UNIVERSIDADE FEDERAL DO RIO GRANDE - FURG

CENTRO DE CIÊNCIAS COMPUTACIONAIS

PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CURSO DE MESTRADO EM ENGENHARIA DE COMPUTAÇÃO

Master Dissertation

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

Lucas Caetano Meireles Pereira

Master Disstertation presented to the Programa de Pós-Graduação em Computação of the Universidade Federal do Rio Grande - FURG, in partial fullfillment of the requirements for the degree: Master in Computer Engineering

Advisor: Prof. Dr. Nelson Lopes Duarte Filho

Co-advisores: Prof. Dr. Marcelo de Rita Pias

Prof. Dr. Paulo Lilles Jorge Drews Junior

Rio Grande, 2023

Ficha Catalográfica

P436t Pereira, Lucas Caetano Meireles.

The not-so-easy task of taking heavy-lift ML models to the edge: a performance-watt perspective / Lucas Caetano Meireles Pereira. – 2023.

93 f.

Dissertação (mestrado) – Universidade Federal do Rio Grande – FURG, Programa de Pós-Graduação em Computação, Rio Grande/RS, 2023.

Orientador: Dr. Nelson Lopes Duarte Filho.

Coorientador: Dr. Marcelo de Rita Pias.

Coorientador: Dr. Paulo Lilles Jorge Drews Junior.

1. Eficiência energética 2. Computação de borda 3. Inteligência artificial 4. Fitoplâncton I. Duarte Filho, Nelson Lopes II. Pias, Marcelo de Rita III. Drews Junior, Paulo Lilles Jorge IV. Título.

CDU 004

Catalogação na Fonte: Bibliotecário José Paulo dos Santos CRB 10/2344

Master Dissertation

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

Lucas Caetano Meireles Pereira

Banca examinadora:

Universidade Federal do Rio Grande
Centro de Ciências Computacionais
Programa de Pós-Graduação em Computação
Curso de Mestrado em Engenharia de Computação

DISSERTAÇÃO DE MESTRADO

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

Lucas Caetano Meireles Pereira

Banca examinadora:

gov.br Documento assinado digitalmente
HELIO CRESTANA GUARDIA
Data: 26/01/2023 17:04:07-0300
Verifique em https://verificador.iti.br

_____
Prof. Dr. Helio Crestana Guardia (UFSCar)

Eduardo Nunes Borges:00057035067
Assinado digitalmente por Eduardo Nunes Borges:00057035067
ND: CN=Eduardo Nunes Borges:00057035067, OU=FURG - Universidade Federal do Rio Grande, O=ICPEdu, C=BR
Razão: https://pessoal.icpedu.rnp.br/public/verificar-assinatura
Localização: Rio Grande - RS
Data: 2023.01.29 18:46:40-03'00'
Foxit PDF Reader Versão: 12.1.0

_____
Prof. Dr. Eduardo Nunes Borges (FURG)

gov.br Documento assinado digitalmente
NELSON LOPES DUARTE FILHO
Data: 24/01/2023 18:21:22-0300
Verifique em https://verificador.iti.br

_____
Prof. Dr. Nelson Lopes Duarte Filho (FURG)
Orientador

ACKNOWLEDGEMENTS

First, I would like to express my deep gratitude to my advisers, for their patient guidance, for the criticism of my work and for the tips in writing my dissertation. Second, I extend my acknowledgement to the entire ASTRAL project team. We all work together, aiming for a world where technology is where it is needed. In particular, I want to thank three members of this wonderful team. Bruna Guterres, Amanda Mendes, and Kaue Sbrina, thank you for all your help, support and companionship during these months when we worked closely together.

Also, I want to thank my love, Leticia. Your love and encouragement allow me to walk this path and complete this cycle. I would never finish my work without you by my side.

RESUMO

PEREIRA, Lucas Caetano Meireles. **The not-so-easy task of taking heavy-lift ML models to the edge: a performance-watt perspective**. 2023. 92 f. Disstertation (Master) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande - FURG, Rio Grande.

Edge Computing é um novo paradigma de desenvolvimento que traz poder computacional para a borda da rede e para o usuário final, por meio de serviços inteligentes inovadores. O paradigma permite que aplicativos sensíveis à latência sejam colocados onde os dados são criados, com satisfação assim a sobrecarga de comunicação e com a segurança, a mobilidade e o consumo de energia. Existe uma conexão de aplicações que se beneficiam desse tipo de processamento. Em particular a classificação de imagens no nível microscópico. A escala e magnitude dos objetos para segmentar, detectar e classificar são muito desafiadoras, com dados coletados usando ordem de grandeza em extensão. O processamento de dados necessário é intenso e a lista de desejos dos usuários finais nesse espaço inclui ferramentas e soluções que cabem em um dispositivo limitado. Levar modelos de classificação inicialmente construídos na nuvem para dispositivos de análise de imagem baseados em mesa é uma tarefa difícil para desenvolvedores. Este trabalho analisa as limitações de desempenho e os requisitos de consumo de energia na incorporação de modelos de classificação, baseados em aprendizado profundo, em um dispositivo representativo de Edge Computing. Particularmente, o conjunto de dados e os modelos de levantamento pesado exploratório no estudo de caso são imagens de fitoplâncton para detectar a antecipação de algas nocivas (HAB) na aquicultura nos iniciais. O trabalho utiliza modelos de aprendizado profundo treinados para classificar de fitoplâncton e os implantar na borda. O *modelo base*, aprimorado em uma forma base juntamente com opções otimizadas, é mantido a uma série de experimentos de estresse do sistema. O perfil de consumo de energia e desempenho ajuda a entender a limitação do sistema e seu impacto na tarefa de classificar a imagem de grau microscópico.

Palavras-chave: Power Efficiency, Edge Computing, Artificial Intelligence, Phytoplankton.

ABSTRACT

PEREIRA, Lucas Caetano Meireles. **The not-so-easy task of taking heavy-lift ML models to the edge: a performance-watt perspective**. 2023. 92 f. Thesis (Master) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande - FURG, Rio Grande.

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 small 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 models 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.

Keywords: Phytoplankton, Light-scattering, Fluorescence, Flow Cytometry .

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

HAB	Harmful Algae Bloon
IoT	Internet of Things
AI	Artificial Inteligence
ML	Machine Learning
DL	Deep Learning
NN	Neural Network
ANN	Artificial Neural Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network
CNN	Neural Network
RNN	Recurrent Neural Network
GAN	Generative Adversial Network
ReLu	Rectified Linear Units
FC	Flow Cytometry
IFC	Imaging Flow Cytometry
FPS	Frames Per Second
ACC	Accuracy
TRT-FP32	TensorRT Floatpoint 32 Bits
TRT-FP16	TensorRT Floatpoint 16 Bits

SUMMARY

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. The evolution of ML-powered technologies has advanced in gigantic steps recently [67]. Deep learning models are at the centre of such advancements. Concurrent changes in embedded devices connected to the network and new requirements in the Internet of Things (IoT) create the need for new computational paradigms. On one side, Cloud Computing has a host-centric perspective, with most services present at the network core. And it processes all requests from the network centre to the edge, where many resource-constrained IoT devices are deployed. Although useful in its own right, the cloud approach could be more optimal when complying with service level agreements of real-time, privacy-first and low-power end-user services [10]. 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 for data processing. It also enhances where Cloud Computing needs the most, latency, privacy, and proximity to the end user.

Artificial Intelligence (AI) applications have grown in recent years, both in number and complexity [52, 12, 71, 55]. As a result, more data is available for advanced big data processing to produce timely and relevant insights. Techniques derived from these concepts are increasingly used for business, healthcare, military, and the industry [33, 57, 42, 16]. For applications in these fields and many others, it is crucial to analyse available data and provide valuable insights in the form of accurate predictions. And as technology improves, the role of artificial intelligence in society expands. In this regard, the significant part of artificial intelligence, machine learning and deep learning is booming. More specifically, deep learning has experienced the most remarkable breakthroughs among many AI subareas [49]. 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, including 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 [53]. This feature allows for fast data processing and real-time response time [30]. Edge devices come in different shapes, sizes, and capacities. In a more traditional context, a router which connects public networks to the internet is an example of an edge computing device. In addition, specialised edge computing devices exist, including the Internet of Things (IoT), Industrial IoT, robots, and intelligent machines. Such edge devices can achieve a broad range of functions that leverage the communication network capabilities to lower the power consumption overhead while sustaining real-time performance (i.e. *performance-watt*). The edge computing application communicates still over the internet to the services running on a cloud server. This way, it builds a system where processing power is accessible from everywhere via the web [50]. For instance, an autonomous vehicle that provides entry to a cloud platform through digital twin technology or a network capitalises on the edge-cloud ubiquitous connection.

Edge devices have grown in popularity, reaching many industrial applications, 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 real-world tasks. For example, the digital twinning in the upcoming Industry 5.0 [4] 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 (e.g. audio signals, video footages, sensor time series). To make sense of this data promptly, edge-device twins can deliver expected outcomes in climate-resilient production applications.

Technology-driven sustainable agriculture and aquaculture are compelling applications for edge device computing. Such applications require data integration, edge processing and embedded machine learning as part of a strategy to deploy low-cost, real-time digital twin sensory 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 [18]. Climate change is creating environmental conditions for the worldwide surge in harmful algal blooms (HABs) [68]. Such harmful phytoplankton biomass seriously impacts aquaculture with undesirable events of oxygen depletion. It might lead to the mortality of aquatic organisms in uncontrolled scenarios. For instance, intense aquaculture farming can face production loss in up to 30 minutes when no mitigation action is taken.

Edge computing can 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). Early HAB detection is essential to react and intervene in the aquaculture process. However, 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 benchtop and desk-based image analysers equipped with self-contained ML models capable of producing image classification results within seconds (i.e., just the time for an inference).

Edge computing benefits a considerable number of other applications as well. Researchers and technology developers 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 the 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 [6] is expected.

1.1 Artificial Intelligence

AI is the crucial aspect that motivates edge processing capabilities to enable emerging applications. For simplicity, we define artificial intelligence as a computer science branch focused on bestowing human-like intelligence on machines. The primary goal of AI is to develop autonomous machines that can think and act like humans, perceiving the environment and taking actions to maximise the chance of success in some goal [46]. These machines can mimic human behaviour (to some extent) and perform tasks through learning and problem-solving capabilities. According to Nilsson and Nilsson [43] AI is related to the intelligent performance of artefacts. Most AI systems simulate natural intelligence to solve complex problems. However, this should be considered with a *pinch of salt* because the superhuman, singularity type of narrative is always dangerous and can easily disconnect from reality. Because of its application-specific goal, we argue that edge computing specificity reassures the down-to-earth AI rather than the more science-fiction view portrayed by many media venues targeting the general public members.

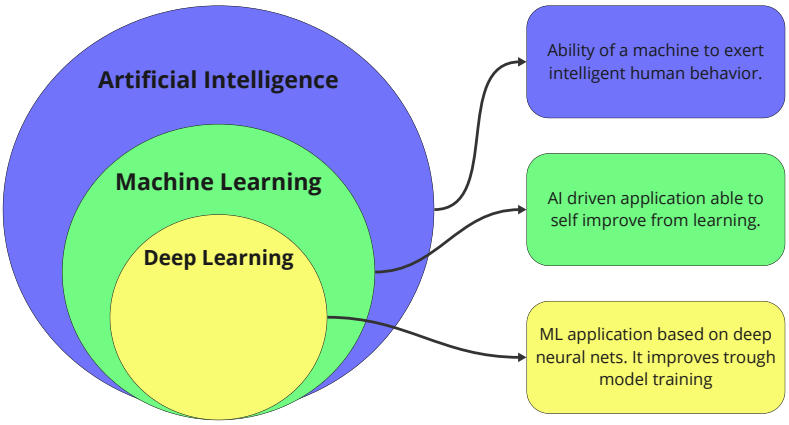

Figure 1: Hierarchical View: Artificial Intelligence, Machine Learning and Deep learning.

Machine Learning is an approach to materialise AI in practical applications. Accor-

ding to Arthur Samuel in 1959, it gives ”computers the ability to learn without being explicitly programmed.”[41]. It consists of using algorithms to analyse data, learn from it and predict or decide something in the world. Instead of manually writing sequences of software code with a specific set of instructions (a.k.a rules) to perform a particular task, the machine is ”trained”with large volumes of data and algorithms that give it the ability to self-learn the rules needed to complete a task.

ML is used in data pattern recognition and prediction as a computational tool for data analysis. It is employed mainly in computational routines where building and programming explicit algorithms with good performance are hard, slow or often intractable. ML requires complex math and powerful algorithms to achieve the desired functions and results. It also needs access to vast amounts of data (both structured and unstructured) as input for learning to predict future scenarios. Traditionally, the training of the ML models relies on large amounts of data. As we provide more data samples from the target data distribution, the model improves its generalisation capabilities to the entire distribution. An ML model training component tries to fit its internal parameters to the data provided by iterating over and over to minimise an error objective function. The optimisation objective in a typical supervised learning classification is the difference between the model's output (predictions) and expected values (ground truth).

There are four types of standard machine learning algorithms: unsupervised, supervised, semi-supervised, and reinforcement learning [20]. Two of the most widely adopted methods are supervised learning and unsupervised learning. Hence, most machine learning is supervised learning, followed by unsupervised learning. Semi-supervised and reinforcement learning are two other approaches that have gathered pace in the last few years, fuelled by autonomous systems learning [46].

- **Supervised Learning** models are trained using labelled data as a correct input that guides a known desired output. The algorithm is fed with an input set alongside associated outputs. It learns by comparing its actual model output with the correct outputs in search of potential errors (discrepancies). The model parameters are updated to minimise errors between the real output and the desired one.

- **Unsupervised Learning** is used when the data lacks known labels or some annotation. The algorithm uses unlabelled data to discover patterns from the data on its own. The goal is to explore the data and find some structure within it. Unsupervised learning works well on transactional data. The systems can identify hidden features from the input data provided. For example, it can find the main attributes that separate similar objects into distinct categories.

- **Semi-supervised Learning** is used for the same applications as supervised learning, but it uses both labelled and unlabelled data for training. Typically it uses a

small amount of labelled data with a large amount of unlabelled data because unlabelled data is less expensive and takes less effort to acquire. Recent work in graph neural networks (GNNs) explores this approach to predict links between entities modelled as a graph [39]

- **Reinforcement Learning** uses a reward system to train its models. The algorithm discovers which actions yield the most significant rewards through trial and error. It functions based on three primary components: the agent (the learner or decision maker), the environment (everything the agent interacts with) and actions (what the agent can do). The objective is for the agent to discover the parameter space which maximises the expected reward in a given amount of time. As a result, the agent will reach the goal much faster by following a reasonable strategy. The purpose of reinforcement learning is to find the best strategy through step-wise learning.

Machine learning applications are optimal in data-intensive but are often hampered when scarce data is available. Few-Shot Learning (FSL), a sub-area of machine learning, is an alternative to tackle this issue. FSL is about classifying new data with only a few training samples. It mainly works as a form of supervised information. FSL takes advantage of the fact that modern computer vision models can work exceptionally well with relatively few training samples. Using prior knowledge, FSL can rapidly generalise to new tasks containing only a few examples with supervised information [37, 39].

An algorithmic approach to implementing ML is artificial neural networks (ANNs). Inspired by our understanding of human brain biology, ANNs (or simply neural networks) are represented in the information space as an arrangement of neurons and interconnections. However, unlike a biological brain, where any neuron can connect to nearby ones, artificial neural networks are structured in layers where connections create a reasonable direction for data propagation. As data is fed to the network's first layer, individual neurons transfer it to the second layer, and so the processes continue up to the result outputted from the last network architecture layer. Neurons assign weight to the internal information based on its suitability for the task. The aggregate of the weights determines the final result.

Deep learning is the training of neural networks that contain more than a single hidden layer [14]. Such network architectures, known as Deep Neural Networks (DNNs), have a similar base structure to classic ANNs. The network architecture comprises an input layer that receives application data and a sequence of internal (hidden) layers that work as sophisticated filters to find specific features from the data. Within the hidden layers, a cascade of non-linear processing units (the neurons) extract features from the data. Each successive layer uses the output from the previous layer as input. The last layer then provides the expected outcome, which is application-specific. For instance, in an image classification task such as phytoplankton classification, the output is the expectation that

a given input is associated with a given phytoplankton species.

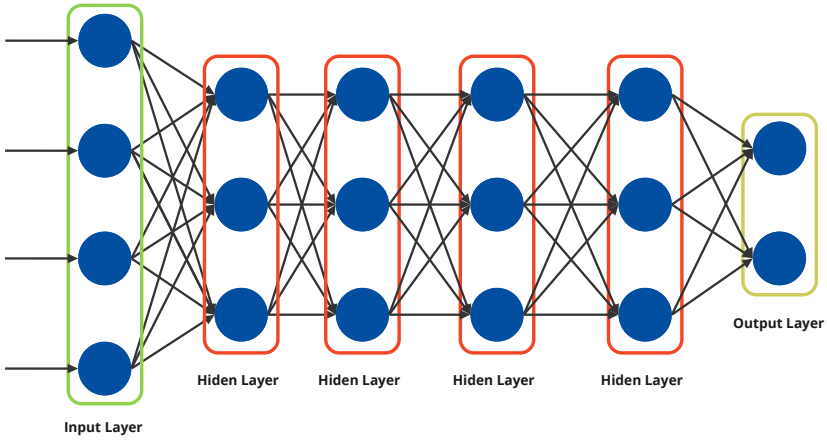

Figure 2: Deep Neural Network (DNN): typical structure of neurons organised into interconnected layers. Several hidden layers are typically used in DNN architectures.

Machine learning algorithms usually require structured data, whereas deep learning networks can work with an enormous amount of structured and unstructured data. The learning algorithms used alongside DNNs can be supervised or unsupervised. This technique produces an abstract, compressed, distributed representation of the input data. During the training process, the neural network updates its internal weights, minimising error to obtain the best possible abstract representation of the input data. The representation is abstract because it is based on learning multiple levels of features. Higher-level features are derived from lower-level features. This representation of the input data present over the network is used to produce results for tasks such as classification, detection, and reconstruction of objects, people or even 3D scenes in virtual reality and digital twinning.

A commonly used type of DNN relevant to this dissertation work is the architecture of a Convolutional Neural Network (CNN). CNNs employ special filters to the data based on convolutions that act as a data summarisation tool hence feature extractors. Also, convolutional layers reduce the model complexity with less interconnection between neuron units compared to traditional ”shallow”neural networks (i.e. fully connected neurons). CNNs primarily helped to develop automated image-specific feature extraction, which has been used extensively for image-based ML tasks [47]. Several possible operations can be applied to the data as it traverses the network architecture ranging from simple scalar products to specialised non-linear filtering functions. Also, a resemblance of the human visual system is exploited in the original CNN proposals where initial layers extract more abstract features (e.g. vertical lines), leaving for the last row of layers to specialise for the output.

