predating, and disease prevention [6].

Climate change is creating environmental conditions for the worldwide surge in harmful algal blooms (HABs) [19]. Such harmful phytoplankton biomass seriously impacts aquaculture with events of oxygen depletion and consequent death of aquatic organisms (e.g. intense aquaculture farming can face production loss in up to 30 minutes in case of no mitigation action is taken). Early HAB detection is essential to react and intervene in the aquaculture process. State-of-the-art methods rely on late decisions based on the cloud-based processing of satellite images. Providing reliable edge-based phytoplankton monitoring contributes towards climate-resilient solutions that comprise desk-based image analysers equipped with self-contained ML models capable of producing results within seconds (i.e. time for an inference).

Edge computing can substantially contribute to building a robust Climate-Ocean-Food value chain, linking expected environmental risks to cost-efficiency and best practices of aquaculture production and food safety (Industry 5.0). Application developers[1] embrace challenges in creating machine learning models in the cloud and attempting to deploy such models into edge computing devices. The severe resource constraints affect data processing, memory usage, communication and power duty cycle. Developers should follow guidelines for deploying models informed by edge device system limitations. Considerable overhead (i.e. "high AI tax") on the overall power consumption [3] is expected. This paper addresses the following Research Questions (RQ):

- **RQ1** What is the performance and accuracy impact of taking heavy-weight cloud-based ML models to resource-constrained devices at the edge?
- **RQ2** What is the power consumption footprint in running ML image classifiers in desk-based analyser devices?

The proposed work integrates new and classic machine learning approaches and deploys models to the edge using a mid-range device. Early HAB detection is the case study explored in this work. Performance-watt, memory usage and power consumption are metrics for system validation. The paper is organised as follows: Section 2 discusses the related work. Section 3 introduces the methodology and the experimental design, and Section 4 discusses the results. Finally, section 5 draws the main conclusions of the work.

2 BACKGROUND AND RELATED WORK

Bianco et al. [1] offer an in-depth analysis of several deep neural networks available in the literature. To achieve this, the authors explore a series of indicators such as accuracy, model complexity, computational complexity, memory usage, and inference time. The behaviour of such metrics and some combinations of them are analysed and discussed. For the present work, we relied on this to establish the metrics used in our case study. We also used these results to guide our choice of model. The results suggested that the MobileNetV2 [12] is a reasonable DNN candidate to be deployed at the edge. The paper discusses solid accuracy results, mainly Top5 accuracy. This metric measures how efficiently each model uses its parameters, indicating the overall efficiency of the results in terms of resources used. However, such work carried out experiments on a general-purpose dataset that does not fully translate into our case study (i.e. early detection of HABs from microscopic images). Additionally, Bianco et al. [1] does not use an optimisation engine to perform experiments on the edge device, leaving an open gap to be explored.

Buch et al. [3] explore individual execution stages of ML applications. It quantifies performance penalties in each process step. First, the paper characterises a high-level pipeline for a typical end-to-end image classification ML task. Such a pipeline consists of Data Capture, Pre-processing, Frameworks, Execution and Post-Processing. The sources of overheads in the ML pipeline are classified as algorithms, frameworks, or hardware. The combined end-to-end latency of the ML execution pipeline is referred to as AI Tax. Finally, the authors explore possible overheads surrounding the inference process, as frameworks deal with low-level software in the form of drivers that coordinate scheduling and optimisations. However, the work offers no comparison with the TensorRT optimisation engine, widely used in embedded hardware. This tool fits the framework defined by the authors. TensorRT serves as a model optimisation interface, running under the inference step. The work is limited in understanding the edge-based performance of this optimisation framework concerning a base case. This paper aims to shed some light on possible trade-offs associated with classification performance (e.g. accuracy, speed) on the edge and resource usage. It investigates the ability to use low-cost off-the-shelf embedded platforms for technology deployment.

Shafi et al. [16] is an effort to characterise the impact of using the TensorRT optimisation engine on the inference process of deep learning models deployed on the edge. The work explores the impact of TensorRT on the results and performance of DNNs deployed at Nvidia hardware. Relevant DNNs are tested, and the interaction between software optimisations and the GPU hardware is examined. It also performs empirical analyses based on TensorRT in real embedded GPU platforms using a variety of widely used DNNs. The authors report some interesting findings, such as that the TensorRT sustains the DNN's accuracy, even compared to the un-optimised DNN models. If the base (non-optimised) models suffer from over-fitting problems, TensorRT optimisations (e.g. weight quantisation) can reduce it, maintaining accuracy or even providing slight improvements. Shafi et al. [16] indicates a significant gain in image throughput from model optimisation

[1]Professionals in multidisciplinary teams of software developers, data engineers and data scientists.

through TensorRT. It also highlights that some models require additional time to copy the TensorRT-optimized models to GPU memory. The findings suggest TensorRT should be considered when proposing an edge system based on DNNs. However, the work does not fully address important metrics for embedded applications, such as the impact of TensorRT optimisation on energy consumption on the edge.