1.2 Edge Computing

Edge Computing, as a concept, has strong roots in the telecom sector where the *edge of the network* forwards traffic from the user access networks to the core network. Besides forwarding data packets, the edge routers can also process the data for several applications, including security (e.g. deep packet inspection for firewall), data minimisation, and traffic forwarding optimisation (e.g. quality of service). Besides this strict and traditional view of edge processing, an emerging form of edge computing considers the edge at the end-user application perimeter. This new perspective works hand-by-hand with the cloud computing paradigm to take advantage of advanced embedded devices that can store and process the data gathered from the applications locally. This solution offers a unique approach to the privacy-preserving handling of user data, allowing citizens to truly own and fully control their data which is often relayed to commercial cloud service providers. Although data privacy is not the core topic of this dissertation, edge computing provides a foundation upon which a new concept of a citizen-centric, data-driven service model can be viably designed.

The Cloud computing paradigm can still run part of its applications on resource-constrained devices (e.g. Amazon Alexa system). The device communicates over the internet to the cloud services via well-specified APIs, thus creating a comprehensive approach where processing power is accessible via the web everywhere [50]. Cloud computing dominated the way computing has been done in the last decade and enabled engineers to circumvent many problems related to resource unavailability. It also retains advantages in economy of scale where centralisation can significantly reduce operational costs [53]. Such technological and economic dynamics led to the consolidation of computing capacity into multiple large data centres.

The departure from centralised processing in the cloud brings many benefits, particularly considering the cost reduction of often prohibitive data communication transfers. Edge computing is by no means a competing approach to the Cloud-based systems, but it is an improved complement to centralised data processing. By reallocating resources, it enhances delay-sensitive applications [30]. The Internet of Things currently exploits the cloud and edge computing spaces. Traditional IoT applications use low-resourced device systems where the network connects the sensors close to the data of interest and the end user to a centralised server. Such a server performs the necessary data processing for the application. However, the boom of IoT applications led to a massive increase in small devices and the steady increase of computational power within off-the-shelf embedded devices. Edge computing is a natural complement to the cloud, but a few characteristics still make a clear cut between those two processing paradigms [30]. The list below elaborates on such vital aspects:

- **High geographical density:** Edge Computing moves the data storage and proces-

sing to the network's edge. It brings services close to the end user by deploying complex networks of mid-range embedded devices. This dense infrastructure improves several applications by providing a real-time response in large-scale IoT deployment [1, 31] and better accuracy in some data-intensive applications [29, 17].

- **Mobility:** Edge computing also supports mobility more efficiently and organically than cloud-based systems. At the Edge, the local hardware performing the operations has a much smaller dependence on a central system.

- **Proximity:** As the computational resources are closer to the user-generated data, it becomes easier to develop user-customised systems. In a complementary way, the computer service provider can take advantage of the mobility and proximity aspect to acquire user contextual data (user, location, environment inserted). This type of data allows the provider to enhance their product or service.

- **Heterogeneity:** this refers to the surplus of different platforms, architectures, infrastructures, computing resources and communication technologies. Heterogeneous edge computing allows the customisation of the end device software and hardware, while such details are commonly transparent. It also includes network heterogeneity, where the diversity of communication technologies enriches edge computing reliability.

- **Low Bandwidth Cost:** In Edge Computing, the data is processed locally at the device, so there is no need to send raw data to a cloud computing data centre. As a result, it does not impose pressure on the available network bandwidth, which leads to an overall network-related cost reduction.

- **Response Time:** Other aspects of edge computing converge to the provision of services with appropriate response time for the application. While in a traditional Cloud-based application, the data is sent to central storage for processing, in edge computing, the processing (e.g. data reduction) happens locally. Edge computing can give insights into intelligent latency-sensitive applications that could, for instance, trigger the work of local actuators in real-time (or quasi-real-time).

Also, traditional cloud-based systems have to secure data transmission. Security is a significant challenge that needs to be addressed in edge and cloud computing. Risks and threats such as data loss and leakage must be appropriately considered [10]. Cloud systems depend on the network's capacity to connect the central server to the "dumb"data collector devices positioned with the end-user application. This raises important questions and possible vulnerabilities. Edge computing offers an opportunity for data minimisation and privacy compliance when the data is still on the end-user side. Nevertheless, the

connection between the local device hardware and the cloud server is security exploitable in any of the computing paradigms.

Malicious agents can gain access to sensitive data travelling on the network or deny service access by overloading the cloud server with other tasks and service requests. It is also important to emphasise that sensitive data is an issue that goes beyond its risk of disclosure to non-authorised third parties. In many cases, the application user provides personal data (say, medical sensor device) so that the contracted service is delivered efficiently and, in many cases, personalised. The edge principle of system proximity to the user cannot be neglected as a strong premise as privacy-preserving technology enable. Systems capable of local processing placed at the edge where the user sits can keep private data where it belongs. The stored data does not need to travel through the entire network to reach a cloud server. The data breach risk can be mitigated with this type of local storage and processing hence avoiding unauthorised access by rogue third parties.

1.2.1 Edge Intelligence

The continuous advances in AI efficacy, the expansion and sophistication of IoT devices and the Edge Computing paradigm updates create a coherent whole that unlocks a set of new perspectives in Edge Intelligence. DNNs and related ML infrastructure have experienced an increase in technology readiness level (TRL) in the last decade, achieving significant practical results in several areas. In addition, interconnected sensing and actuator systems (IoT devices) have penetrated many industrial sectors. This advance has driven the design of modern embedded systems that contribute with extended low-power computational power and heterogeneous communication systems to enable emerging intelligent systems based on deep learning platforms to be realised at low-resourced systems (e.g. mobile processors, mobile GPUs, a new generation of microcontrollers, etc.). Edge computing aggregates such advances by acting as a system-level catalyst.

The integration of AI and edge computing is a plausible pathway to bring intelligence to the end-user applications [73]. On the Edge Computing side, its goals are to coordinate a set of cooperating devices to perceive the environment and process the associated data close to the user application (e.g. IoT, robotics, enhanced reality). In contrast, AI is experiencing a trend of delivering sophisticated ML model results to users at the edge to benefit from low latency, proximity, and bandwidth usage. The coupling of AI and edge computing help many applications. First, aggressive penetration of more sophisticated embedded devices has generated a large volume of sensory multi-modal data, such as audio, images and video. Second, AI in the form of an acceptable tuned machine learning model is vital to high-quality data analysis for rapid decision-making.

Despite the advantages of AI-based Edge Edge Computing, there are challenges to be overcome. Machine learning models, especially Deep Learning ones, are complex computational artefacts. Such models can have several millions of parameters and use large

amounts of computational cycles to generate valuable results. Also, ML tend to be highly memory hungry and needs access to powerful processing units such as highly parallel vector machines (e.g. GPUs) to train models accurately and deliver model inferences.

It is critical and urgent to map such system bottlenecks, overheads, costs and "AI taxes"generated when taking these models to an edge device. Mitigating such edge device limitations to the ML model becomes a priority to support the ever-growing emerging ML applications that benefit the end user. Zhou et al. [73] presents an initial list of metrics that could be explored to evaluate the inference service of ML models at the edge:

- **Latency:** It refers to the time spent in the model inference process, including prepossessing, model inference itself, and data transmission. The present work focuses on the inference phase of ML models deployed at the edge and the likely performance and resource usage impacts.

- **Accuracy:** The average ratio of the number of input samples that receive the correct predictions from inference to the total number of input samples. This is a typical variant of ML model performance that should be fully considered when taking models to the edge. Also, some safety-critical applications at the edge (e.g. assistive medical robots, driverless cars) may require a higher level of reliability that can be fulfilled with additional model quality metrics.

- **Memory Usage:** The memory footprint of DNN models is a concern in the context of Edge Computing. Typically, a high-precision DNN model is accompanied by millions of parameters which can easily consume significant memory space. Unlike resourceful Cloud machines, memory availability is a bottleneck in embedded systems. The memory footprint is a non-negligible indicator that should be considered in DNN model optimisations.

- **Energy:** Running a DNN model is expensive and often time-consuming. While cloud-based systems have a dedicated power supply, edge devices tend to be battery-limited. In addition, the computational processing needs for DNN inferences account for additional energy consumption. It is crucial to ensure the energy efficiency of AI edge-based models to leverage the inherent advantages of integrating edge processing devices into cloud servers.

1.2.2 Federated Learning

It is important to highlight an emerging Edge Computing technique that covers the distributed training of DNN models. Federated Learning (FL) framework in Edge Computing provides a practical mechanism to implement parameter learning training across a set of networked devices. It enables Edge devices to collaboratively learn a shared prediction model while keeping all the training data on the device. FL works through a

decentralised strategy where client devices download a global DNN model from a server and train it with private data stored by the device, turning it into a local model. After that, all devices send the local model weights to the server to formulate a new global model. The federated devices perform these cyclic steps, further training the model [51]. Although FL is a promising approach for edge computing, the present work focuses solely on the inference phase of models deployed at the edge, and most importantly, it attempts to understand the performance and resource-related limitations.

1.3 The Case Study: Edge Computing for Water Quality Management

The H2020 ASTRAL[1] is a European Commission project focused on integrated multi-trophic aquaculture (IMTA) farming that will define, support and promote sustainable aquaculture production across the Atlantic. The project addresses sustainability along a strong Climate-Ocean-Food value chain. It links expected environmental risks to cost-efficiency and best practices of aquaculture production and food safety. This target is supported by technological innovations providing a significantly improved capability of observing and monitoring three main environmental water risks: harmful algae blooms, water pathogens and microplastic pollutants. This will lead to concrete recommendations on monitoring programmes and technological development and promote sustainability.

The ASTRAL project prioritises aquaculture due to its high potential for sustainable jobs and growth, specifically by focusing on new value chains for aquaculture production. As a result, the sector has constantly increased its percentage in the total fish production, reaching 53% of the total seafood production from fisheries and aquaculture in 2015 [60]. Furthermore, considering the potential of the human population to reach 10 billion by 2050 [15].

A relevant issue for aquaculture and water quality monitoring is the detection of HABs that can grow extremely fast, expanding into the whole area of the fishing farms. Detecting HABs at an early stage is essential to react and intervene accordingly. However, the state-of-the-art methods rely on late decisions based on the cloud-based processing of satellite images. Edge computing enables solutions closer to the water environment (in-situ monitoring). It brings the detection platform with self-contained models for local device processing and detection results in seconds (i.e. time of an inference).

The most traditional method for studying phytoplankton distribution and abundance is the microscopic examination of water samples or cell harvesting by filtration, which sacrifices information about the distribution of properties among the cells. Traditional microscopy allows high-resolution images and thus provides detailed inspection of individual phytoplankton at the expense of imaging throughput. The identification of species

[1] https://www.astral-project.eu/

is based on the morphological characteristics of the collected phytoplankton. However, manual microscopy testing techniques require specialists with rich knowledge. While useful for visual inspection, conventional microscopy lacks high throughput and analytic-level accuracy.

It is essential to bring emerging technologies to classify phytoplankton and predict HABs. Automated ML-based classification can bring many advantages to the early detection of HABs. Examples include automatically classifying individual organisms, counting individuals in each group, and critically analysing population variations. Over the years, DNN models have achieved impressive breakthroughs in many areas. Expanding its use for HAB detection through sorting and counting could lead to more complete warning systems in regions of interest. Once the model is trained, most of the specialised work is done, thus enabling fast and reliable sorting without the immediate need for an expert.

Edge computing can, in turn, significantly contribute to monitoring HAB-related events. Bringing ML models to the end user, in this case, aquaculture farmers is not easy. However, the computational paradigm of Edge Computing has the potential to enable such technologies to roll out in relevant environments. Traditional HAB detection methodologies rely on satellite images to identify visual cues, such as the red tide [72, 34]. However, farmers benefit from a portable benchtop device deployed directly on the field with sufficient intelligence to detect harmful algae blooms from phytoplankton species classification. This requires significant edge design advances to accommodate ML models on resource-restrained systems.

1.4 Data Science to Edge Deployment

A typical data science development workflow is described in Fig 3. The accuracy and performance are usually evaluated for the Data Science Phase, where the model is validated using different metrics and sets of testing data. However, the data scientist faces challenge to deploy the models created in highly resourced cloud servers into edge devices for some applications (e.g. driverless vehicles, image object detectors, personal voice assistants, and so forth). The case study explored in this dissertation - namely the microscope-level image classification of phytoplankton - also follows the steps presented in the diagram (Fig. 3). The **In-Situ** phase includes the base hardware and software of the phytoplankton sensor that acquire the images and deliver them to the deployed model. In the first step in this phase, textbfvideo capturing accounts for the main system component, followed by the essential software for **Frame Extraction** and **Specimen Segmentation** where the collected data is pre-processed before it is inputted into the models. Next, in the **Data Science** phase, data is handled to build and validate models for the application iteratively. The **Data integration** step is a process where datasets from different sources are prepared for model development and validation. During the **Model Training**, the different

model architectures (new or based on readily available architectures) are trained using the target dataset created in the previous step. Finally, in **Cloud Inference**, the models can be used in cloud-based applications. In cases of edge computing deployments, modes are tested in search of the best results in quality-related (typical) but also performance-watt metrics specifically for model selection in edge computing deployments.

In this sense, this work addresses the needs of a large number of researchers and developers who have either limited guidelines or sometimes follow ad-hoc principles on how to approach the *Edge Phase* of the workflow.

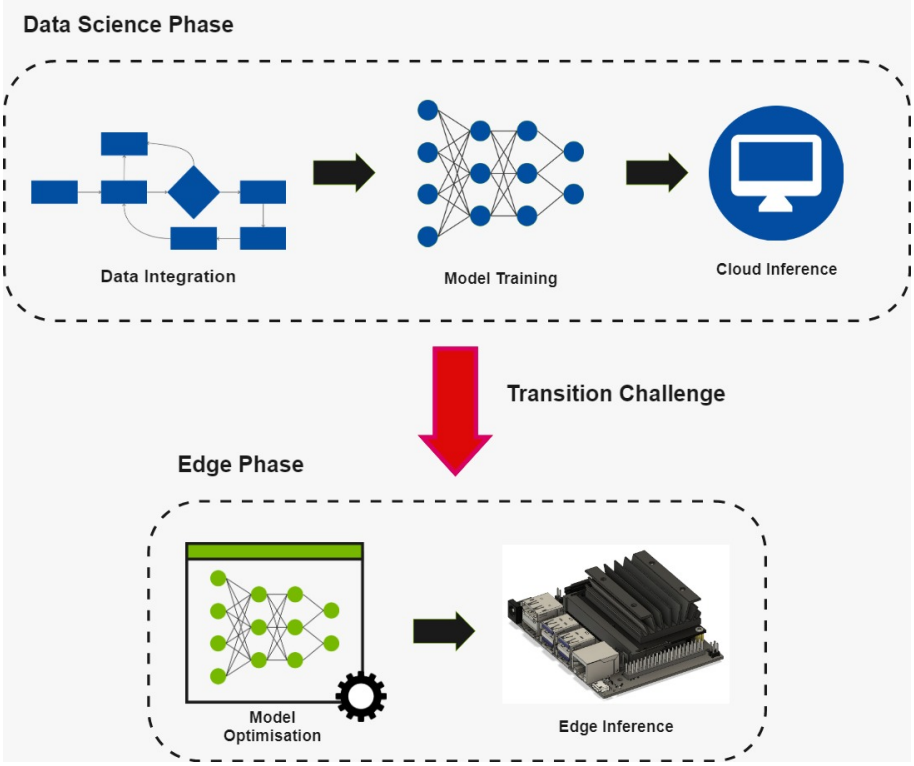

Figure 3: Data Science to Edge Deployment Workflow.

The **Edge** phase is crucial in the edge application development cycle. This phase consists of two steps. The **Model Optimisation** comprises specialised libraries (e.g. NVIDIA TensorRT) with cloud-trained models to generate optimised versions of the model.

The final phase, **Analytics**, consists of handling the results generated when deploying the models to edge devices. **Power and Performance** related data is continuously analysed as models are tested in the target embedded platform.

1.5 Research Questions and Objectives

Edge Computing can deliver high-speed ML models at the edge, closer to the application. For instance, the proposed case study benefits from a device that can detect HAB events quickly to minimise adverse economic and social effects. Provided the techno-

logy is low-cost, its penetration into aquaculture facilities may guarantee efficient and controlled fishing production as a green technology to help our planet's sustainability.

However, Edge Computing comes with a complex non-trivial price tag regarding ML models' computational capacity. As already mentioned, cloud-based ML models are cumbersome, power-hungry and have a high memory footprint. In contrast, Edge Computing relies on low to mid-range embedded devices, which have limited capabilities in terms of power, computational power and available resources. As a result, deploying such models into these devices is a real challenge.

The present work explores how feasible it is to bring complex DNN models to edge devices. The work addresses the following Research Questions (RQ):

- (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 microscopic-level image benchtop analyser devices? And what is the impact of edge device software libraries on the deployed models in terms of performance-watt?

1.5.1 Aim

This work aims to understand further the system limitations associated with the *transition challenge* of taking models trained in the resource-rich cloud environment (Data Science Phase) into resource-limited edge devices (Edge Phase). Fig. 3 illustrates the transition challenge.

The limitations in terms of performance and accuracy of embedded machine learning models will be validated with datasets and context for an environmentally sustainable application-driven scenario: water quality management in aquaculture.

1.5.2 Specific Objectives

This overall objective will be reached through the following specific objectives:

- **S01 - To investigate the accuracy of cloud-based models once deployed into edge devices [RQ1].**
  An ML classification model's accuracy (ACC) is among the most important metrics to measure model quality and performance. Thus, exploring how much of the model ACC can be affected as it is deployed to a resource-constrained device is crucial as part of a new set of guidelines. For instance, the accuracy discrepancy between cloud trained model and its optimised version for an embedded system (e.g. Jetson NANO) will be further explored.

- **S02 - To investigate possible performance gain (in image FPS) in model post-optimisation [RQ1].**
  As we deploy the models to the edge, an important aspect to analyse is the achievable image throughput in the device for each model and its variations. The model optimisation software (e.g. TensorRT) promises significant gains. However, performance in terms of FPS (frames per second) depends on the deployment, so exploring alternatives to enhance system performance is crucial. This work looks at the impact of the TensorRT optimisation software in the embedded system.

- **S03 - To conduct a deep analysis of system power consumption [RQ2]**
  The most critical gap in the literature this work seeks to fill in the preparation of a comprehensive power consumption study of selected embedded models. The experimental regime will be constructed in such a manner as to highlight the power consumption across the whole machine-learning data pipeline. A detailed analysis of each experiment step will be carried out for each selected model, enabling a thorough power consumption profiling.

- **S04 - To analyse the resources used [RQ2].**
  This work will explore how each model affects computational resources: memory consumption and CPU & GPU usage in an embedded device. To determine the model feasibility for edge device deployment, we should also investigate how much resources are used on average to fulfil an end-to-end model inference path.

- **S05 - To understand the performance-watt [RQ2].**
  Modern embedded system evaluation employs a two-dimensional analysis that integrates performance with power consumption. The performance-watt metric will be studied by considering simultaneously the image throughput and power consumption for the resources used.