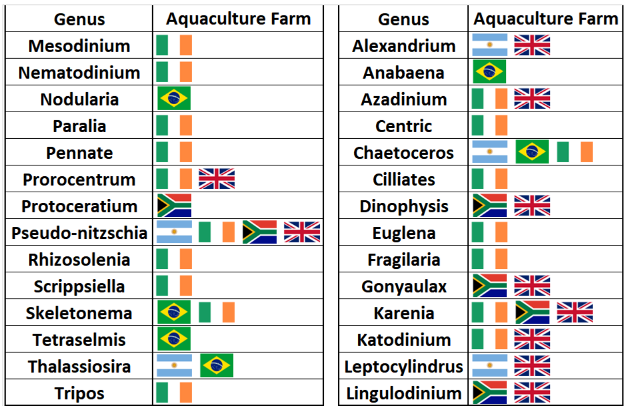
Guterres et al. [8] proposes and develops a complete pipeline for integrating and standardising phytoplankton image databases that can be used as a case study for edge application validation. Over time researchers presented multiple phytoplankton datasets to the scientific community. However, they are heterogeneous in many ways. Phytoplankton biota is the basis for the worldwide marine ecosystem, it is vastly rich in species variety and geographical presence. The construction of reliable AI models for HAB monitoring relies on a rich representative database to fulfil the application needs regarding species considered and classification performance. Guterres et al. [8] created a data integration pipeline to take advantage of the sparsely available databases and generate a unified dataset, homogeneous in form and content. It applies it to the public phytoplankton dataset to generate a geographically representative, suitable and labelled phytoplankton database.

The present work sheds some light on the gaps identified and the feasibility of using the TensorRT optimisation engine to deploy state-of-the-art models into resource-constrained edge-based embedded systems. It attempts to build a set of system guidelines and principles for application developers wishing to develop ML models for embedded systems at the network edge.

2.1 Case Study: Application Requirements

Digital technologies – digital twin, IoT and Edge AI – can significantly contribute to developing global Aquaculture Climate-Ocean-Food value chains. The Edge AI case study proposed in this work evaluates state-of-the-art HAB monitoring ML models to the aquaculture farming end-users (edge of the network). The aim is to develop edge-based predictive monitoring to enhance local producers' knowledge and lower existing barriers in this application domain. The list that follows describes the requirements gathered from the end-users:

- Low-cost and real-time monitoring of farming facilities.
- Low-power and off-grid technological solutions deployed in remote areas.
- Improved communication coverage in areas of increasingly difficult access.
- Continuous monitoring and real-time visualisation of relevant event data (e.g. water dissolved oxygen).
- Access to desk-based devices for advanced water quality monitoring (e.g. harmful algae bloom detection) using automated software.



Genus	Aquaculture Farm	Genus	Aquaculture Farm
Mesodinium	🇮🇪	Alexandrium	🇦🇷 🇬🇧
Nematodinium	🇮🇪	Anabaena	🇧🇷
Nodularia	🇧🇷	Azadinium	🇮🇪 🇬🇧
Paralia	🇮🇪	Centric	🇮🇪
Pennate	🇮🇪	Chaetoceros	🇦🇷 🇧🇷 🇮🇪
Prorocentrum	🇮🇪 🇬🇧	Cilliates	🇮🇪
Protoceratium	🇿🇦	Dinophysis	🇿🇦 🇬🇧
Pseudo-nitzschia	🇦🇷 🇮🇪 🇿🇦 🇬🇧	Euglena	🇮🇪
Rhizosolenia	🇮🇪	Fragilaria	🇮🇪
Scrippsiella	🇮🇪	Gonyaulax	🇿🇦 🇬🇧
Skeletonema	🇧🇷 🇮🇪	Karenia	🇮🇪 🇿🇦 🇬🇧
Tetraselmis	🇧🇷	Katodinium	🇮🇪 🇬🇧
Thalassiosira	🇦🇷 🇧🇷	Leptocylindrus	🇦🇷 🇬🇧
Tripos	🇮🇪	Lingulodinium	🇿🇦 🇬🇧

Figure 1: Genera of interest within aquaculture applications in several countries.

2.2 Phytoplankton Datasets

The present work extends cloud-based ML models to embedded system platforms. The aquaculture case study is the challenging scenario of climate-resilient solutions for HAB monitoring and early detection. The application would benefit from a very low-power device that sits on a desk and is readily available for a practitioner to analyse a water sample. Such a device could be a low-power embedded system – *edge computing device* – that comprises a CPU, GPU accelerator, memory, communication subsystem, and a camera with a microscopic tunable lens.

The data integration pipeline consists of publicly available image datasets (at a microscopic scale) that cover phytoplankton genera collected from aquaculture facilities in several countries. Figure 1 shows the phytoplankton genera. The image database includes representative grey-scaled images for the harmful phytoplankton classes that create blooms in aquaculture facilities. Model training has used this dataset to adjust the parameters of convolution neural networks (CNNs) architectures. Figure 2 depicts microscopic-level phytoplankton image examples used to build the image classifier.

The integrated dataset contains around 81,391 images across target phytoplankton genera. It has been split into *training (80%)* and *testing (20%)* datasets. The training part has been used in the model training in a cloud-based server, whereas the testing part has supported the edge AI experiments (inference task) carried out in this work.

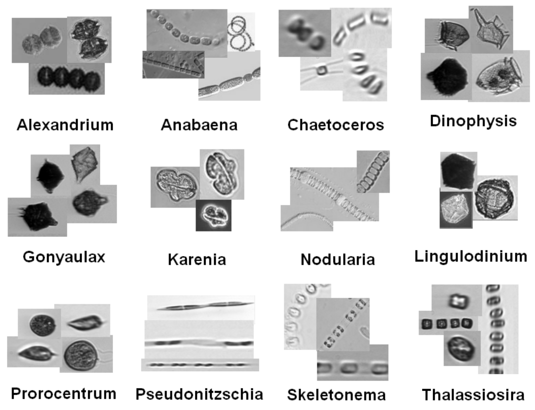


Figure 2: Phytoplankton image examples (50+ times magnification). High intra-class variability, inter-class similarity and imbalanced scenarios bring issues for the practical identification of phytoplankton organisms.