A research programme was established to attain the objectives above with weekly meetings between the candidate and their supervisors. This dissertation text reports on this journey: Chapter 2 overviews concepts and critically discusses the related work. Chapter 3 proposes the methodology, including systems, experimental design and validation approaches. Chapter 4 reports on the experimental analysis results. Chapter 5 draws the main conclusions and explores areas for future work.

2 BACKGROUND AND RELATED WORK

This chapter serves two purposes. Firstly, it motivates the work by introducing key concepts in the application domain used as the guiding line for the technical work. It presents currently available devices in worldwide labs (commercial and open-source) that can be used for microscopic imaged-based assessment of water quality parameters. The work of this dissertation goes beyond this system's functionality in a mixed human-AI system for human empowerment. The user application analyst will have access to emerging Edge Devices embedded with intelligence for a trustworthy hybrid decision-support system. The second part of the chapter covers the concepts and key system aspects of a representative of such edge computing device that retains some processing power at a low-cost price tag. Finally, the chapter discusses the issues and gaps identified from the literature. The performance-watt profiling of such edge computing devices emerges as a first step yet crucial in understanding system limitations for novel intelligent edge computing applications. This explores some system limitations in taking machine learning models trained in a cloud-based environment with plenty of computational resources into a resource-constrained edge device for inference purposes.

2.1 Use Case Context, Traditional Standalone Devices

Several traditional data analysis applications can move part or the entire data processing to devices operating on the end-user premises at the very edge of the network. Because the application space to survey is enormous, we take a single application hoping it can represent a future edge computing application reasonably well. This is also motivated by the context of this work which develops under the European Commission-funded project ASTRAL [1]. In this initiative, FURG is the work package leader for developing a pool of intelligent advanced sensor technologies. The ASTRAL project is timely because it offers the playing field for the design of low-cost, edge-computing IoT and AI-based vision sensors for water quality management in aquaculture. This real-world context provides the necessary application requirements, common across almost all edge computing

[1] https://www.astral-project.eu/

use cases, particularly the quality of the machine learning models and the duo factor performance-watt for AI embedded in resource-constrained devices.

Water quality management in many areas is considered challenging, especially in cases with limited intervention time. The aquaculture farming explored in the ASTRAL project is undoubtedly one of them. Global food systems will be pressured to produce animal protein in large quantities [65]. Aquaculture's rapid and steady growth is necessary and should be boosted and refined whenever feasible. ASTRAL aims to promote aquaculture development by implementing innovative and resilient value chains. It assesses their circularity and sustainability to improve production control and monitor environmental hazards with innovative low-cost sensing and actuation technology.

A relevant issue for aquaculture and water quality monitoring is the detection of harmful species of phytoplankton that form what is called *algae blooms* by collective grouping together. Such grouped toxic phytoplankton (or just harmful algae blooms - HABs) can proliferate in the water environment in a way that threatens other living species.

Detecting HABs at an early stage is essential to react and intervene accordingly. However, the state-of-the-art methods rely on very late decisions based on the cloud-based processing methods for satellite images. In contrast, Edge computing enables solutions closer to the water environment (in-situ monitoring). It brings the detection platform with self-contained analysis models to the user's point of interest. The local device processing and detection results in seconds (i.e. time of an inference) are key features in applications such as in-land remote aquaculture, often with minimal power supply and network connectivity.

The diversity of algae blooms and their impacts present a significant challenge for those responsible for managing coastal resources such as wild fish and marine aquaculture. The extreme diversity in genus and species makes phytoplankton populations vary highly in size, shape and morphology [32]. As a result, the strategies needed to protect wild fauna, minimise economic and ecosystem losses, and protect public health vary between geographical locations and associated HAB types. However, a critical step in many monitoring processes encompasses the detection of the HAB species directly. Counting individual species can guide either management decisions or additional monitoring activities [2]. The ability to classify phytoplankton individuals extracted from suitable locations becomes vital in the early detection of HAB events.

The key to developing early warning and detection capabilities for toxic algae blooms is a well-designed field-sampling program [22]. The focus should be on the detection of dangerous species. Classifying individuals seeking the excess of known dangerous species is crucial in this case. The traditional analysis method is labour-intensive through extensive manually operated microscopic examination (e.g. Flow cytometry - FC; and Image Flow cytometry - IFC). Conventional microscopy allows high-resolution images; hence detailed inspection of individual phytoplankton can be delivered at the expense

of imaging throughput (i.e. the number of images analysed per time unit). FC enables quantitative analyses of particles' optical properties in a fluid, resulting in particle size and composition information [44]. In contrast, IFC integrates the traditional FC technique with advanced image acquisition, producing an image output enriched by tailored particle-specific information.

2.1.1 FlowCam: manual and visual analysis

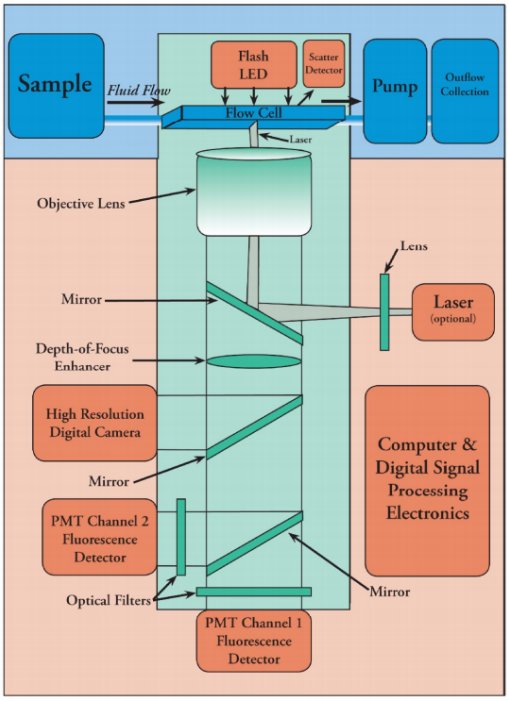

Figure 4: FlowCAM Block Diagram [40]. Computational components - hardware and software) are in red blocks. Other components in represented in blue.

The FlowCAM is a bench-top integrated commercial system for particle analysis in a moving fluid. Despite its high cost, the instrument is flexible and widespread, highly used to analyse phytoplankton populations [7, 36, 35, 59]. It provides the capabilities of fluorescence detection, light scattering detection and image processing used to enumerate particles in a fluid for human-based classification assessment. The user is usually a well-trained biologist with years of experience analysing this image. The instrument can be operated in fluorescence-triggered or auto-trigger modes. In addition, the analysis software features valuable tools for manually characterising each particle image under many measurements, including length, width, aspect ratio, equivalent spherical diameter, etc. More recently, the device manufacturer introduced AI-based software that can help trained biologists with the daunting task of visual-based phytoplankton classification. However, as it is a very early stage of development, the manufacturer does not provide sufficient information on the method or technique used to build this classification capability. In addition, the robustness of this method is yet to be released.

FlowCam can be considered a ”golden standard”for manual and visual classification of phytoplankton populations. However, the instrument is costly and remains so mainly because the offset of R&D investments cannot be neglected. The economy of scale has not reduced the price tag of FlowCam equipment which figures as strictly for work use requiring a reliable power supply. FlowCam tabletop design choice prevents deploying applications that favour data analysis as close to the target water environment.

2.1.2 FlowCytobot: in-situ applications

Figure 5: The FlowCytoBot system and this external protection tube.

Imaging FlowCytobot Olson and Sosik [45] is an IFC device specialising in marine in-situ applications. The system can be deployed in many scenarios intended for marine experiments that have grown in number in recent years [9, 25, 8, 24, 5]. The device uses a combination of video and flow cytometric[2] technology to capture phytoplankton organisms images with particular attention to target chlorophyll in each image. The FlowCytobot is submersible, operated at a maximum depth of 40 meters with a system autonomy of 6 months, transmitting real-time data to a remote facility. The device works as a standard flow cytometer and relies on hydrodynamics that focuses on a sample fluid stream that passes across a laser interrogation point. The laser excitation for the flow sample leads to light scattering and fluorescence emission from chlorophyll in cells. The scattered light triggers a xenon flash lamp to illuminate the flow, and a monochrome CCD camera

[2]Flow Cytometry is the analysis and feature extraction of particles in a passing fluid. Cytometry uses, in most cases, a laser excitation procedure and extracts feature such as morphology and size via light scattering.

captures the image. Similarly to other IFC instruments, the FlowCytobot generates a rich information set that annotates the captured images. Although this device enables in-situ data analysis at a high cost, the system falls behind in providing a more automated and rapid analysis solution for the end users. The advanced intelligence enabled by machine learning at edge devices is an area the manufacturer recognises as promising but still has not yet delivered an integrated solution. The current solution for the image analysis is to connect the FlowCytobot to a workstation computer for manual data download. The generated raw images and annotated information should be analysed by a well-trained biologist, either by error-trial visual inspection or third-party software.

2.1.3 FlowSight: from biomedical to marine applications

Recently, the number of IFC instruments has increased to include the Imagestream X Mark II (Amnis-Merck Inc, Seattle, USA), also known as FlowSight. Unlike the other imaging flow cytometers for aquatic studies, the Imagestream instrument family has been primarily developed for cellular analysis in the biomedical industry. The common usage includes tailored analysis such as cell signalling, cell cycle and mitosis, cell-to-cell interactions, internalisation and co-localisation, cell death and many others. Nevertheless, this device has entered the marine community to support phytoplankton-related research [66, 13]. More relevant work is expected as this device spreads through the marine scientific community. Any attempt to classify phytoplankton using FlowSight has to be made through visual analysis by trained biologists.

2.1.4 PlanktonScope: A call for low-cost, community-based device

Pollina et al. [48] presents a flow-based imaging platform in two distinct configurations built around open-hardware principles, leveraging off-the-shelf hardware components. An open-source software strategy relies on existing interfaces, image processing libraries and a flow-based visual platform, thus enabling users to customise acquisition and data processing steps. The system leverages the open-source hardware Raspberry Pi computer[3] enabling a mechanism to (a) control the electronics, (b) acquire and process the image, (c) and serve as the user/machine interface. The work presents a workflow of image processing executed as a fixed number of images collected and turned into a batch. First, the system calculates the average background from five frames surrounding the current one. Next, this background is used for subtraction. Then a threshold is applied to the image, highlighting the object and allowing it to be extracted. Finally, the method pulls features from the binary extracted objects, including eccentricity, equivalent diameter, Euler number, area, orientation, perimeter and many others. The image processing

[3]Raspberry Pi is a series of small single-board computers developed by a group of Cambridge University academics to promote teaching activities worldwide. Raspberry Pi has gained significant ground in the embedded systems space, and it is now integrated into many systems and products worldwide.

(version v.1) is done with a desktop computer for faster data computing. However, version V.2 does initial processing in the Raspberry Pi. Afterwards, the segmented images can be offloaded to another device for further processing.

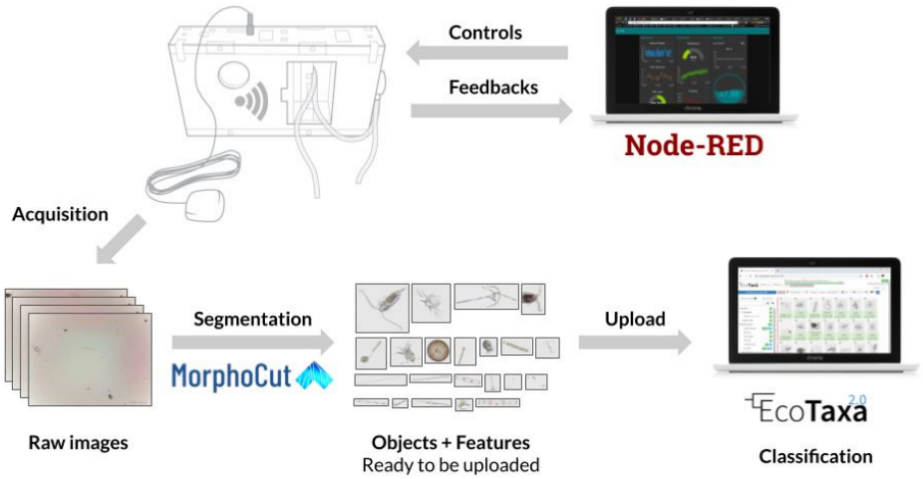

Figure 6: PlanktonScope Software Flow[48].

This can be considered a step towards automated phytoplankton classification and early detection and warning of HABs. The proposal of a low-cost embedded system capable of acquiring and storing data for classification allows the user to take this technology to the water environment location. The system modularity is a positive aspect o the system as it will enable integration using commercially available low-cost hardware components. However, the system still lacks intelligence. The embedded software does not give functionality for any machine learning classification model. Data must be uploaded to an external online tool so that server-based classification software can process the data for user analysis.

The premise of this dissertation work is that running this image analysis step on a locally available system means getting the results as the data collection happens. This system capability brings agility to water quality management, especially for those applications where clock ticks are of utmost importance. For example, aquaculture farming and early detection of toxic algae fall precisely into this application constraint and requirement. Table 2.1.4 provides key facts about the surveyed devices. This is not an exhaustive list, and it should serve only as a discussion guide for the current system limitations in water quality monitoring and, more generally, on any application that can potentially have embedded intelligence using machine learning models.

FlowCAM and FlowSight are market-available bench-top products with limited phytoplankton classification capabilities. Such device manufacturers have not disclosed details of their tools' classification methods. In addition, both devices require a reliable power supply to work.

Table 1: Review summary of devices.

	Off-Grid Power	Embedded Classification	Low Cost
FlowCAM	NO	Varies	NO
FlowCytoBot	YES	YES	NO
FlowSight	NO	NO	NO
PlanktoScope	YES	NO	YES

FlowCytobot and PlanktoScope are both embedded devices intended for field-based missions. In both cases, the instruments take the samples and process them for classification upstream (off the device). FlowCytobot uses an external algorithm that classifies phytoplankton images. Similarly, the PlanktoScope connects to the EcoTaxa cloud-based service and uploads images for classification.

Regarding cost, PlanktonScope is the only one designed for the low-cost end of the spectrum. The other are products used in a wide range of high-end research laboratories worldwide. The cost might be suitable for a highly funded research project, but it is prohibitive for end-users such as farmers in aquaculture.

2.2 AI Edge Devices: Beyond Traditional Devices

Embedded systems have intensified as high-performance and low-power consumption systems. This mid-range device represents much of the increase in the IoT, robotics and extended reality emerging applications in recent years. The Jetson NANO[4] is an entry-level Edge device, a powerful single-board computer in a small, portable form factor providing parallel cores for AI applications. The onboard GPU enables the platform to perform tasks of higher complexity than classic embedded IoT devices[5] As mid-range devices mature as technologies, their computational power can push innovative models and techniques to a new frontier. A fully equipped edge device for AI applications supports complex algorithms to overcome application constraints in areas dominated by non-intelligent devices. Also, it enables applications that already have AI support but depend on cloud connectivity to run independently and more reliably. But, it is important to highlight the challenges of taking machine learning algorithms to the edge. ML models typically run in the cloud, in resource-rich platforms, and communicate with less intelligent devices via a communication network. Bringing those resource-hungry models to the edge requires (a) a thorough understanding of the edge device limitations and (b) reasonable steps to fine-tune models to run optimised versions on resource-constrained devices – power, memory, CPU and communication constraints. To what extent such constraints impose limits to the cloud-based models remains a question worth investigating further.

[4]https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[5]GPU-enabled ARM-based systems are now available from Intel, AMD, ARM and others. This work focuses on one of them with reassurance that the resource limitation model is transferable across these devices.

The literature offers various techniques to optimise, miniaturise and compress ML models [28]. Each method addresses the problem of large and power-hungry models from different viewpoints where interesting results are reported. Using the Jetson NANO opens a viable path towards embedded ML models in edge devices. This NANO system offers an ML model optimisation tool. The TensorRT library is primarily developed for desktop hardware and generates an inference engine customised to the architecture where the target optimisation is intended.

2.2.1 Jetson NANO Plataform

This platform is a low-cost development tool for image classification, object detection, segmentation, speech processing, autonomous driving, industrial robotics and other applications. The Jetson NANO (Figure 7) module has a comprehensive development environment called JetPack, a complete Linux environment with ready-to-go libraries. On the hardware side, the Jetson NANO is equipped with a heterogeneous architecture integrating an ARM-based CPU core with a customised NVIDIA GPU. This integration has some memory optimisation benefits (shared space) that allow the CPU to support embedded applications with the option of acceleration through CUDA-capable GPU cores to speed up complex ML tasks.

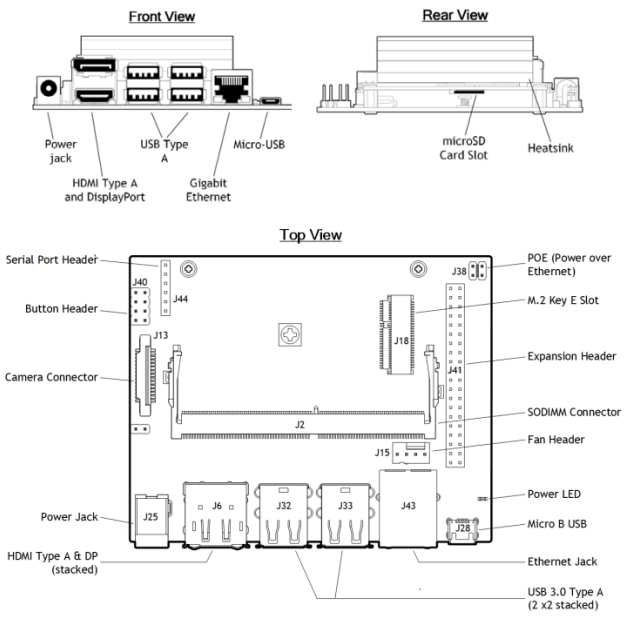

Figure 7: NVIDIA Jetson NANO architecture, highlighting each component in the carrier board. [62]

The Jetson NANO platform is fully equipped to handle machine learning applications. The system is built on a Quad-core ARM Cortex-A57 MPCore Processor and a 128-core NVIDIA Maxwell GPU. It also comes with 4GB (or 2GB) memory that facilitates project tasks and helps applications achieve operational performance and capabilities. Also, the

system has an in-built 16GB storage device and a removable microSD card for further storage and booting functions. The system specification is summarised in Table 2.

Table 2: 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	-

2.2.2 Model Optimisation

Model optimisation is a natural process of achieving an objective at the expense of some other aspects. Model Optimisation techniques attempt to maintain the quality of the AI services (and associated models) at the edge despite the lack of sufficient computing resources, storage, and communication bandwidth. ML models are notoriously memory hungry and require large amounts of power. Therefore, it is necessary to use techniques that facilitate its use on edge. Optimisation techniques in this area can be broadly categorised into model compression and conditional computing. For this work, we will explore a few model compression methods and outcomes that use them. Such optimisations explore complexity reduction in deep neural network architectures as valuable heuristics for deploying resource-hungry ML models into resource-constrained edge devices.

We review pruning, quantisation, and low-rank factorisation as model compression techniques. Also, the TensorRT optimisation tool is presented as an alternative for practical deployment in the NVIDIA underlying hardware.

2.2.2.1 Model Compression: Pruning

Pruning of parameters is the most widely adopted approach to model compression. This technique evaluates the neural network parameters and their contributions to the results. Each neuron contributing to the inference is then pruned from the trained DNN. Pruning can reduce the DNN size at the expense of negative performance impact.

Han et al. [23] is an effort towards model optimisation using pruning as the primary tool. This work reduced the VGG-16 model size by 13 times without incurring accuracy losses. The paper explores how such a reduction allows for on-chip model storage. This reduction can avoid loading the model from off-chip RAM, which is a costly operation in terms of energy. However, the study does not offer experimental data on power consumption differences, where the work primarily focused on desktop systems. There is room

to explore issues associated with a limited resource environment, mainly in applying new edge device optimisations.

Li et al. [38] is a one-shot[6] *structured pruning* framework. Traditional pruning methods operate on the entire deep neural network architecture, subjecting the complete model parameters to pruning. In contrast, independent pruning tasks are performed on filters and channels in structured pruning. This work reports that the model does not suffer any penalty in terms of performance after passing through the framework, thus achieving a similar level of accuracy as the base model. Although the authors comment on the framework's usefulness for low-power systems, the work does not offer details on experimental results.

2.2.2.2 Model Compression: Quantisation

The quantisation is a compression optimisation method that decreases computational complexity with minimal accuracy losses. In particular, it is a technique in which a model's arithmetic representation of the tensors (parameters) is reduced. For instance, quantisation can be applied to various target machine data representations, such as floating point 16-bits, integer 16-bits, integer 32-bits, and so forth. A tensor quantised to a lower form of machine number representation executes all or some of its operations in lower precision. As a result, the data within the model is represented more compactly, leading to insufficient memory and energy requirements. Unlike pruning, quantisation promises power consumption improvements directly.

Yang et al. [69] integrates parameter quantisation with dynamic programming by proposing an algorithm with two DNN quantisation approaches. The author reports a 16x compression factor for a ResNet-18 model. However, such a result has a 3% accuracy drop associated but remains encouraging for developing intelligent systems at the edge. Although the consensus is that quantisation improves the model's energy efficiency footprint, the work does not elaborate on results in this direction.

Huang et al. [27] proposes a mixed precision quantisation scheme used in DNN inferences. The scheme targets weights along model inputs and partial sums inside the hardware accelerator. It also explores an automated quantisation flow powered by deep reinforcement learning to search for the best quantisation setup. The author reports a reduction in model latency of 3.89x and a decrease in power consumption of 4.84x while model accuracy drops by 1.18%.

2.2.2.3 Low Rank Factorization

Low-rank factorisation helps condense the set of DNN parameter weights hence limiting the number of required computations in convolutional layers. The technique is

[6]One-shot learning is a method whereby an ML model is trained to measure the distance between 2 input images. The distance is the learnt function as part of the model training.

centred on calculating another low-rank matrix that approximates the DNN parameter set, convolutional kernels, or both. The low-rank factorisation optimises memory usage on Edge devices but also seeks to reduce latency simultaneously. Unlike pruning, after applying low-rank factorisation, there is no need to retrain the model.

Chen et al. [11] used the low-rank factorisation by applying a singular value decomposition (SVD) transformation. As a result, the work reports a substantial reduction in the number of parameters in the model's convolutional kernels. Such a reduction led to floating-point operations (FLOPs) falling by 65.62% in VGG-16 trained models. Also, the work reports that an increase in accuracy of 0.25% has also been observed.

2.2.2.4 A Practical Tool: TensorRT Optimisation Engine

NVIDIA's TensorRT [7] 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 [64, 61, 54, 19]. In addition, Shafi et al. [56] has extensively tested this platform as an optimisation engine for edge applications. TensorRT acceleration library boosts general AI models for integration into NVIDIA hardware.

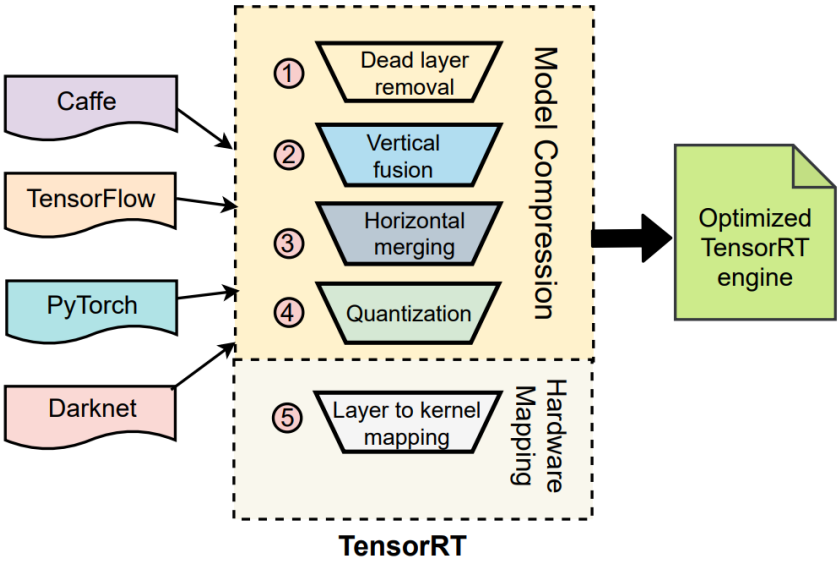

Figure 8: TensorRT optimisation steps [56]

The library runs directly on the available GPU's Compute Unified Device Architecture (CUDA) cores. These are the fundamental blocks with which NVIDIA creates its GPU—enabling native use of the CUDA system and benefiting from GPU computation acceleration. The CUDA cores are an array of parallel processing units within the GPU system that can be interfaced and controlled with C, C++ and Fortran without requiring

[7] https://developer.nvidia.com/tensorrt

assembly knowledge. The use of TensorRT for deep learning optimisation goes through two phases: build and deployment. First, the library optimises the model's network configuration and generates an optimised plan for computing the forward pass through the neural network. The deployment stage takes the form of a long-running service or user application that accepts batches of input data and executes the inference by following the optimised plan.

In the deployment phase, TensorRT does not require the user to run a deep learning framework on the target hardware. Instead, TensorRT identifies opportunities to optimise the network and runs the optimised network, minimising latency and promoting high throughput. The library optimises trained neural networks for run-time performance, delivering faster and less power-hungry models. It executes several crucial transformations and optimisations directly on the AI model's graph (Figure 8):

- It drops any layer with unused outputs, thus avoiding wasting computational resources and time.
- It searches for any 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.
- It also performs horizontal layer fusions to enhance performance by combining network layers that take the source tensor and apply the same operations with similar parameters. The resulting single extensive layer contributes to computational efficiency and avoids rework.
- Finally, the model is quantised with the precision of choice (e.g. 32-bit floating point).

2.3 Related Work

This section discusses other literature work highlighting their contribution and unresolved issues.

Bianco et al. [3] offers 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. The results in this paper suggest that the MobileNetV2 [52] is a reasonable DNN candidate to be deployed at the edge. The paper discusses accuracy results, mainly the Top5 accuracy ranking. This metric measures how efficiently each model uses its parameters, showing the overall efficiency of the results in terms of used resources. However, such work carried out experiments on a general-purpose dataset that does not fully trans-

late into our case study (i.e. early detection of HABs from microscopic images). Additionally, Bianco et al. [3] does not use an optimisation engine to perform experiments on the edge device, leaving this an open gap to be explored. Nevertheless, the set of metrics proposed is useful, and this dissertation expands on it for the evaluation experimental work.

Yi et al. [70] explores the concept of *cold inference* in DNN-based inference and the effects of this phenomenon on ML applications. The work highlights recent trends in developing and deploying embedded applications based on deep neural networks. Such model complexity has steadily increased recently, which is not always good news for engineers attempting to place models into edge devices. The cold inference is a phenomenon observed in ML model usage. It is characterised by the inference process taking significantly longer to execute in the first rounds of inference. Cold inference occurs primarily due the device running the model needs, in most cases, to take critical algorithmic steps before the inference. For instance, the device may need to load necessary packages to perform the inference or cache data. After that, the ”warm inferences”do not suffer such setbacks.

Consequently, it is challenging to deploy complex models or multiple simple ones into device memory and expect that when a DNN inference is needed, the system is readily available (warmed up). The cold inference needs to load, initialise and ”execute”a DNN model in a resource-constrained device for a set of data input (batch of inferences). The speed of cold inference matters critically to the user experience and application quality of experience. The work in this paper establishes a series of heuristics to measure and understand the phenomenon of cold inference in embedded devices. The authors test a series of models from the literature and report the performance gap between cold and warm inferences. The authors note, for example, that the cold inference can be up to 3x slower than its warm counterpart. The findings of this work are fundamental for a healthy and well-informed development of DNN-based systems at the edge. Understanding such effects in deployed models and the overall embedded system is crucial.

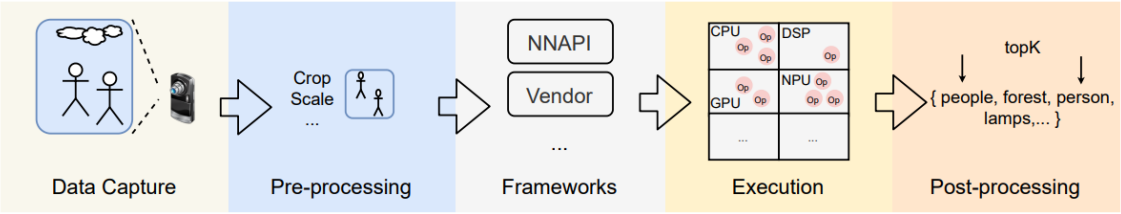

Figure 9: Steps taken in a typical flow: ML image classification task [6].

Buch et al. [6] explores 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 comprises Data Capture, Pre-processing, Frameworks, Execution and Post-Processing, Figure

9. Next, the authors classify the sources of overheads in the ML pipeline 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 and classify it as follows:

- **Algorithms**: Runtime overheads associated with data capturing, pre-processing, preparing data for the model, and running post-processing code that often gives the structured result from the model output.
- **framework**: Overheads related to low-level software in the form of drivers coordinating scheduling and optimisations.
- **Hardware**: CPU, GPU, or accelerator offloading costs, run-to-run variability, and lagging due to concurrent models' execution.

The authors argue that inference-only performance analysis rarely captures the non-model execution overheads. The high-level application pipeline of ML varies very little in many mobile applications, so understanding bottleneck patterns can help mitigate inefficiencies. The goal is to summarise the overheads surrounding the ML model execution and highlight the potential bottleneck steps other than the model execution itself. This time the system has to spend on efforts other than model execution to enable it is the AI Tax properly. The taxonomy proposed by the work is an advance to understand better how such overheads affect the performance of ML applications. This paper sheds some light on trade-offs associated with classification performance (e.g. accuracy, speed) on the edge and resource usage. In addition, it investigates the ability to use low-cost off-the-shelf embedded platforms for technology deployment. However, the work does not sufficiently provide a benchmarking with the TensorRT optimisation engine, widely used in embedded system hardware. And such a tool would be a good fit for the framework the authors propose as the TensorRT serves as a model optimisation interface, running under the inference step.

Shafi et al. [56] is an effort to characterise the impact of using the TensorRT optimisation engine on the inference process of deep learning models deployed on edge. The work describes recent research progress and development of hardware platforms and software stacks focused on the efficiency of embedded NN inference. A hierarchy of hardware and software levels of machine learning-powered inference is presented in Figure 10.

This work explores the impact of TensorRT (Level 6) on the results and performance of DNNs deployed in NVIDIA hardware. The authors experiment with relevant DNN models and examine the interaction between software optimisations and GPU hardware. They also perform empirical analyses based on TensorRT in real embedded GPU platforms using a variety of widely used DNNs. The authors report some promising findings, such as that the TensorRT sustains the DNN's accuracy, even compared to the

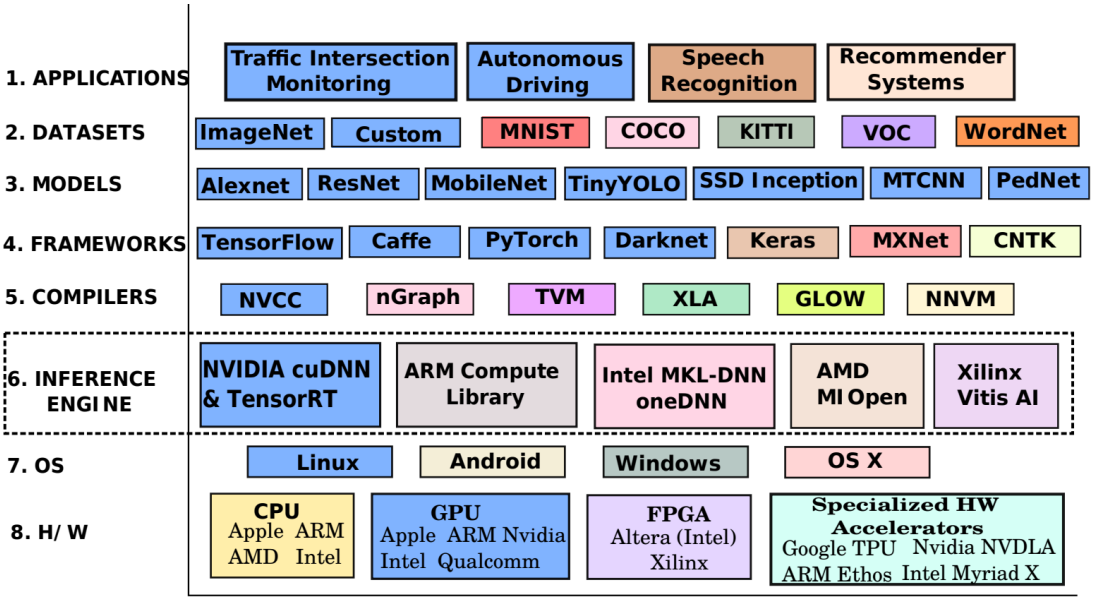

Figure 10: Hardware/Software stack hierarchy for Neural Network inference proposed in Shafi et al. [56]. Layer marked (Inference Engines) will be explored in this dissertation.

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. [56] shows a significant gain in image throughput from model optimisation through TensorRT. It also highlights that some models require additional time to copy the TensorRT-optimised models into the GPU memory. The findings suggest that TensorRT should be considered when proposing an edge system based on DNNs. However, the work does not fully address important metrics for embedded applications, including the impact of TensorRT optimisation on the energy consumption of edge systems. It is difficult to evaluate the practical implications of TensorRT for the models studied because it is not easy to analyse the optimisation gains under the resources used.

Joshi et al. [28] explores the challenges of pushing deep learning methods into an ”All-in Edge paradigm”. First, the paper defines ”Edge Intelligence”as the convergence between AI and Edge computing. To facilitate AI modelling closer to the data generation source, focusing on decentralised and distributed architectures. It is cited how the boom of IoT devices led to a paradigm shift in using Deep Learning. From cloud-only centralised deployment to the distributed model enabled by Edge computing. Among the research questions addressed by the work, we highlight ”What is the standard Key Performance metric to compute for the All-in EDGE paradigm?”. The paper then discusses a series of important metrics for the Edge paradigm.

- **Training Loss:** It captures how well a DNN model fits the training data. Different metrics are selected based on the type of problem, i.e., classification or regression.

- **Convergence Rate:** It defines the number of iterations one algorithm will take to converge on an optimum solution.
- **latency:** Computational Latency estimates the time the DNN model will require to process a query input and infer over it.
- **Communication Cost :** Active monitoring of the communication cost is important to ensure concise data flow and prevent any potential congestion points.
- **privacy:** Privacy-preserving metrics help quantify users' freedom of privacy when an application offers privacy protection.
- **Power consumption:** For some resource-constrained environments, it becomes unfeasible to host models with a larger energy footprint. The energy requirement of a DNN model should be considered for both the training and inference phases.
- **Memory footprint:** Edge Devices usually have limited resources, and it becomes challenging to host a DNN model because of its computational requirements.

Finally, this work presents open challenges within the Edge paradigm and offers future directions deserving of further studies. Latency, Memory footprint and privacy are categorised as the main challenges in the area. The importance of these metrics are of the utmost for evaluating edge-deployed applications. However, the work leaves out the importance of the energy-related aspects of the models. Although listed among the main metrics, the work does not explore in which ways the energy dependence affects the models. Limitation in terms of power supply is a fundamental issue in Edge computing. In this way, the work poses a significant problem that needs addressing. It becomes clear that taking these models to the edge is not a straightforward task, at least if some functional and non-functional constraints need to be preserved. The effects of power usage and the system limits for this are topics that practitioners (e.g. ML data scientists and data engineers) will follow closely.

An analysis Table 2.3 suggests apparent gaps in the literature. Transitioning ML models to the edge certainly has challenges relevant to edge computing research. But a significant challenge is the energy efficiency of the embedded models. In addition, key-related work explores model performance in terms of accuracy without bringing the complete picture of performance against resources used. Also, it is crucial to understand which costs (i.e. also called "taxes") are presented when taking the models into the context of Edge computing. Finally, essential techniques, including quantisation, are necessary to optimise models. The list below summarises some of the open issues identified in the literature:

- *Lack of performance-watt profiling.* The surveyed work analyses performance, accuracy and CPU and GPU usage. An analysis of power consumption as part of the

duo factor performance-watt metric is lacking. This is key information for edge computing. For instance, Bianco et al. [3] explores CPU and GPU usage without considering power consumption. Buch et al. [6] lacks power profiling of the systems.

- *Cold inference.* The work Yi et al. [70] does not show the relationship between cold inference and possible overheads related to available resources.

- *Lack of optimisation techniques profiling.* The impact of model compression techniques goes beyond the accuracy of the inferences. Also, the power consumption impact that arises from such optimisation heuristics needs to be considered. For instance, the optimiser features are not fully explored in the work Shafi et al. [56]

To bring complex deep neural network models to edge devices, the following questions need to be addressed:

- (RQ1) What is the performance and accuracy impact of taking cloud-based models to resource-constrained devices at the network edge? This is the guiding question and represents the issue identified across the related work.

- (RQ2) What is the power footprint in running machine learning classifiers in microscopic-level image bench-top analyser devices? And what is the impact of edge device software libraries on the deployed models in terms of performance-watt? This question addresses the important optimisation techniques and heuristics that can be used, focusing on a case study (e.g. image classification task).

The present work explores these two questions in more detail in the following chapters.