2.3 Classifier Model

The MobileNet architecture [12] is a simple but efficient convolution neural network (CNN) intended for computer vision applications in mobile devices. It does not require intensive computational resources for the training and inference tasks. Many real-world applications employ this kind of CNN model (e.g. object detection, fine-grained classifications, face recognition, and object localisation), especially on embedded devices. MobileNet introduces depth-wise separable convolutions along with 1×1 point-wise convolutions. The first layer applies a convolution filter for each input channel. The second layer builds new features using a 1 x 1 convolution and linear combinations of the input channels.

The model training used the phytoplankton dataset and transfer learning from ImageNet. The training happened in a cloud server equipped with an NVIDIA GPU Pascal family (e.g. 3090). The output of this step is a standard TensorFlow model meant to be used in cloud-based applications. We also evaluate the potential advantages of model optimisations for taking the standard TensorFlow model (cloud-based) to an edge device (optimised). To achieve this, the optimisation engine TensorRT [2] helped with a set of tricks that include model parameter changes, neural network architecture adjustments and numeric data representation (quantisation). In addition, an assessment of power consumption, resource usage and classification accuracy was carried out for the original and optimised models.

3 EXPERIMENTAL SETUP AND ANALYSIS METHODS

The Jetson Nano platform is a low-cost, widely available edge device[3]. The platform's TensorRT engine optimises trained AI models for run-time performance in resource-constrained environments. The engine minimises latency, increases throughput and delivers faster and less power-hungry models. The Jetson Nano platform is used to run experiments and evaluate the possible effects of taking a model trained in a cloud-based environment (cloud model) with plenty of computational resources into a resource-constrained edge device for the model inference task. The TensorRT acceleration library also boosts cloud-based AI models for further integration into a Jetson Nano platform. It aims to provide models tailored for the resource-constrained environment. The present work evaluates the advantages of the TensorRT optimisation engine when taking cloud-based deep learning models to edge-embedded systems.

3.1 System Setup

The Jetson Nano (Table 1) is an entry-level Edge device by NVIDIA, a powerful single-board computer in a small, portable form factor with parallel cores for AI applications. It is a low-cost development tool for image classification, object detection, segmentation, and speech processing applications. The development environment JetPack allows AI applications to start quickly through a complete Linux environment with ready-to-go libraries. NVIDIA's TensorRT is a high-performance library that interfaces deep learning applications with production environments. It enables easy deployment of deep learning models in edge environments with improved performance and efficiency. Many applications have used this edge computing platform in recent years [7, 14, 17, 18]. In addition, Shafi et al. [16] has extensively tested this platform as an optimisation engine for edge applications.

[2]https://developer.nvidia.com/tensorrt
[3]https://developer.nvidia.com/embedded/jetson-nano-developer-kit

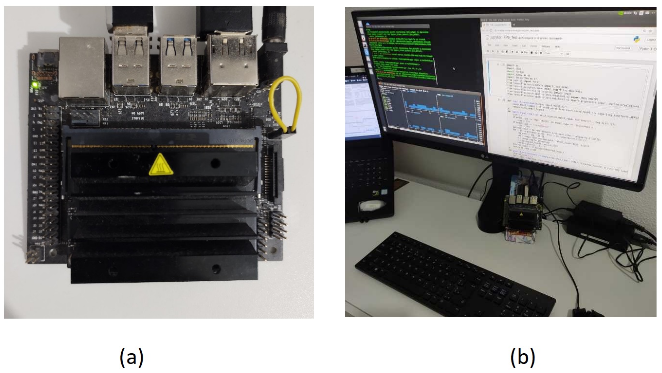


Figure 3: (a) Jetson Nano platform and (b) the experimental system setup

Table 1: NVIDIA JETSON Nano Specifications

CPU	ARM Cortex-A57 (quadcore)	@1.73GHz
GPU	256-core Maxwell	@998MHz
Memory	4GB 64-bit LPDDR4	@1600MHz | 25.6 GB/s
Storage	16GB eMMC 5.1	-
Power	10W	-
Jetpack	4.6	[L4T 32.6.1]
CUDA	10.2.300	-
cuDNN	8.2.1.32	-
TensorRT	8.0.1.6	-

TensorRT executes several crucial transformations and optimisations directly on AI models (Figure 4). First, it drops any layer with unused outputs, thus avoiding wasting computational resources and time. Following that, the TensorRT searches for any sets of remaining layers that can be fused. In this step, convolution, bias and ReLU layers across the network graph are fused vertically to improve running time. TensorRT also performs horizontal layer fusions to enhance performance by combining network layers that take the same source tensor and apply the same operations with similar parameters. The resulting single extensive layer contributes to computational efficiency. Finally, the model is quantised with the precision of choice (e.g. 32-bit floating point).

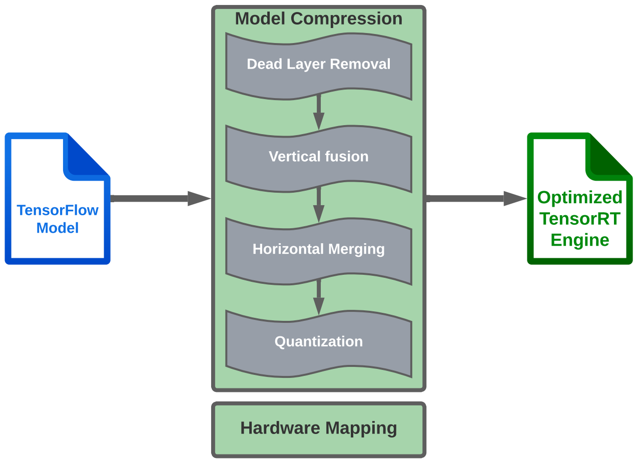


Figure 4: Optimisation steps

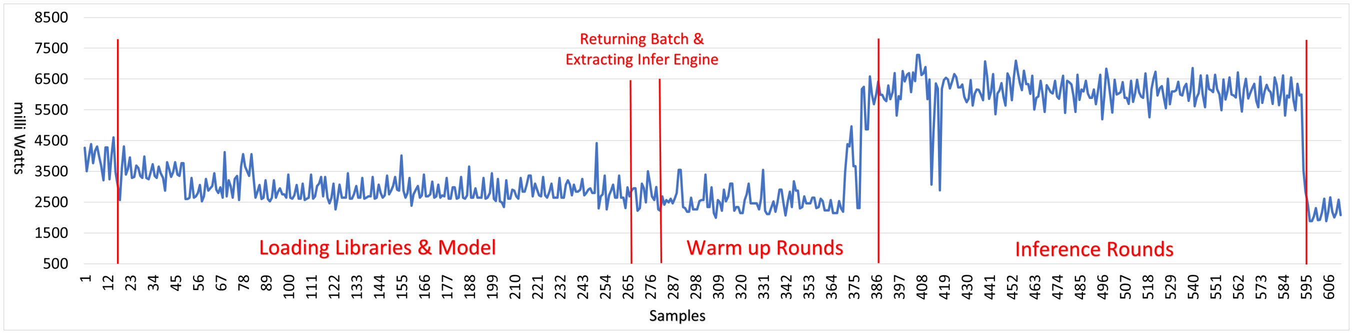


Figure 5: Power consumption: MobileNetV2 on the Jetson Nano platform. The *y axis* is the instantaneous power (in milli Watts); *x axis* is the time (in samples)

The present work carries out a set of experiments on a Jetson Nano platform (Figure 4). It aims to assess the effects of taking a model trained in a cloud-based environment to the Jetson Nano platform for edge inference tasks. It also evaluates the potential of using TensorRT optimisation engine to boost and support model deployment on the edge. The following AI models are considered and compared in terms of power consumption, resource usage and classification inference accuracy:

- **Cloud-based or base model:** MobileNetV2 architecture trained in a cloud-based environment with plenty of computational resources.
- **TRT-FP16 model:** cloud-based model transformed through optimisations to fit into a Jetson Nano platform. TensorRT performs model optimisations, including quantisation of 16-bit floating point.
- **TRT-FP32 model:** cloud-based model modified for the Jetson Nano platform. A 32-bit floating point quantisation has been used.

The TRT-FP32 and TRT-FP16 models are also referred to as *optimised models*. The resulting AI models are installed into the Jetson Nano platform. The experimental work has been split into steps to evaluate better how different AI model execution stages influence the performance, energy profile and resource usage on the edge device. The steps have vital roles in running the model for inference purposes. In addition, a monitoring logger software runs in parallel with the model execution to gather resource usage and energy consumption data. The steps are detailed below in sequential order:

- **1 - Loading Model** - Initial processes such as loading packages, initialising variables and establishing the DL model for the inference task.
- **2 - Extracting Infer Engine & Returning Batch** - Transparent steps without high-level libraries (e.g. Keras). These operations should be performed manually at this abstraction level to provide the code with the data and structure capable of passing it through the model.
- **3 - Warm Up Rounds** - Performance discrepancies are observed between the initial and last rounds of data inference on GPU. In the first rounds, it is still necessary to cache data and other procedures. So warm-up rounds are important to avoid "cold start" problems.
- **4 - Real Rounds** - This last stage includes the inference rounds when energy profile and resource usage are assessed.

3.2 Performance and Energy

3.2.1 Image Throughput. This metric estimates the image classification speed on edge devices (in Frames Per Second - FPS). This is a standard performance metric widely used in the image classification field. FPS is the number of images per unit of time that traverses the model to accomplish the inferences. We report and analyse the FPS metric to explore better the experiment's effects and costs of the inference task. In addition, one should address the overhead in the system resource usage.

3.2.2 Accuracy. Accuracy is a widely used and intuitive model evaluation metric for classification problems. In simple terms, it is the ratio of correctly predicted observations to the total number of observations in the dataset. We estimate Top-1 accuracy based on testing the image set of the integrated dataset tailored for aquaculture application needs. Shafi et al. [16] report that TensorRT can achieve similar (and even slightly better) accuracy results to models intended for the cloud. However, comparing cloud-based and optimised model versions from an accuracy point of view remains vital to identify possible trade-offs between model optimisation and performance gains on edge devices.

3.2.3 Memory Consumption. The present work reports the total memory consumption and a complete memory usage profile for each experiment undertaken. Memory is very limited in embedded systems, and monitoring this resource is crucial for deploying edge computing applications.