Work	Year	Authors	Research Aim	Open Gap
Bianco et al. [3]	2018		Deep-Learning Benchmark	The work offers a comprehensive view of a series of models in the literature. Although the authors analyse the models in a strict environment, performance factors are considered widely. However, in resource use, the work is limited to reporting the memory use of the models in the experiments. CPU and GPU usage is hidden behind model data, with no experimental support. The work exempts itself from an analysis of the use of the power of the models, even performing embedded experiments. Without a detailed analysis of the energy used to run the experiments, it is challenging to assess the costs related to the performance of each model.
Yi et al. [70]	2022		Cold Inference	The characteristics cold inference are presented. The work does not show the relationship between cold inference and possible overheads related to available resources. The effect of cold inference on performance is clear. However, it is difficult to assess the damage done without an analysis of the models' consumption profile during the phenomenon.
Buch et al. [6]	2021		AI Overheads	In this work, different forms of AI-related overheads are explored. However, the work leaves out overheads related to the software layer for model optimisation. As the focus of the work is on the embedded world, this class of software cannot be ignored. The work explores the concept of AI TAX and studies the overheads of process steps in addition to inference. However, the work does not explore the energy impacts of these steps. Understanding how each step affects overall power consumption is essential to understanding the trade-offs of each model used in an application.
Shafi et al. [56]	2021		TensorRT Characterisation	The work competently analyses the performance of TensorRT on an embedded platform. However, the work does not elaborate further on the analysis of the energy footprint of the tool. The results indicate the clear and visible effects of TensorRT on the models.

3 EVALUATION METHODOLOGY

This chapter introduces the methodology used to carry out the experimental work, including how we built the image classification system in the edge device to undertake the experiments. The chapter also explores the phytoplankton dataset and the associated model trained from such data. The chapter also proposes an experimental design to evaluate and extract relevant information from the models embedded in an edge device. The evaluation metrics are introduced and their importance for the applications in general, and superficially for phytoplankton image classification – a case study of interest.

3.1 System Profiling to the Transition Challenge

Transiting cloud-based ML models to the edge carries a series of challenges. Some steps should be taken to properly run the models in the target device under the TensorRT optimisation regime. Such challenges are framed as follows:

- **Build an embedded system environment.**
  First, we need a stable environment to take the cloud-based models and deploy them to the edge. This embedded environment can run TensorRT and perform experiments to explore the power footprint of the models in base and optimised versions. This is a crucial step in the *Transition Challenge* as it guarantees that the target device (Jetson NANO) can carry out the experiments. Although, NVIDIA claims TensorRT to fully compatible with all NVIDIA hardware, it has been originally developed for desktop hardware. Using the tool on the Jetson NANO device proved a little more complicated. Different from what is indicated, some manual work is required for the tool to be used on the Jetson NANO, such as creating a virtual environment to encapsulate the project[1].

- **Use TensorRT to create derived versions of models for benchmarking.**
  Once we ensure that our platform for experiments is ready, we can use the TensorRT tool to generate the optimised versions of the classification models. This

[1] Some dockers are available, but most of them are not compatible with the Jetson NANO

way, the experiments can be performed with some optimisation variations: base model, TRT-FP32 and TRT-FP16. As TensorRT generate an inference engine for the architecture where the optimisation is carried out, most available examples of TensorRT converters are made with the TensorFlow 1.x. NVIDIA documentation on TensorRT highlights how to create a converter using the most recent TensorFlow. Such a converter is used within the target device. This is due to the optimisation operation leaving behind architecture-specific operations. The end result is the models in the base form, and the optimised versions for the target device.

- **Select Dataset and Classifier Models**
  Our choice of data and the ML models used in experiments are essential to carry out this research work. Models and data are the two cornerstones of any DL classification solution. The target dataset deployed through the integration pipeline proposed in Guterres et al. [21] has been used. The data and model requirements from the ASTRAL project have helped to specify a typical workflow pipeline. A series of models are selected for deployment on the edge. Each model is optimised with TensorRT to generate performance and resource consumption benchmarking results.

- **Define the most relevant metrics.**
  A key part of any research work is understanding what we are looking for. To properly evaluate the project and its experimental results, we define by which rule we will measure it. So we should survey relevant metrics to help the evaluation efforts, ensuring that such metrics will measure how well the *Transition Challenge* is accomplished.

- **Develop a standardised testing regime.**
  To use the evaluation metrics, we should create an experimental regime. Once the models and their versions are all available and running satisfactorily on the device, they should run in a standardised way. While the scripts that launch the experiments run, a parallel logger collects data from the edge device. The logger interfaces directly with the Jetson NANO and provides energy and computing resource usage data. The experiments were constructed to separate the essential parts of the algorithm so that the analysis could be done over the most relevant data. The warm-up inference step, in particular, was added later just after we detected that the first inferences tended to be slower. After this, research on cold inference has been carried out.

- **Data Aggregation.**
  The data generated for each model should be aggregated in a broad format to use the selected metrics. After this raw data is cleaned and formatted, it is ready for

the metrics to be used. In the end, analyses that cross the metrics, focusing on the performance-watt relation, are offered as a contribution to the evaluation aspects.

3.2 Case Study: Phytoplankton Dataset

A substantial part of the ASTRAL project is related to the early detection and warning of HABs. To this end, it is necessary to research AI and image classification to deploy efficient, intelligent systems capable of classifying microscope-level organisms among phytoplankton populations. A cornerstone for efficiently developing such classification systems is the availability of datasets enriched with information on various classes.

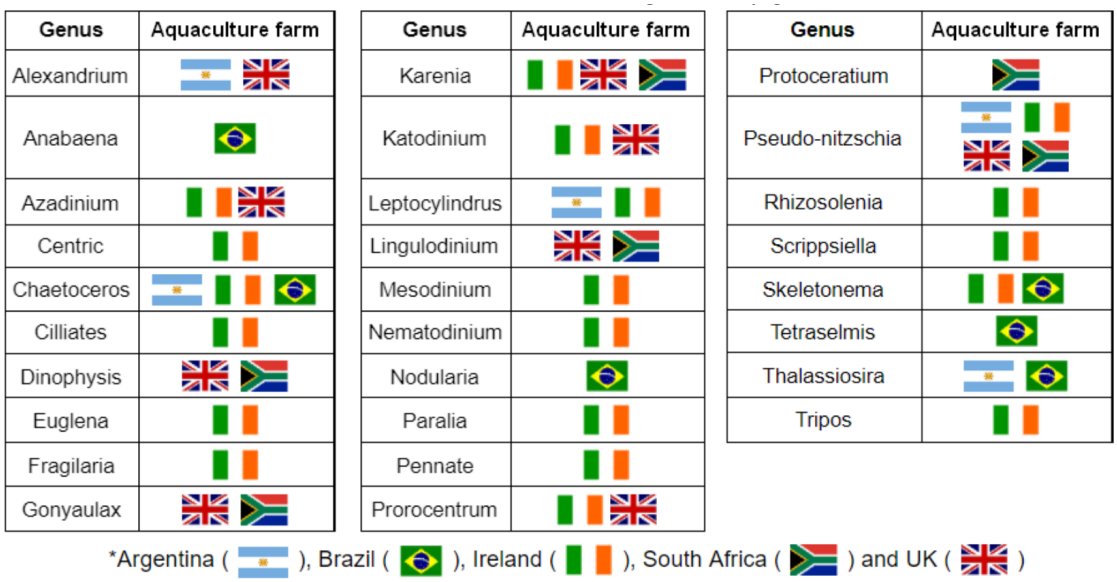

Figure 11: Genus of interest and countries interested in each one. [62]

ASTRAL IMTA labs provided a list of target phytoplankton species and genera for early HAB detection purposes (Fig 11). However, the literature does not satisfactorily present a dataset of phytoplankton images with the scope and richness necessary for our case study. The species of interest to the ASTRAL project appear diffusely spread in several datasets. Theoretically, it can aggregate several datasets to cover multiple species of interest. However, different datasets format their data in the most implicit internal ways. It is imperative to develop standardisation methods that allow using these diffuse data for classification. We deploy an integrated dataset comprising as many species of interest as possible using the integration pipeline proposed in Guterres et al. [21].

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. The image database includes representative grey-scaled images for the harmful phytoplankton classes that create blooms within target aquaculture facilities. Model training has used this dataset to adjust the parameters of Convolution Neural Networks

(CNNs) architectures. Figure 12 depicts microscopic-level phytoplankton image examples used to build the image classifier. The integrated dataset contains around 81,391 images across the target phytoplankton genera. It has been split into *training (80%)* and *testing (20%)* datasets. The training has been a cloud-based server, whereas the testing has supported the edge AI experiments (inference task) carried out in this work. The final dataset contains classes that reflect the following genera of interest: *Alexandrium, Anabaena, Chaetoceros, Dinophysis, Gonyaulax, Lingulodinium, Nodularia, Prorocentrum, Pseudonitzschia, Skeletonema*, and *Thalassiosira*.

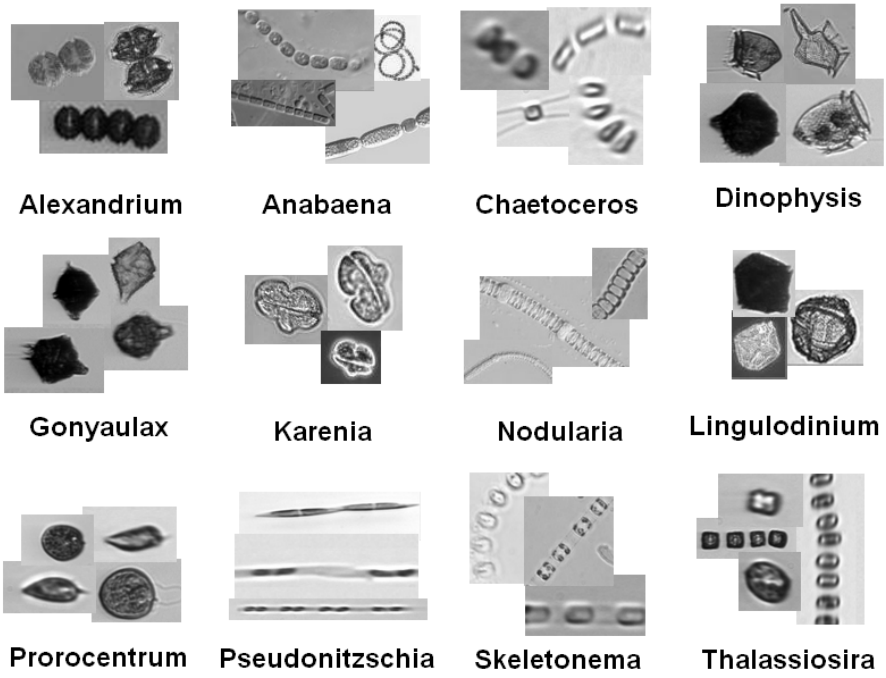

Figure 12: Phytoplankton image examples (50+ times magnification). High intra-class variability, inter-class similarity and imbalanced scenarios bring issues for the practical identification of phytoplankton organisms.

3.3 Classifier Models

The literature has a wide range of models for image classification. Within ASTRAL, the modelling team tested a series of models to select the most suitable option regarding the accuracy and similar metrics. The selected models to be deployed at the edge device are, as follows: MobileNET V2 [26], NASNet-Mobile [74], Inception V3 [63, 62] and VGG [58]. The model training relied on the case study data (phytoplankton datasets) and transfer learning from ImageNet. The training was performed in a cloud machine equipped with an NVIDIA GPU Pascal family (e.g. 3090). This step output is a standard TensorFlow model for cloud-based applications. We also evaluate the potential advantages of model optimisations for taking the standard TensorFlow model (cloud-trained) to an edge device (optimised). To achieve this, the work used the optimisation engine

TensorRT.

3.3.1 MobileNET V2

The MobileNet architecture [26] is a simple but efficient CNN focused on mobile vision applications. This model is widely employed in many real-world applications, including object detection, fine-grained classifications, face attributes, and localisation, especially on embedded devices. It introduces depthwise separable convolutions. A form of factorised convolution that factorise a standard convolution into a depthwise convolution and a 1×1 convolution called a point-wise convolution. The idea is to replace a full convolutional operator using a factorised version that splits it into two separate layers. First, it performs light filtering by applying a single convolutional filter per input channel. The second layer is a 1 × 1 convolution that builds new features through linear combinations of the input channels.

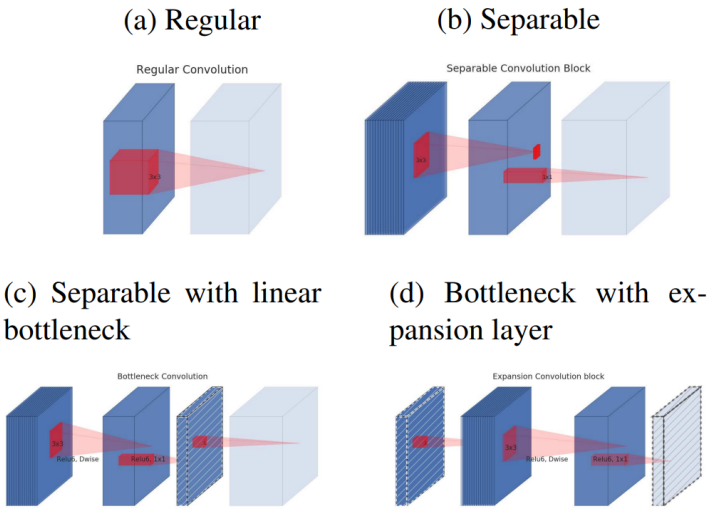

Figure 13: MobileNet: evolution of separable convolution blocks. The diagonally hatched texture indicates layers that do not contain non-linearities [52].

The MobileNetV2 paper [52] then advances this architecture by adding a novel layer, the inverted residual with a linear bottleneck. It takes a low-dimensional compressed representation as an input, which is first expanded to a high dimension and filtered with a lightweight depthwise convolution. Features are subsequently projected back to a low-dimensional representation with a linear convolution.

3.3.2 NASNet Mobile

The Neural Architecture Search Network (NASNet) original paper [74] develops an optimal CNN architecture. NAS stands for Neural Architecture Search and is a technique developed for searching through a space of neural network configurations. The novel idea is to search for the best combination of parameters of the given search space, including

filter sizes, output channels, strides, and number of layers, among others. The reward in this reinforcement scenario is the accuracy of the searched architecture on the given dataset. The general setup of NAS involves three components :

1. Search Space
2. Search Strategy
3. Performance Estimation Strategy

Search Space defines the space of all possible architectures concerning a given problem. For example, one can view neural networks as a function that transforms input variable 'X' into output variable 'Y' through various operations, including convolutions, pooling and activation functions. The search strategy estimates the best architecture for performance and avoids the testing of poor models. There are many search strategies, and the list can include Grid, Random and Gradient-based Searches and others. In performance estimation, the NAS algorithm evaluates the performance of a possible neural network from its design without building and training it. For every architecture in the search space, it can return the estimated performance in accuracy terms.

3.3.3 Inception V3

The Inception V3 [63, 62] is an image recognition CNN architecture that integrates many ideas proposed by multiple researchers over the years. The model comprises symmetric and asymmetric building blocks, such as convolutions, average pooling, max pooling, concatenations, dropouts, and fully connected layers. This model increases depth and width without causing computational strain compared with past approaches.

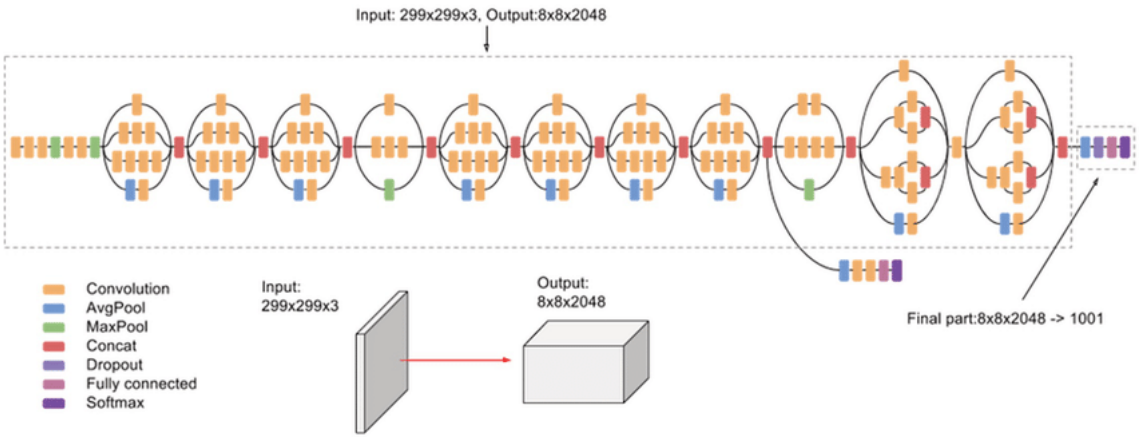

Figure 14: Inception V3 model architecture: schematic view [62]

The whole idea is that, in many cases, several connections between layers of a model are ineffective and have redundant information related to their correlation. To exploit this fact, this architecture introduces an "Inception module", a sparse CNN builds from 22

layers with internal parallel processing. It benefits from auxiliary ”internal”classifiers in the hidden layers to improve its data-oriented and discriminatory capacity. While past CNNs, such as AlexNet and VGG, use either a convolutional or a pooling operation in each layer, the Inception module can use both at each layer. The convolutional filters with different sizes are present in the network layers, thus providing more detailed information on extracting patterns in various sizes. The bottleneck layer, a 1×1 convolutional layer, is used to decrease the computational complexity. The bottleneck layer reduces the number of parameters leading to a network learning deeper representations of features with fewer parameters (compared to past model approaches).

3.3.4 VGG

The VGG network architecture is another proposal for a deep convolutional neural network [58]. VGG is an acronym for the Visual Geometry Group at Oxford University, which proposed the architecture. The VGG model investigates the depth of layers with a tiny convolutional filter size (3×3) to deal with large-scale images. Each VGG block consists of convolutional layers, followed by a max-pooling layer. The same kernel size (3×3) is used in each convolutional layer. After each convolution, a 1-size padding step is used to keep the output size normalised. The VGG16-derived architecture comprises 13 convolutional layers, five max-pooling layers, and three fully connected layers.

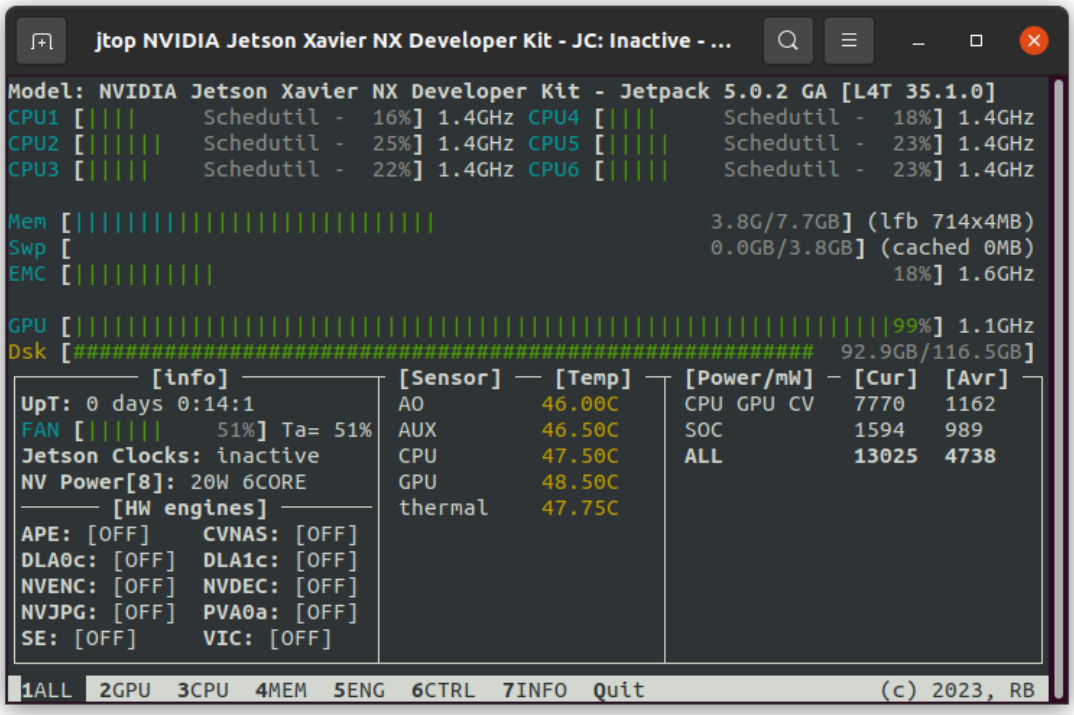

Figure 15: Jetson-Stats interface.

3.4 Logger

To collect data in each of our experiments we build and deploy a imple logger script based in [2]. Gathering data on power, memory consumption, GPU consumption, etc is essential for this work. It consists a fundamental step in its development cycle. The package (Jetson-Stats) selected offers a series of functionalities to monitor and control devices from the Jetson family. Jetson-Stats is a powerful tool to analyse the board in parallel with our main application. It offers a basic interface (Figure 15) for instantaneous monitoring and an API to build custom solutions to better suit our needs. We use the Jetson-Stats' API to deploy a python based script to collect and gather relevant data from the board. Each result explored in this work as collected in this manner.

After collection the data is aggregated, cleaned and normalised in a simple spreadsheet so we are able to extract the graphs and offer analysis. This process was a simple one but laborious as the logger is a external software to the experiments, so the timings of the data samples were meticulously paired with the timestamps of the experiments.

3.5 Experimental Design

This session explores the experimental design aligned with the application use case – the Phytoplankton Classification. The ASTRAL research on phytoplankton classification is divided into phases (Figure 16) covering different application development aspects: (1) In-situ, (2) Data Science, (3) Edge Computing and (4) Analytics Phase. The **In-Situ** phase includes the hardware and software of the phytoplankton sensor tasked with image acquisition to be inputted into the deployed model. **video capturing** represents the majority of hardware development on this work front. Next, the software is represented by **Frame Extraction** and **Speciment Segmentation**, where the collected data is pre-processed before being used in the models. The **Data Science** phase handles the required data and models for the classification application. The **Data integration** step is a process related to the pipeline reported in Guterres et al. [21], deployed in this work to build a target dataset containing the relevant phytoplankton genera. During the **Model Training**, the different models available in the literature are trained using the dataset created previously. The **Cloud Inference** allows models to be evaluated in search of the best parameters and models.

The **Edge** phase is crucial for this dissertation work. The first step – **Model Optimisation** – comprises using the TensorRT library alongside the cloud-trained models to generate the optimised inference engine. This step contains the first tests of TensorRT on the embedded platform and causes the variants of each model to be tested. Next, the **Edge Inference** builds the Jetson Nano environment that supports the optimised models.

[2] github.com/rbonghi/jetson$_stats$

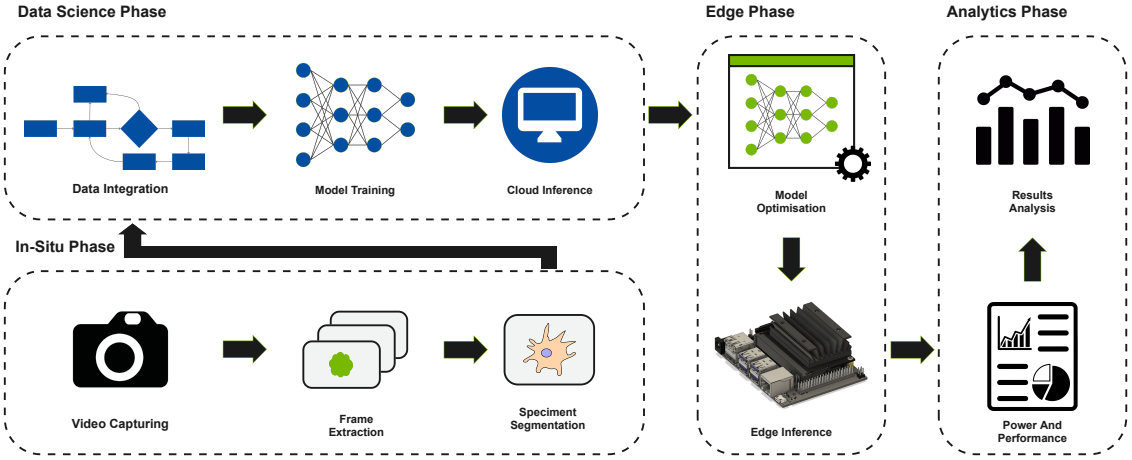

Figure 16: Workflow of ASTRAL phytoplankton classification application. This workflow is typical in many other image classification applications.

This step involves the hard coding to integrate the models into the Jetson NANO, and, subsequently, usage for inferences. The results reported in this work are generated by varying each selected model.

The **Analytics** handles the results generated when deploying to the edge including **Power and Performance** information associated with model deployments. The **Result Analysis** step assesses the output performance and resource consumption metrics customised to the embedded systems.

3.5.1 System Setup

The present work carries out a set of experiments on a Jetson Nano platform to assess the effects of taking a model trained in a cloud-based environment to edge computing platform (representative). In this regard, the virtual environment[3] was built in the Jetson NANO to support the necessary frameworks and libraries (e.g., TensorFlow, CUDAnn, TensorRT). TensorRT engine optimised the models for the target underlying system. An experimental regime runs on the base models and variations. Performance and resource consumption data were collected for analysis.

The TensorRT executes optimisation and compression steps (Figure 8). In the end, a quantisation step provides an optimised model called an inference engine. Each selected model has been optimised in this work using two quantisation options. This quantisation step allows the model to operate only in the desired arithmetic range. The first uses a 32-bit float-point (TRT-FP32), and the other a 16-bit float-point (TRT-FP16). Both are also referred to as *optimised models*. This pipeline is briefly observed in Figure 18. The *base model* is the unmodified TensorFlow model created in the cloud environment, running in the edge device; whereas the *optimised models* are the base model that went through

[3]https://virtualenv.pypa.io/en/latest/

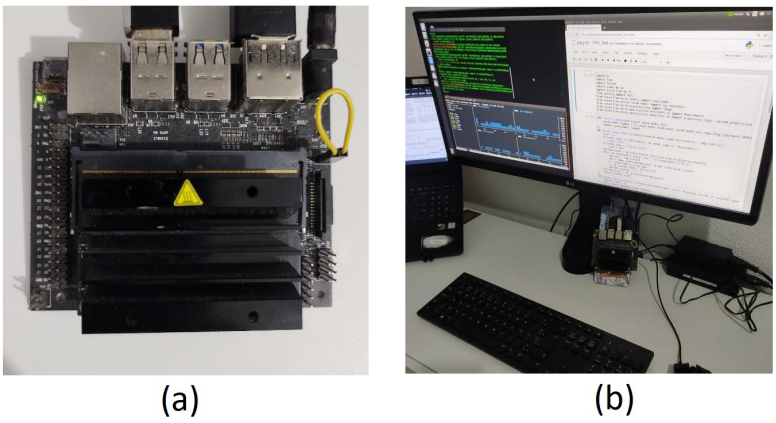

Figure 17: (a) Jetson Nano platform and (b) the experimental system setup.

the TensorRT optimisations. The following AI models and variants are considered and compared with respect to power consumption, resource usage and classification inference accuracy, as follows:

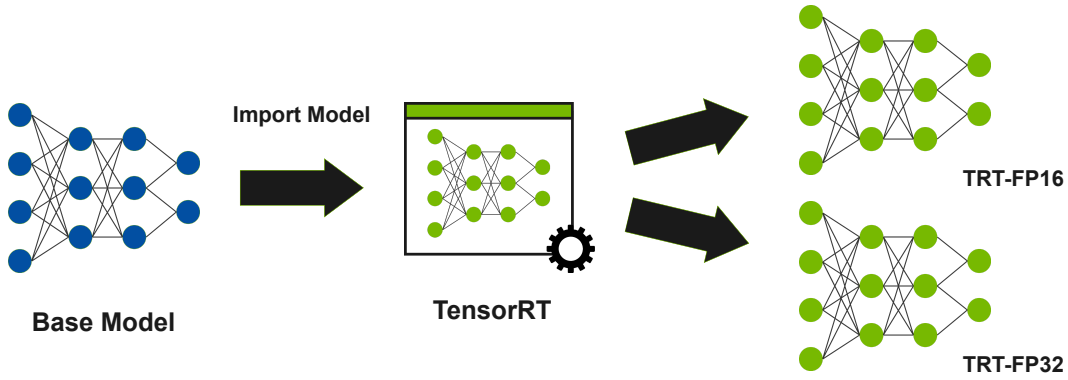

Figure 18: TensorRT: top level view for Optimisation and Quantisation processes. The system imports the cloud-based model trained on the integrated dataset in a high-resource environment. The TensoRT provides an optimised inference engine as the output for the selected quantisation range, namely TRT-FP16 and TRT-FP32.

- **base model:** The original architectures (MobileNetV2, NASNetMobile, Inception and VGG), each trained with the integrated phytoplankton dataset in a cloud-based environment, but running in the Jetson NANO device.

- **TRT-FP16 model:** base model transformed through TensorRT optimisations. TensorRT performs model optimisations, including quantisation of 16-bit floating point.

- **TRT-FP32 model:** base model, also transformed through TensorRT optimisations. A 32-bit floating point quantisation is applied.

Each model and its variants underwent an experimental regime to benchmark performance and system resource usage.

A major issue in our development cycle was the integration of TensorRT with the Jetson NANO board. NVIDIA claims a Plug-and-play relation between their software and hardware, without the need for extensive configuration or customization. However, in this case the integration doesn't work as intended, resulting in challenges and inconvenience. To overcome such challenges, we scour for driver updates, compatibility issues, and to follow Nvidia available instructions carefully. Dependency management took a long time to resolve as conflicts between packages and versions start to show. Resolving these conflicts can be challenging and time-consuming, such conflicts can be summarized as:

- **Conflicting dependencies:** Different packages can have conflicting dependencies, making it difficult to install and configure them together. In our case many TensorRT related packages have deep dependencies needed to work properly. The abundance of needed packages and their complex web dependencies was a tough to configure. In many instances to proper configure a set of dependences break the ones for another package.

- **Version compatibility:** Dependencies in many cases involve outdated versions that may not work with newer versions of other coexisting packages. Updating these packages cause compatibility issues on its own as different packages may require different versions of the same other package. This problem was persistent as many packages automatically install the version of their dependency more suitable for its current version.

- **System conflict:** Both previous problems culminated in the circumstance where many versions specific packages conflict with the ones in the wide scope of Linux. So, the whole process was made within a virtual environment. This solution allowed a packed system containing all the right dependencies and right versions.

3.5.2 Experiments

The resulting AI models are installed into the Jetson NANO platform supported by a virtual environment. The experimental work has been split into steps to evaluate better how different AI model execution stages can influence the performance, energy profile and resource usage on the edge device. The steps have valid roles in running the model for inference purposes. In addition, a monitoring logger software runs in parallel with the model execution to gather measurements for resource usage and energy consumption. The steps are detailed below, sequentially:

- **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.

Each step of the experiment has its own significance. The first two steps comprise the essential base code to import everything needed, extract the data structures from the model and format the data so the model can use them. High-level frameworks usually have such steps performed transparently. However, using TensorRT forces us to build the experiment algorithm without these facilitators, so the steps must be done manually. Then a set of inferences is executed to avoid problems related to cold inference, as reported in Yi et al. [70] and discussed in the previous chapter. This cold inference behavior was a point of tension in the development of the present work. Discrepancies in the results of experiments before and after ”warming up”were identified early in our experiments. As we identify such expressive change in results after an warming up window we focus the research work on the characterisation of this phenomena.

We fine tune the length of the warmup phase to extract most of the already warmed inferences. If the warmup phase were too small the overall time cost of the experiment becomes distorted. In other hand if its to long we also start to waste time away from our inferences of interest. In theory each model has a optimal number of warmup inferences, however 10% of the real inferences was adopt to guarantee standardization of experiments. After we adopt the 10% a long set of experiments were made only to observe if this quantity of warmup inferences were ideal.

3.5.3 Performance and Energy Metrics

We consider multiple performance and resource usage metrics to provide a fair comparison. Precisely, we measure Image Throughput, Accuracy rate, Memory Consumption, Power Consumption, and CPU and GPU usage.

3.5.3.1 Image Throughput

When embedding a model into an edge device, a key system aspect with relative improvement expected is the latency offered. The Image Throughput metric gives an

estimation of the image classification speed at the device (in Frames Per Second - FPS). FPS is a standard performance metric unit, widely used in this classification domain. FPS stands for the number of images per time unit that traverses the model to accomplish the inferences. We report and analyse the FPS metric to better explore the experiment's effects and costs of the inference task. In addition, one should address the overhead in the system resource usage.

3.5.3.2 Accuracy

Accuracy is a widely used and intuitive metric to evaluate model quality, commonly used in image 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. [56] report that TensorRT can achieve similar (and even slightly better) accuracy results to models originally intended for cloud usage. However, comparing cloud-based and optimised model versions from an accuracy viewpoint remains vital to the identification of possible trade-offs between model optimisation and possible performance gains on edge devices. Mathematically, accuracy is calculated using Equation 1.

$$Accuracy = \frac{TruePositive + TrueNegative}{TruePositive + TrueNegative + FalsePositive + FalseNEgative} \quad (1)$$

3.5.3.3 Memory Consumption

We report on the total system memory consumption and a complete memory usage profiling in each experiment undertaken. Memory is very limited in embedded systems, and monitoring this resource closely is crucial.

3.5.3.4 Power Consumption

Power consumption of the base and edge-optimised models is assessed to understand the TensorRT impact on the edge applications. In addition, 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, which is a modern view of performance evaluation reliant on a 2-dimensional metric that fuses performance with power consumption aspects (in this case, possible FPS gains and energy resource usage).

3.5.3.5 CPU Usage

The Jetson Nano is a mid-range edge system equipped with a quad-core ARM processor. However, a powerful one for the embedded devices world is far inferior to the

high-end processors available in a cloud-based environment. So it is essential to observe how much of this hardware resource is spent on the inference.

3.5.3.6 GPU Usage

The critical aspect of image classification is the heavy use of GPU-related operations. The Jetson Nano is again equipped with powerful hardware for embedded standards, such as the 256-core Maxwell GPU. But still, this is an entry-level GPU which cannot be compared to the computing resources available in cloud environments. The task of bringing DL-based classification models to the edge requires investigating how much of this resource is needed to perform edge-based inferences for applications.

4 RESULTS AND DISCUSSION

This section presents the experimental results, including further details on comparing the base model against its optimised versions. This work exploits the image classification task (our case study) through four ML models trained on the dataset of microscopic-level phytoplankton images. All models (base and optimised) were deployed into the Jetson NANO embedded system. A note here is that the base model is the unmodified TensorFlow model created in the cloud environment, running in the edge device, whereas the optimised models are the base model that went through the TensorRT optimisations.

4.1 Accuracy

We evaluated the accuracy using a testing dataset with around 16,000 images, around 25% of the training dataset. The AI models (base and optimised versions) were deployed into the Jetson NANO platform for inferences on the testing dataset. The achieved accuracy for the base model versions are as follows: MobilenetV2 - **97%**, NASNetMobile - **97%**, Inception - **95%** and VGG - **91%** (please refer to Table 4.1 for a full account).

Table 3: Accuracy of each model and its variations.

	CLOUD	BASE	TRT-FP32	TRT-FP16
MobileNetV2	97.36%	97.35%	97.35%	97.35%
NASNetMobile	97.10%	97.09%	97.06%	97.06%
Inception	95.05%	95.05%	-	-
VGG	91.45%	91.44%	-	-

Each model's base version achieves a reasonable classification accuracy for the phytoplankton species of interest. Moreover, the accuracy is compatible with the one obtained during the validation tests in the cloud. This result was expected, as the model parameters calculations remain unchanged despite the model deployment in a low-memory, constrained device. When evaluated on the same testing dataset, both optimised model versions (TRT-FP16 and TRT-FP32) reached a similar accuracy level compared to the counterpart base versions for the MobileNetV2 and NASNetMobile. This result confirms Shafi et al. [56] findings that TensorRT optimised models tend to achieve reasonably comparable ac-

curacy results to their non-optimised model versions.

All four models at the edge behave similarly to their counterparts in the cloud-based environment. The optimised models successfully validated on the edge device (MobileNET and NASNetMobile) present the same levels of accuracy. The MobileNET was superior by a small margin in accuracy. The models edge deployment did not present a notable decrease in accuracy. Our results reproduced what was also remarked by the ASTRAL data science team. Fairly stable accuracy levels when optimising the models indicate that data-to-model over-fitting risks have been mitigated. Overall, MobileNET accuracy performed better than the other models on the phytoplankton dataset. Since the accuracy difference across models was very small, other metrics will play a central role in demonstrating the most suitable model for the classification application at hand. The AI research community (NeurIPS 2019-2022) has pursued other metrics to assess the quality of ML models quality which track the distortion of embeddings into vector spaces. Although this is relevant, it is out of the scope of this work.

4.2 Image Throughput

In this section, we report the inference speed of the analysed models. This metric is crucial for us to measure the applicability of the models and their variations in the real-time environment of the target application. Classifying the pre-processed images at a high frame rate is essential, so there are no bottlenecks in the process. A typical video application frame rate is anything between 15 and 30 FPS. However, such rates are the speed of a standard video, typically for object detection applications, and may vary depending on the application. Therefore, any FPS improvement is welcome. For instance, FlowCAM shoots around 100 images per second from the samples. However, the manufacturer does not disclose in the technical documentation how long it takes to deliver the cropped images and their associated labels.

Figure 19 presents the image throughput (in FPS) across all models and their variations. The plotted data suggests that the TensorRT optimisation engine plays a crucial role in performance across the experiments. First, it yields a substantial Throughput gain for both models successfully executed in the Jetson NANO device (NASNetMobile and MobileNet). Second, the inference throughput doubled on the edge device with TensorRT optimisation. Finally, both optimised options (TRT-FP32 and TRT-FP16) doubled image inference throughput in each case. Such an improvement is expected because the engine puts tremendous effort into model optimisation. These results support the initial feasibility assumption of using TensorRT on edge devices for an image classifier system.