3.2.4 Power Consumption. Power consumption of the base and edge-optimised models is assessed to understand the impact of TensorRT on edge applications. The energy profile of the base and optimised models are compared. We also evaluate the ratio between FPS and power consumption to build a performance-watt 2-dimensional metric that brings information on possible FPS gains along with energy resource usage.

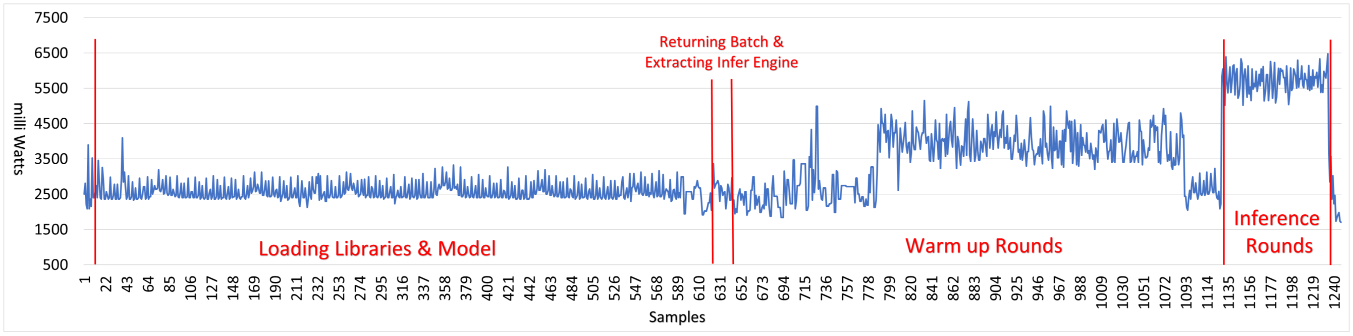


Figure 6: Power consumption: Jetson Nano System running the MobileNetV2 with TensorRT FP32. The *y axis* is the instantaneous power (in milli Watts); *x axis* is the time (in samples)

4 SYSTEM VALIDATION RESULTS

This section provides the experimental findings on taking cloud-based models to resource-constrained devices. It briefly presents the results at a high-level overview and provides further information on comparing the base model against its optimised versions. Image throughput improvements are also discussed in the context of the TensorRT optimisation engine.

Table 2: General Results

	Cloud-based	TRT-FP32	TRT-FP16
Accuracy (%)	97	97	97
Throughput (FPS)	41	82	84
Memory use (%)	93,2	99,5	99,8
Total Avg Power (mW)	3386	3185	3229
Inference Avg Power 2 (mW)	6125	5711	5860
Total Performance-watt (FPS/W)	12,10	25,74	26,01
Inference Performance-watt (FPS/W) 2	6,69	14,35	14,33

Table 2 suggests that the TensorRT optimisation engine plays a key role in performance across the experiments. First, it yields a substantial Throughput gain (in frames per second). The inference throughput doubled on the edge device with TensorRT optimisation. Such an improvement is expected because the engine is putting much effort into optimisation. Third, results support the initial feasibility assumption of using TensorRT on edge devices for a desk-based image analyser system. We evaluated the accuracy using a testing dataset with around 16,000 pictures, a quarter of the training dataset size. The AI models (base and optimised versions) were deployed into the Jetson Nano platform for inferences using the same testing dataset. The base model reached 97% accuracy, achieving a reasonable classification accuracy for the phytoplankton species of interest. The accuracy is the same as the one obtained in the validation tests in the cloud. Both optimised models (TRT-FP16 and TRT-FP32 model), when checked on the same set of testing images, reached a similar accuracy level compared to the base model. This result confirms Shafi et al. [16] findings that models optimised with TensorRT tend to achieve reasonably comparable accuracy results to non-optimised models. Loss in classification accuracy is minimal.

4.1 Power Consumption Profile

Power consumption is crucial in designing embedded edge devices, as the system can operate on a battery pack entirely. Power and energy profiling help understand the developed solution's applicability to the target environment, considering resource-constrained devices. Table 2 depicts the memory consumption results. Total Avg Power is the average power consumption across the entire experiment. Inference Avg Power is the average power in the inference step only. The Total Performance-Watt (in FPS/W) represents the cost of the achieved performance results. Also, the last table row shows the performance-watt, specifically for the inference phase.

4.1.1 Base Experiment. This experiment employs the base model MobileNetV2 (i.e. running an unmodified base model with no optimisations). Figure 6 presents the power consumption in sections that relate to the following steps: **1** Loading Model, **2** Extracting Infer Engine Returning

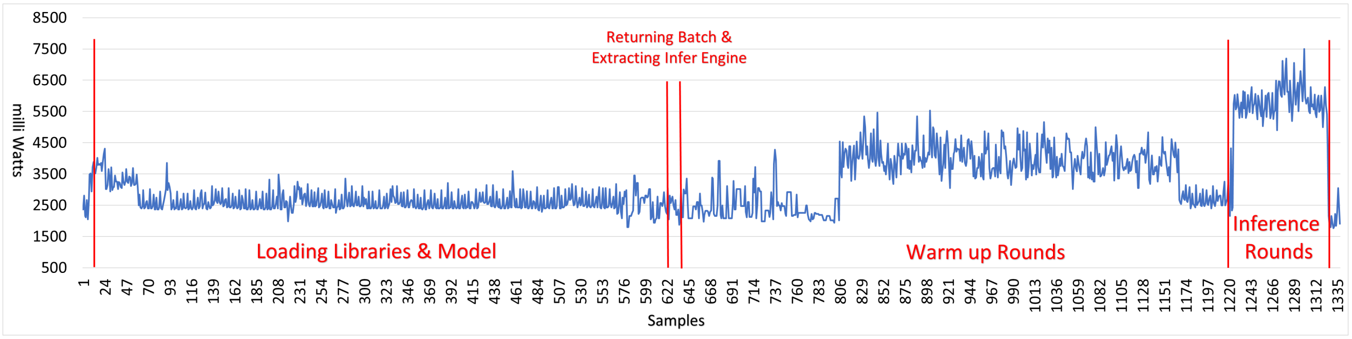


Figure 7: Power consumption: Jetson Nano System running the MobileNetV2 with TensorRT FP16. The *y axis* is the instantaneous power (in milli Watts); *x axis* is the time (in samples)

Batch, **3** Warm Up Rounds, **4** Real Rounds. The Loading Model step uses a considerable slice of the total experiment time, keeping energy consumption low and constant over time. The second stage is tiny time-wise, with no sudden change in power consumption.

The warm-up rounds step shows relevant results. As reported in the literature [20], GPU-equipped edge devices can suffer from cold start-related issues when the system setup overhead time dominates. These experimental rounds aimed at avoiding possible effects of cold start in assessing performance and power consumption. The power consumption remains relatively constant, even decreasing, to the point where cold start issues are left behind. At the end of this stage, a significant spike in memory usage is observed in the last warm-up interactions (Figure 8, sample 339 onwards). At the end of the experiment, the inference rounds pick up where the warms-up ones left off. While the GPU hardware runs at optimal performance with the cached data, the memory consumption remains constant until the end of the experiment. The TensorFlow average power consumption is $3386mW$ and $6125mW$ across the whole experiment and inference phase specifically. Such a promising result led to acceptable performance-watt, consisting of $12.10FPS/W$ across the entire experiment and $6.69Fps/W$ in the inference phase. The results demonstrate the device's capabilities for AI edge computing.

4.1.2 TensorRT FP32. In the first two algorithm steps (Figure 6), power consumption follows a similar pattern to the base case using float point 32 bits quantisation (TRT-FP32 model). However, the first stage lasts much longer (doubled number of samples), indicating that the device has more difficulty loading the TensorRT engine than the base model experimental scenario. The extra time is reported in TensorRT-related technical user forums, as TensorRT forces the loading of entire libraries into RAM to function properly. It imposes considerable overhead (i.e. "higher tax") on the overall power consumption [3]. The system tax could be diluted over a larger system uptime. The TRT-FP32 model causes a more variable power consumption during the third stage with warm-up rounds. This creates a power consumption higher than the one at the beginning but a similar consumption to the base experiment. That may be related to the inference phase being faster through the optimised models. The cold start issues are mitigated earlier than in the base experiment. In the actual rounds (FPS analysis) step, the power consumption results are compressed in a shorter time, supporting the improved image throughput obtained from the optimised models. Reaching 3185 mW across the experiment and 5711 mW in the inference step, the optimised model outperforms the base one but by a small margin. The enhanced experiment's performance-watt (FPS/W) already has a superior outcome compared to the base model. The TRT-FP32 experiment achieves 25.74 FPS/W and 14.35 FPS/W in the inference step.

4.1.3 TensorRT FP16. The results of TensorFlow floating-point 16 bits (TRT-FP16) show a profile similar to that of the TRT-FP32 experiment 7. However, the power consumption remains relatively low during the first stage, when the TensorRT library is loaded. The second stage does not present any significant change, primarily because of the stage's reduced time. The third stage follows a similar pattern to the previous experiment, exhibiting a consumption plateau mainly after the cold start issues are resolved. Finally, the last stage shows a high plateau for consumption due to the increased use of GPU resources during inference. The TRT-FP16 model outperforms the base experiment by a small margin. It reaches an average consumption of 3229mW and 5860mW during the inference stage. This model's performance surpasses the base model but lags behind the TRT-FP32 model. However, the performance-watt reached 14.33 FPS/W, a negligible difference between the optimised model experiments.

4.2 Memory Consumption

This section reports on the memory usage during the three experiments running models on the Jetson Nano device. The logger (running in parallel) aggregated the RAM usage to make the comparisons between the models validated (TensorFlow base, TRT-FP32 and TRT-FP16). Figure 8 reports on the first experimental results using the MobileNetV2 base model. From the very beginning, the process of loading the model into the Jetson Nano platform already consumes a considerable amount of memory. The next spike in consumption occurs during the inference process. Memory consumption reached a peak of 93.2% (Table 2).