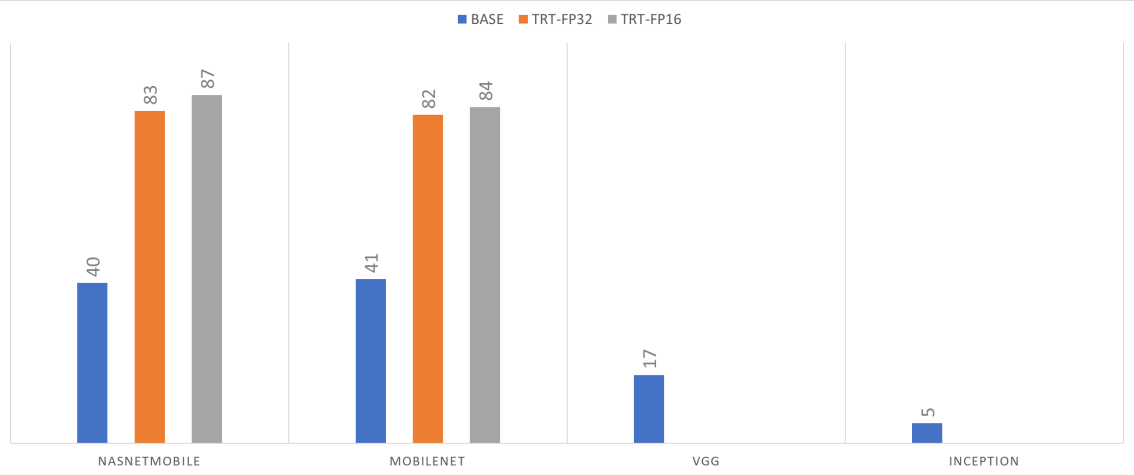

Figure 19: Image throughput results in FPS across all models and versions. Each section is a different model. In sequential order: NASNetMobile, MobileNet, VGG and Inception. Inside the sections, versions are organised as follows: base version, TRT-FP32 and TRT-FP16 versions. As the VGG and the Inception models could not be successfully tested in the Jetson NANO, those sections have empty gaps for the optimised version results.

4.2.1 NASNetMobile

The NASNetMobile model obtained the best results in terms of image throughput. The model running its base version on the Jetson NANO reached 40 frames per second. However, this model doubled its image throughput after TensorRT optimisations, achieving over 80 FPS in both optimisation options. The TRT-FP32 version achieved 83 FPS, a promising figure for edge-based in-situ real-time phytoplankton classification. TRT-FP16 version achieved a slightly better result (87 FPS), considered the best mark among all models and variations. The TensorRT documentation[1] indicates a significant difference between the various options. However, this difference is expected to vary depending on the model used and the target platform for optimisation.

4.2.2 MobileNET

Initially, this model outperformed NASNetMobile by a virtually negligible margin. Then, the model reached 41 FPS, being on the same order of magnitude as NASNetMobile and surpassing it by just one frame per second. Its optimised versions followed the pattern of a gain of around 100% in image inference throughput. The TRT-FP32 version had a slightly lower gain than the previous model, but it doubled the throughput result, achieving 82 FPS. The TRT-FP16 version reached 84 FPS.

4.2.3 VGG and Inception

VGG's base model reached 17 FPS, a result well below the previous models for image throughput. Inception's base model, in turn, achieved a mere 5 FPS, an even poorer result

[1] https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html

Table 4: Coefficient gain of model's variations (successfully completed).

	NASNetMobile	Mobilenet
TRT-FP32	2.075	2.000
TRT-FP16	2.175	2.049

in image inference throughput. Moreover, as reported at the beginning of the section, the VGG and Inception models could not be fully executed for their TensorRT-optimised versions. After the optimisation process for the Jetson NANO hardware,

The optimised models attempted to run many times, constantly failing due to extensive memory usage. We explore in later sections the nature of this error and how the models behaved from the point of memory consumption. The lack of optimised versions of VGG and Inception is detrimental to establishing the best option for classifying phytoplankton at the edge. However, the poor results of the base versions of each model architecture indicate that it would be difficult for these models to surpass the first two choices, as reported in the previous sections. Table 4 shows the gain coefficients of the optimised models relative to their base versions counterparts. In all cases, the coefficient remains around two. We can hypothesise how these models would behave concerning this metric by applying the best-known case (NASNetMobile TRT-FP16) to the VGG and Inception models. VGG could reach around 37 FPS and Inception around 11 FPS. The models cannot overcome MobileNET and NASNetMobile even in their base versions with no optimisation. But, of course, this is just an extrapolation from the available data.

We report the results in terms of image throughput across all tested models. Regarding raw FPS numbers, the TRT-FP16 version of NASNetMobile showed the best results. This version of the model was the closest to the FLOWCAM throughput. Considering that FlowCAM only presents the rate with which it captures the images, not the one it uses to classify. A low-cost system similar to the one pursued by the ASTRAL project could be well served by a classification tool with this type of throughput performance. The MobileNET TRT-FP16 is a closer second place. The difference between MobileNET and NASNetMobile in each version is slight, so the other metrics could enhance the analysis of differences in all models.

4.3 Power Consumption Profile

Power consumption is crucial in designing embedded edge devices, as the system can operate entirely on a battery pack. Power and energy profiling help understand the developed solution's applicability to the target environment, considering resource-constrained devices. An off-grid device is essential to take technology to the far edge of the end user. Analysing how running models change the device's energy consumption profile is important for performance and understanding resource usage trade-offs. The energy profile

of each experiment (model and variation) will be reported. The power consumption is split into the following system steps: **(1)** Loading Model, **(2)** Extracting Infer Engine & Returning Batch, **(3)** Warm Up Rounds and**(4)** Real Rounds.

4.3.1 MobileNetV2 Base Model

This experiment employs the base model MobileNetV2 (i.e. running an unmodified base model with no optimisations). Figure 20 presents the power consumption in sections related to the previously mentioned steps. First, the Loading Model step uses a considerable slice of the total experiment time, keeping energy consumption low and constant over time (around $2900mW$). This first stage contains the library and package loading processes the automation scripts require. Maintaining low consumption, practically at a baseline level of the device, is very much desired. The energy profile does not show any spikes or excessive fluctuations at this stage. The second stage is tiny time-wise, and the energy is at similar levels to the first stage but slightly below (around $2800mW$).

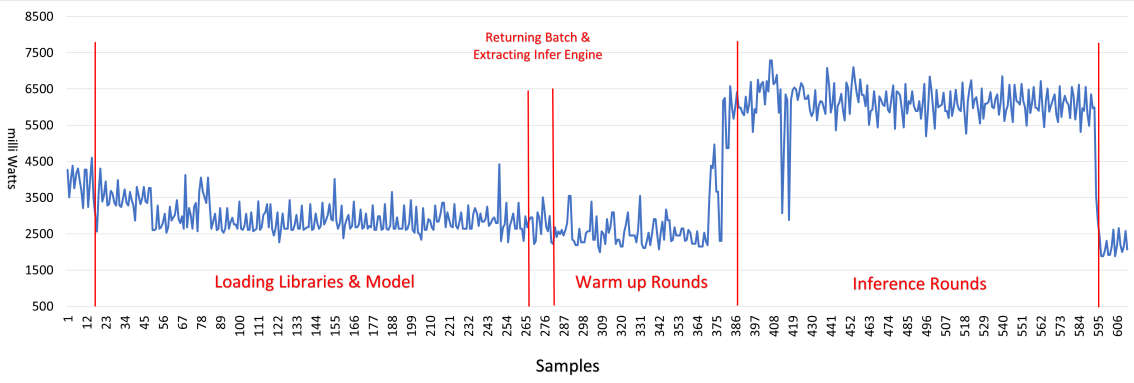

Figure 20: Power consumption: MobileNetV2 on the Jetson NANO platform. The *y axis* is the instantaneous power (in milliWatts); *x axis* is the time (in samples)).

The warm-up rounds step shows relevant results. As reported in the literature, GPU-equipped edge devices can suffer from cold start-related issues when the system setup overhead time dominates [70]. 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 probably left behind. The mean value of the complete step is around $3100mW$. At the end of this step, it is possible to observe a significant increase in energy consumption. These problems cease at this point. It indicates the possibility of the experiment having excessive warm-ups. However, to keep the experiment standardised, all models run the same amount of warm-ups. At the end of the experiment, in the last stage, energy consumption skyrockets. And it remains high throughout the inference process (around $6100mW$). This is expected because, at this stage, the inference process fully uses the Jetson NANO GPU. At the end of the experiment, energy consumption drops sharply to

the probable baseline consumption of the device.

4.3.2 MobileNet TRT-FP32

Next, we report the experiment with the MobileNet optimised with the TRT-FP32 option. Power consumption follows a similar pattern to the base case in the first two algorithm steps (Figure 21). Siting around $2600mW$ for both phases, this decrease can point to the slower import process, leaving the processing aside. However, the first stage lasts much longer (doubled number of samples). It indicates 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. Therefore, it imposes considerable overhead (i.e. "higher tax") on the overall power consumption [6]. A more significant up-time could dilute the system tax.

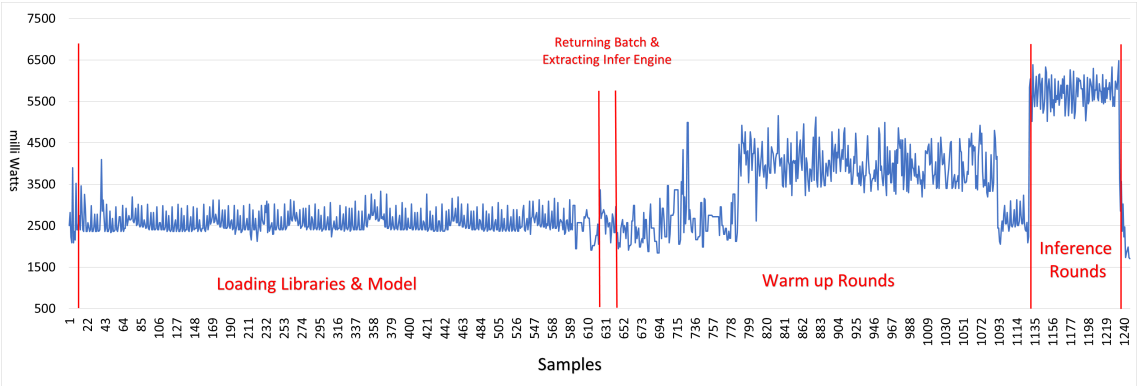

Figure 21: Power consumption: Jetson Nano System running the MobileNetV2 with TensorRT FP32. The *y axis* is the instantaneous power (in milli Watts); *x axis* is the time (in samples).

The TRT-FP32 model causes a more variable power consumption during the third stage with warm-up rounds, creating higher consumption than the beginning and a similar consumption to the base experiment (around $3400mW$). It 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. The addition of TensorRT has an observable effect on the power profile when running MobileNET. It prevails mainly during inference running stages. The warm-up phase is the only one where the average consumption increased from the base version to the TRT-FP32. 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. At this stage, the energy consumption is slightly lower than in the base model. The average energy consumption is $5600mW$ in this step.

4.3.3 MobileNet TRT-FP16

The results of TensorFlow floating-point 16 bits (TRT-FP16) show a profile similar to that of the TRT-FP32 experiment 22. However, the power consumption remains relatively low during the first stage, when the TensorRT library is loaded (around $2700mW$). The second stage does not present any significant change, primarily because of the stage's reduced time (around $2600mW$). The third stage follows a similar pattern to the previous experiment, exhibiting a consumption plateau mainly after the cold start issues are resolved (around $3450mW$). Finally, the last stage shows a high plateau for consumption due to the increased use of GPU resources during inference (around $3450mW$).

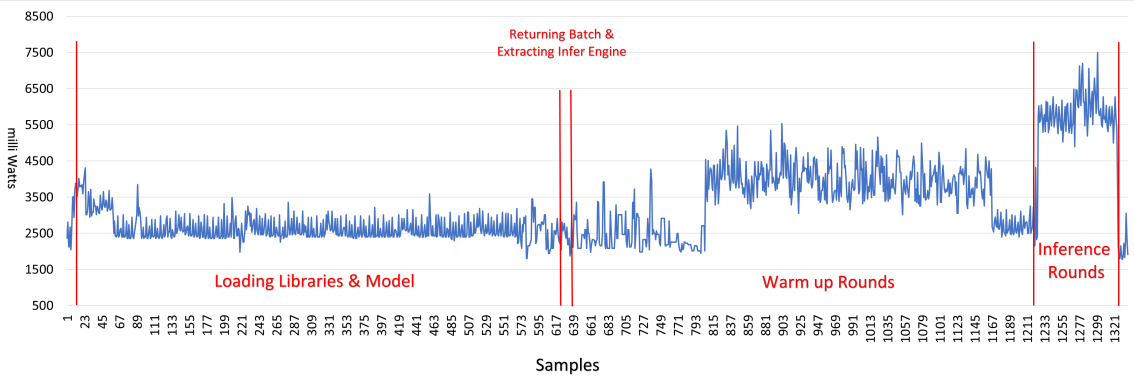

Figure 22: Power consumption: Jetson Nano System running the MobileNetV2 with TensorRT FP16. The *y axis* is the instantaneous power (in milli Watts); *x axis* is the time (in samples).

4.3.4 NASNetMobile Base Model

This experiment employs the base model NASNetMobile. Figure 23 presents the power consumption in sections that are the same as all previous models. This model follows the standard of the base MobileNET.

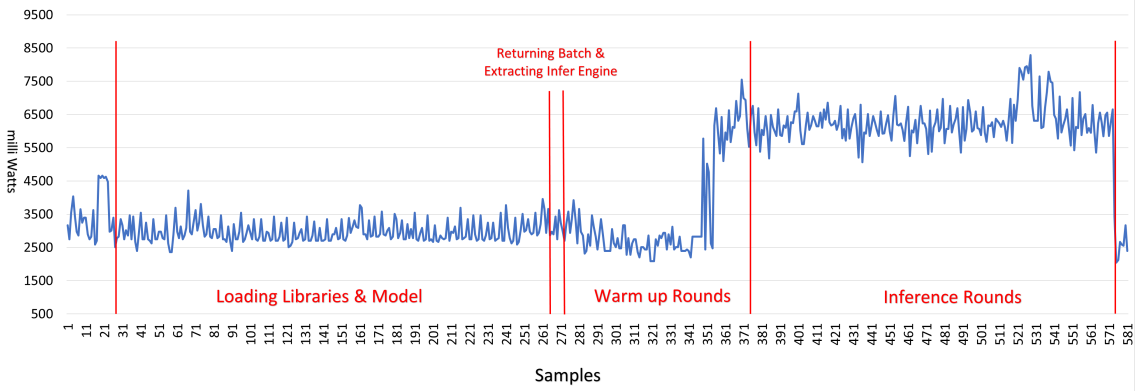

Figure 23: Power consumption: NASNetMobile on the Jetson Nano platform. The *y axis* is the instantaneous power (in milli Watts); *x axis* is the time (in samples)).

The first stage uses an extensive part of the experiment, and no observable spikes exist. Neither expressive fluctuations with a mean power consumption of $3000mW$. Slightly above the NASNetMobile mean power consumption in this step. The second stage remains small for this model and again shows no considerable spikes and fluctuations; its mean energy consumption is around $3050mW$. In the warm-up rounds stage, the NASNetMobile model demonstrates and reinforces the effect of cold inference on power consumption results. At the beginning of the process, the device maintains low consumption, around $3400mW$, at the same level as the base version of the previous model. However, consumption increases even within this stage, strengthening the hypothesis about the impact of cold inference. In the end, there is the consumption of the inferences. The increase of the previous step remains as a plateau of high consumption (around $6300mW$), slightly higher than the base model of MobileNET.

4.3.5 NASNetMobile TRT-FP32

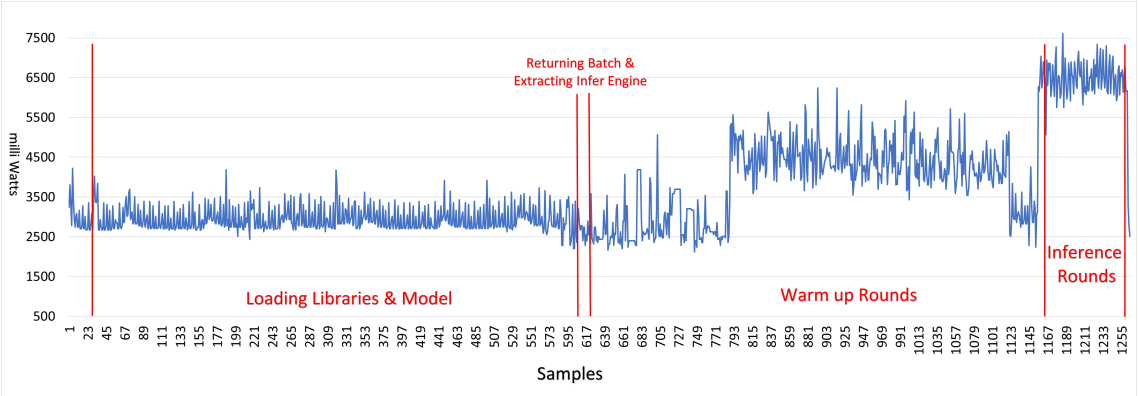

Figure 24: Power consumption: Jetson Nano System running the NASNetMobile with TensorRT FP32. The *y axis* is the instantaneous power (in milli Watts); *x axis* is the time (in samples).

This experiment (Fig 24) represents NASNetMobile optimised with TensorRT in the TRT-FP32 option. The graph demonstrates that the TensorRT optimisation engine has a similar effect on the models where it is applied. The energy profile has an expected consumption pattern. In the first step, a long period is used to load packages and libraries, with a constant and stable consumption of around $2900mW$. Then the second step remains within the pattern already observed, with an average consumption of $2700mW$. Then the model remains within the observed pattern, a more chaotic consumption than the base model (around $3800mW$). But it is compared to the previous model's TRT-FP32 version, highlighting how TensorRT affects models similarly. At the end of the inference step, consumption develops the high consumption plateau pattern expressed in a short time, again supporting the high frame rate results of the optimised models. The average value of consumption in this stage was $6400mW$.

4.3.6 NASNetMobile TRT-FP16

Fig 25 shows the energy profile result of NASNetMobile optimised with the TRT-FP16 option. This model again holds the expected pattern. The first step is the most extensive, steady with no spikes and an average of $2900mW$, entirely within expectations given the other reported models. Then the second step is again tiny compared to the first, with low and constant consumption, with an average of $2700mW$. In the warm-up phase, the model again presents consumption with more pronounced fluctuations compared to the base model. Comparable to other optimised models. The average consumption in this stage is around $4000mW$. Finally, in the inference stage, the high consumption plateau pattern appears again, resulting in an average consumption of $6400mW$.