Memory consumption results suggest that despite the differences in speed, all model versions exhibit a consistent memory usage pattern (Figure 9). However, the optimised versions (TRT-FP16 and TRT-FP32) take substantially longer time to consume memory space. They also

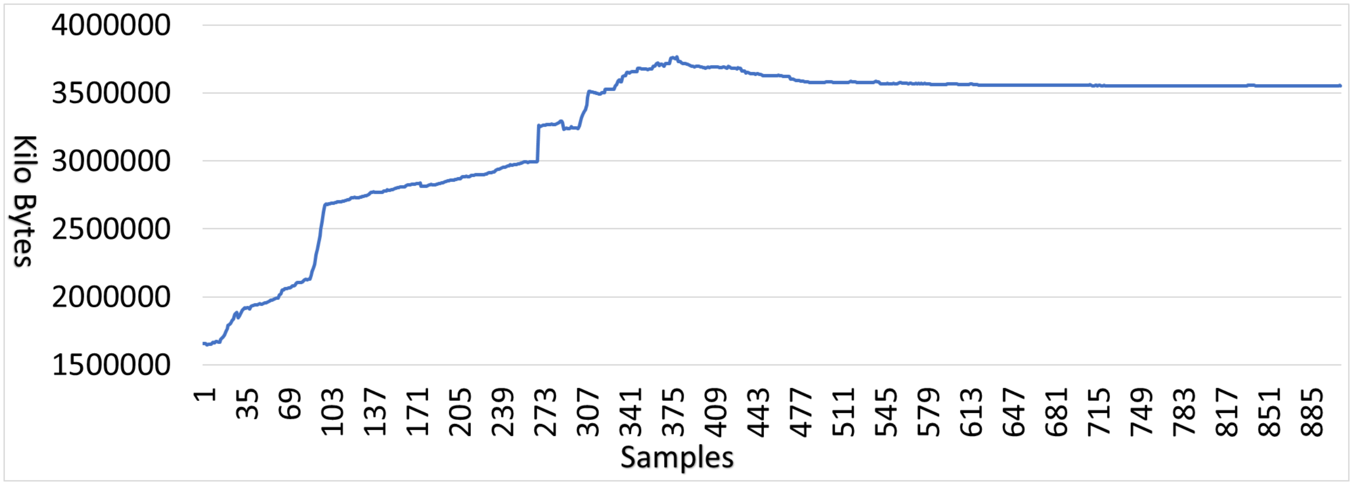


Figure 8: Memory usage: MobileNetv2 base model embedded in Jetson Nano.

exceeded the memory limits in several instances of the experiment. In such cases, we discarded data because the inference rounds could not be fully completed. This shows that the optimised versions are memory-hungry despite reaching image transfer rates higher than the base model. TRT-FP16 and TRT-FP32 reach 99.5% and 98.8%, respectively. It is essential to carefully design based on the application requirements for the intended project. The feasibility of using Tensor RT should also be considered case-by-case in projects, as the embedded systems are typically low in resources, including memory and processing.

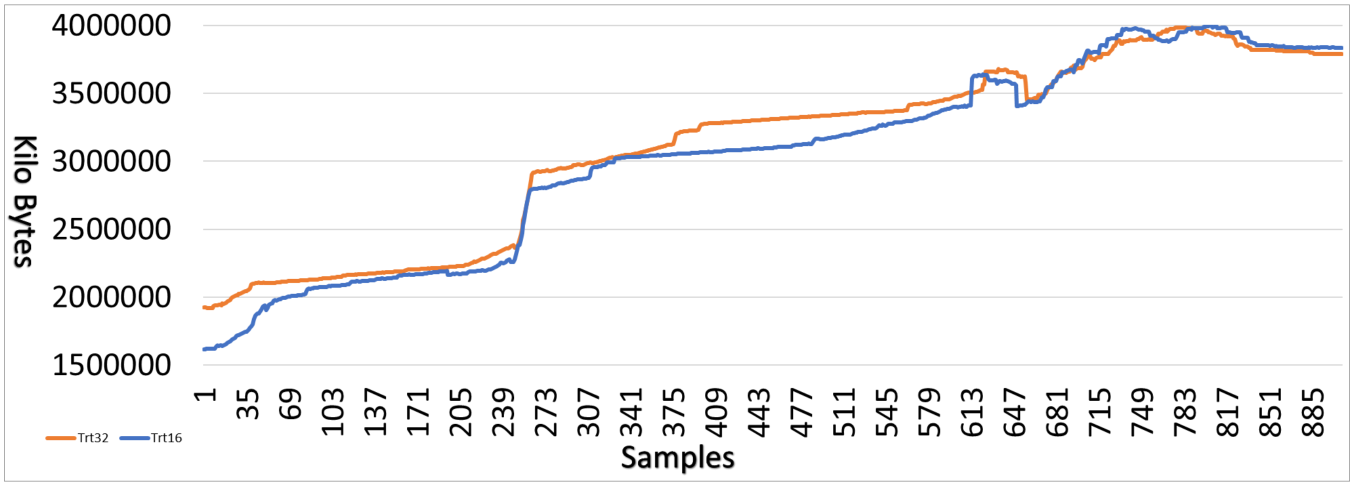


Figure 9: RAM consumption in the JETSON NANO device over the two later experiments. Using the optimized versions of the DDN model, TRT-FP32 and TRT-FP16

5 CONCLUSIONS AND FUTURE WORK

The present work attempted to answer the research questions: (RQ1) What is the performance and accuracy impact of taking cloud-based models to resource-constrained devices at the network edge? (RQ2) What is the power footprint in running machine learning classifiers in edge desk-based microscopic image analyser devices?

The experimental work evaluated possible trade-offs between model performance, accuracy and resource usage (RQ1). TensorRT engine optimised a MobileNet model trained in the cloud (base model), resulting in a model fine-tuned for an embedded edge system (NVIDIA Jetson Nano). Optimised model versions were evaluated using float-point quantisation of 16 and 32 bits.

The Jetson Nano platform also deployed model versions for inference purposes. Evaluation results of power-consumption profile, resource usage, classification performance and image throughput suggest that TensorRT is an efficient option for machine learning edge computing

(RQ2). TensorRT boosted the performance-watt ($6.69 FPS/W$, $14.35 FPS/W$ and $14.33 FPS/W$) for the base, TRT-FP32 and TRT-FP-16 models during the inference stage, respectively), and both optimised model versions doubled the image processing throughput and inference speed from 41 FPS (base model) to 82 FPS (TRT-FP32) and 84 FPS (TRT-FP16). However, application developers following these guidelines should take caution as the observed high memory usage presents a system limitation that needs to be investigated in future work.

ACKNOWLEDGMENTS

This work is part of the ASTRAL (All Atlantic Ocean Sustainable, Profitable and Resilient Aquaculture) project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 863034.

REFERENCES

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

[2] M. Braque, L. De Nul, and A. Petridis. 2021. *Industry 5.0 : towards a sustainable, human-centric and resilient European industry.* European Commission, Directorate-General for Research and Innovation, Publications Office, Brussels. https://doi.org/doi/10.2777/308407

[3] Michael Buch, Zahra Azad, Ajay Joshi, and Vijay Janapa Reddi. 2021. Ai tax in mobile socs: End-to-end performance analysis of machine learning in smartphones. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).* IEEE, 96–106.

[4] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. 2020. An overview on edge computing research. *IEEE access* 8 (2020), 85714–85728.

[5] Donghyeon Cho, Yu-Wing Tai, and In So Kweon. 2018. Deep convolutional neural network for natural image matting using initial alpha mattes. *IEEE Transactions on Image Processing* 28, 3 (2018), 1054–1067.

[6] FAO (Food and Agriculture Organization). 1997. Bangkok FAO Technical Consultation on Policies for Sustainable Shrimp Culture.

[7] Wei Ge, Jia Sun, Yihang Xu, and Hao Zheng. 2021. Real-time Object Detection Algorithm For Underwater Robots. In *2021 China Automation Congress (CAC).* IEEE, 7703–7707.

[8] Bruna Guterres, Sara Khalid, Marcelo Pias, and S Botelho. 2022. A data integration pipeline towards reliable monitoring of phytoplankton and early detection of harmful algal blooms. In *NeurIPS 2021 Workshop Tackling Climate Change with Machine Learning*, Vol. 2021. NeurIPS.

[9] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. 2019. Edge computing: A survey. *Future Generation Computer Systems* 97 (2019), 219–235.

[10] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. 2018. A Survey on Deep Learning: Algorithms, Techniques, and Applications. *ACM Comput. Surv.* 51, 5, Article 92 (sep 2018), 36 pages. https://doi.org/10.1145/3234150

[11] V Rajaraman. 2014. Cloud computing. *Resonance* 19, 3 (2014), 242–258.

[12] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 4510–4520.

[13] Mahadev Satyanarayanan. 2017. The emergence of edge computing. *Computer* 50, 1 (2017), 30–39.

[14] Daniel Seichter, Mona Köhler, Benjamin Lewandowski, Tim Wengefeld, and Horst-Michael Gross. 2021. Efficient rgb-d semantic segmentation for indoor scene analysis. In *2021 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, 13525–13531.

[15] Dmitrii Shadrin, Alexander Menshchikov, Andrey Somov, Gerhild Bornemann, Jens Hauslage, and Maxim Fedorov. 2019. Enabling precision agriculture through embedded sensing with artificial intelligence. *IEEE Transactions on Instrumentation and Measurement* 69, 7 (2019), 4103–4113.

[16] Omais Shafi, Chinmay Rai, Rijurekha Sen, and Gayathri Ananthanarayanan. 2021. Demystifying TensorRT: Characterizing Neural Network Inference Engine on Nvidia Edge Devices. In *2021 IEEE International Symposium on Workload Characterization (IISWC).* IEEE, 226–237.

[17] Lei Su, Xingang Yang, Boyuan Cao, Yun Wang, Xinjia Li, and Wenlian Lu. 2021. Development and application of substation intelligent inspection robot supporting deep learning accelerating. In *Journal of Physics: Conference Series*, Vol. 1754. IOP Publishing, 012170.

[18] Lei Tao, Tao Hong, Yichen Guo, Hangyu Chen, and Jinmeng Zhang. 2020. Drone identification based on CenterNet-TensorRT. In *2020 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB).* IEEE, 1–5.

[19] Mark L Wells, Vera L Trainer, Theodore J Smayda, Bengt SO Karlson, Charles G Trick, Raphael M Kudela, Akira Ishikawa, Stewart Bernard, Angela Wulff, Donald M Anderson, et al. 2015. Harmful algal blooms and climate change: Learning from the past and present to forecast the future. *Harmful algae* 49 (2015), 68–93.

[20] Rongjie Yi, Ting Cao, Ao Zhou, Xiao Ma, Shangguang Wang, and Mengwei Xu. 2022. Understanding and Optimizing Deep Learning Cold-Start Latency on Edge Devices. *arXiv preprint arXiv:2206.07446* (2022).

[21] Wenbin Yue, Zidong Wang, Hongwei Chen, Annette Payne, and Xiaohui Liu. 2018. Machine learning with applications in breast cancer diagnosis and prognosis. *Designs* 2, 2 (2018), 13.