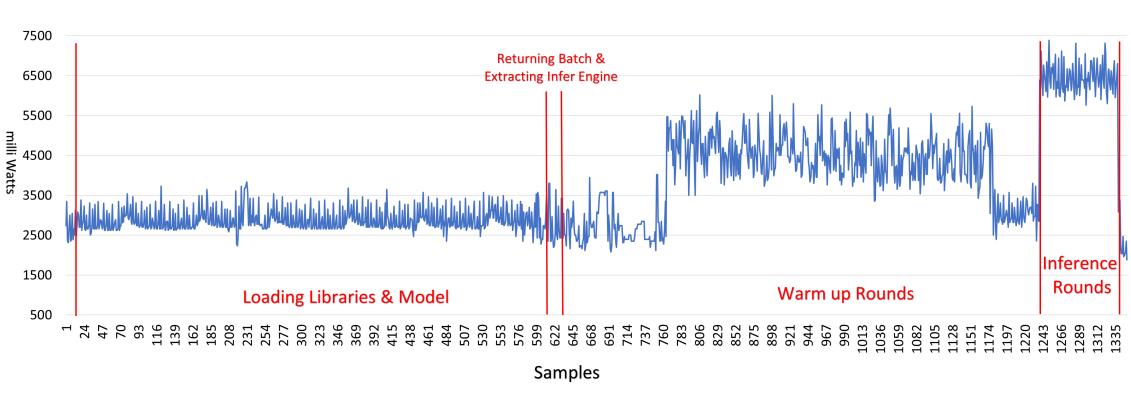

Figure 25: Power consumption: Jetson Nano System running the NASNetMobile with TensorRT FP16. The *y axis* is the instantaneous power (in milli Watts); *x axis* is the time (in samples).

4.3.7 VGG Base Model

This experiment explores the VGG base model (Fig 26), and it profiles the power consumption profile. In this model, the first step comprises the loading of packages which is extremely fast compared to the previous models. In terms of consumption, the stage has an average of well above the other models and variations, sitting around $3600mW$. As in previous models, the second stage is small compared to the first one. The average consumption is $2900mW$, a figure in the same range as the earlier models. In the warm-up rounds stage, the graph shows promising results. As soon as the script enters this step, consumption becomes more volatile, showing more significant fluctuations, with an average of $6200mW$. Approximately halfway through this step, the consumption profile changes abruptly, forming the beginning of the high consumption plateau that remains in the next step. In the last step, the model presents an interesting pattern. The VGG inference process is hugely costly in energy, maintaining an average of $7500mW$ and showing an upward pattern throughout this step. Interestingly, the inference process expressively expands in this model, which is the experiment's most significant piece. The

observed expansion reinforces the image throughput results, where VGG performs far below MobileNET and NASNetMobile.

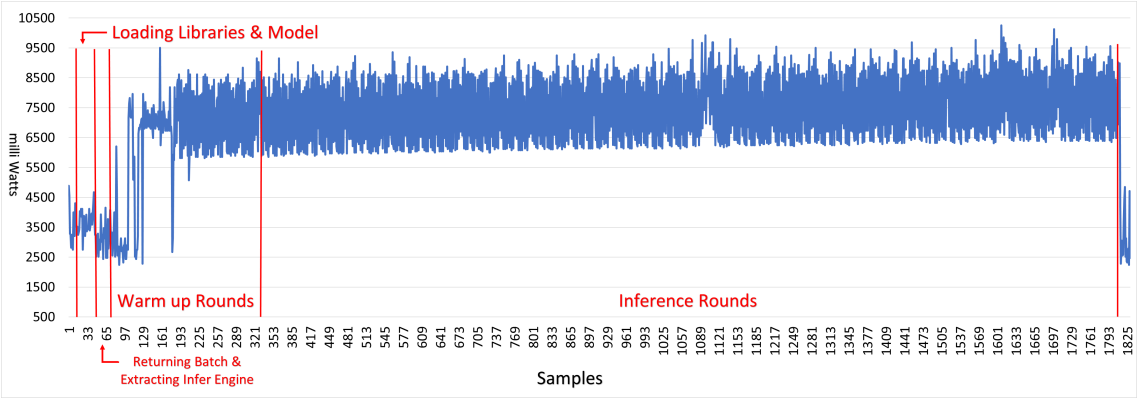

Figure 26: Power consumption: Jetson NANO System running the VGG. The *y axis* is the instantaneous power (in milliWatts); *x axis* is the time (in samples).

4.3.8 Inception Base Model

The final model successfully tested on the device is the base version of Inception. This model returns to the pattern observed throughout this section, except for the VGG. The first step is extensive, with an average of $3000mW$, slightly above most of the studied models. Unlike the others, this model has a more significant fluctuation in consumption at this stage. The second stage remains small compared to the first, with an average of around $3050mW$. Then the third presents more significant fluctuations than the previous ones, which is expected as this pattern was already observed previously. The average at this stage is $3750mW$. In the end, the model presents the plateau pattern of energy cost in the final stage of inference. The model reaches an average of around $6700mW$. It is important to note that the energy profile varies more smoothly at this stage than in any other model (Fig 27).

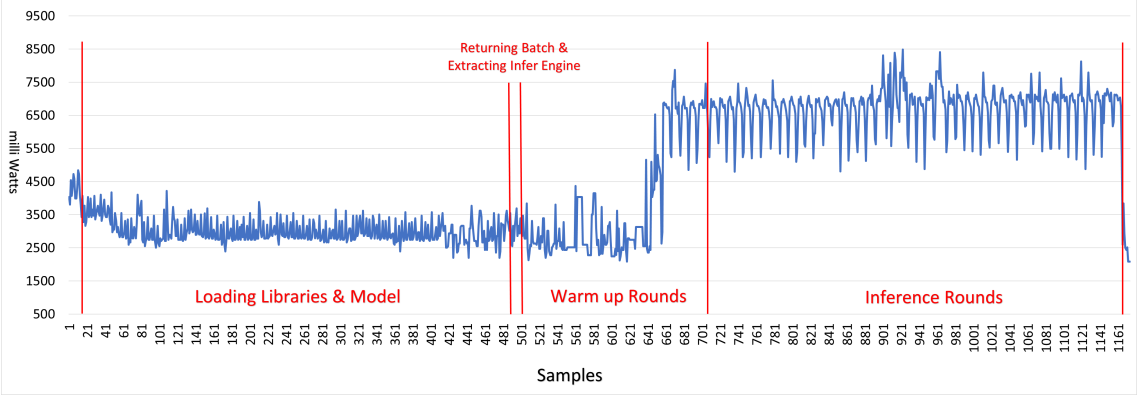

Figure 27: Power consumption: Jetson NANO System running the Inception. The *y axis* is the instantaneous power (in milliWatts); *x axis* is the time (in samples).

We report the energy consumption results of the studied models. The models were systematically tested, and the results are organised to highlight how TensorRT's optimisation affects each step of the script that runs the experiments. Table 4.3.8 presents the grouped data of average consumption of each script stage for each model studied in this session.

Notably, specific models are much more expensive in terms of power. The last column of the table presents promising data for our analysis. It contains the average power consumption values for each model's experiment—first, the MobibleNET model and its variations. The base version already has the lowest average value among all the models in their base versions. Comparing it with TensorRT-optimised versions, one can observe an improvement in average consumption. MobileNET versions TRT-FP32 and TRT-FP16 show a decrease in average consumption compared to the base version. Except for the third stage, the warm-up, the optimised versions consume more than the base version. Comparing the two optimised versions, it is noticeable that they present similar results in all steps. However, the TRT-FP16 version obtained consumption results slightly above the others in three phases, resulting in higher average consumption.

Table 5: Average power consumption in each step across all experiments. The columns report on each algorithm step of the model version as follows: **(1)** Loading Model, **(2)** Extracting Infer Engine & Returning Batch, **(3)** Warm Up Rounds and**(4)** Real Rounds. The last column is the average power consumption in the experiment as a whole.

	1	2	3	4	Avg Pw
MobileNET Base	2900	2800	3100	6100	3386
MobileNET TRT-FP32	2600	2600	3400	5600	3185
MobileNET TRT-FP16	2700	2600	3450	5800	3229
NASNetMobile Base	3000	3050	3400	6300	4236
NASNetMobile TRT-FP32	2900	2700	3800	6400	3624
NASNetMobile TRT-FP16	2900	2700	4000	6400	3641
VGG Base	3600	2900	6200	7500	7216
Inception Base	3000	3050	3750	6700	4633

NASNet Mobile, in its base version, has similar results to the previous model. But with a difference in consumption that cannot be overlooked. In this sense, it is visible that this model has a higher energy cost than the previous one. There was greater consumption in each stage of the script, with variation in the difference resulting in a higher average consumption than expected. TensorRT's effect on NASNetMobile power consumption results is similar to the previous model. However, specific differences must be highlighted in the first two steps where the effect is the same: a similar drop in consumption at an equal rate. The difference appears in the later steps, while the first model slightly increased in the third step. NASNetMobile shows a more expressive increase, mainly in the TRT-FP16 version. In the final stage, there is an increase in power consumption, even if it is small. The table indicates that VGG is a highly costly model compared to the others.

The model has the highest average consumption in three of the four stages of the script. Additionally, this consumption is higher than the other ones by a considerable margin. In the warm-up stage, it is possible to see that the increase is extreme, being almost twice as high in average consumption compared to other models. This increase is maintained in the inference stage. VGG has the highest average consumption in the inference, far above the others and with a clear upward trend. This is worrying because this average may be even higher with more images to be classified. Inception's last model presents results within the pattern observed among the other models. In the first and second stages, consumption is at the top of the scale, excluding VGG, whose numbers are very different from the others. The same goes for the average consumption of the last step. The average power consumption results, aligned with the graphical analysis of the consumption profile, exposed the energy characteristics of each model. These results point to a series of interesting findings. First, optimisation by TensorRT has a similar effect on the models where it is applied. The scale of change can vary, but in the studied models, it is clear that TensorRT generates a decrease in energy consumption, whether in the TRT-FP32 or TRT-FP16 versions. And when compared, the optimisation options indicate that the difference between them is negligible for this metric. Second, MobileNET turned out to be the cheapest model in terms of energy. The base model does not present the highest average value in any step, and its average consumption in the experiment is among the lowest, surpassing even optimised versions of NASNetMobile. The optimised versions outperform the base model by a small but considerable margin. The average consumption of the MobileNET version TRT-FP32 was the best in the experiments, followed by the TRT-FP16 version of the same model. This outcome consolidates MobileNET as the best model in terms of energy consumption. The VGG is the worst among all in terms of power consumption which gives some preliminary indication that the VGG model is not a fit for embedded and off-grid applications.

4.4 Memory Consumption

This section reports on the memory usage during the experiments, running models on the Jetson Nano device. The logger (running in parallel) aggregated the RAM usage to make the comparisons between the models (MobileNET, NASNetMobile, VGG and Inception) and their versions (Base, TRT-FP32 and TRT-FP16). Memory consumption is one of the most critical metrics for defining the feasibility of the models studied for edge deployment. Because the memory available on the device is a formidable barrier and possible bottleneck, as each model may require different levels of memory.

4.4.1 MobileNET

Fig 29 illustrates the memory consumption profile of the NASNetMobile and its optimised variations. The graphs of the three variations repeat the pattern found in the previous model versions, constantly growing to the peak in the warm-up stage. The same steep notch found in early models is seen in the NASNetMobile profile. After that, the model has the same steady growth seen previously until the peak in the inference stage. It is interesting to see how the addition of TensorRT in the two cases already presented does not generate any additional artefacts in the consumption graph line. A sharp second step was expected from the TensorRT loading.

Figure 28: Memory consumption: MobileNetV2 and variations on the Jetson NANO platform. First, at the top is the base version, followed by TRT-FP32 and TRT-FP16. The *y axis* of each graph is memory usage (in MegaBytes); *x axis* is the time (in samples)).

4.4.2 NASNetMobile

We report on the memory consumption of the NASNetMobile and its optimised variations. The memory consumption profile can be seen in Fig 29. The curves of the three variations repeat the pattern found in the versions of the previous model, constantly growing to the peak in the warm-up stage. The same steep notch found in early models is seen in the NASNetMobile profile. Afterwards, the model has the same steady growth seen previously until the peak in the inference stage. It is interesting to see how the addition of TensorRT in the two cases already presented does not generate any additional artefacts in the consumption curve. A sharp second step was expected from the TensorRT

loading.

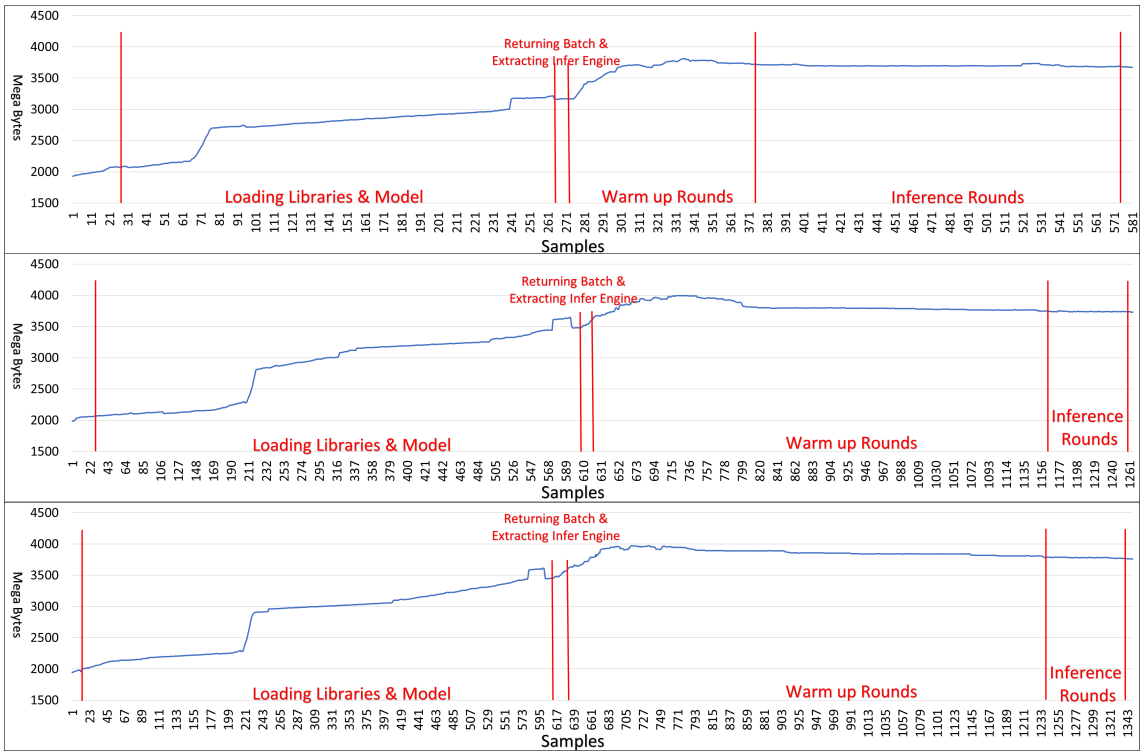

Figure 29: Memory consumption: NASNetMobile and variations on the Jetson NANO platform. First, at the top is the base version, followed by TRT-FP32 and TRT-FP16 versions. The *y axis* of each graph is memory usage (in MegaBytes); *x axis* is the time (in samples)).

4.4.3 VGG

Figure 30 presents the energy profile result of the experiment running VGG on the device. This pattern is reasonably different from the ones seen earlier. At the end of the second stage, this model consumed memory at a level similar to the previous ones. However, the consumption growth was much sharper, something expected since the energy profile already indicated that loading the model is much faster for the VGG case. Again, the memory consumption peak occurs in the warm-up stage. In the inference phase, consumption stabilises until the end of the experiment.

4.4.4 Inception

We finally report on the memory consumption profile of the last model (Inception). The profiling curve can be seen in Fig 31, and the MobileNET and NASNetMobile default reappear, the same as in the power profile analysis. Following this pattern, the profile presents the accentuated step of loading the model in the first stage. Afterwards, it grows steadily to a peak in the warm-up phase.

There is a slight variation in the memory consumption profile for the studied models.

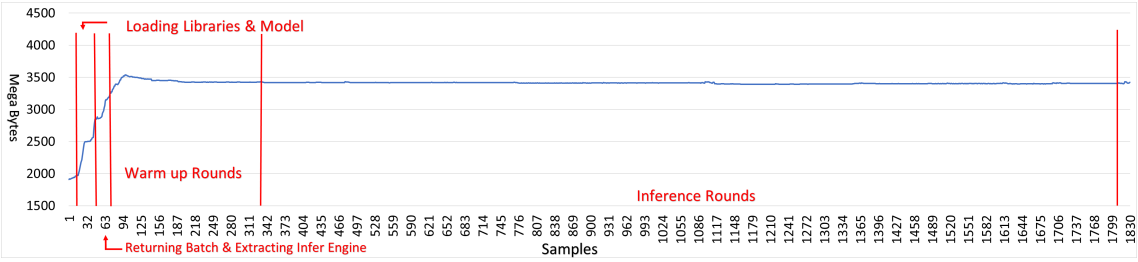

Figure 30: Memory consumption: VGG on the Jetson NANO platform. The *y axis* is memory used (in MegaBytes); *x axis* is the time (in samples)).

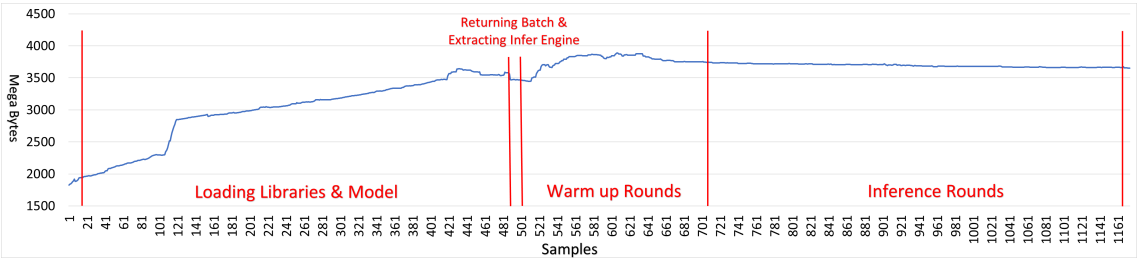

Figure 31: Memory consumption: Inception on the Jetson NANO platform. The *y axis* is memory spend (in MegaBytes); *x axis* is the time (in samples)).

Only VGG stands out with the sharp increase in the beginning. The results obtained in such profiles reinforce evidence found in the energy profiles. However, to better analyse the consumption of the models, we report the peak memory consumption of each model studied (Table. 6).

Table 6: Results of peak Memory usage across all models.

	Peak Memory Usage
MobileNET Base	93.10
MobileNET TRT-FP32	99.61
MobileNET TRT-FP16	99.99
NASNetMobile Base	95.25
NASNetMobile TRT-FP32	99.95
NASNetMobile TRT-FP16	99.28
VGG Base	88.53
Inception Base	97.27

Exposing and analysing peak consumption helps to understand the model's feasibility and how close the model is to the available ceiling. Initially, it is already possible, as observing the table, that all models approach the available memory limit. However, this leaves little space for other applications to execute.

Starting with the base versions of each model, MobileNET reaches a peak memory usage of 93.10%. NASNetMobile consumed slightly higher at (95.25%), reaching a point close to the available memory ceiling. The VGG differs again from the other models with the lowest memory usage among the studied models by reaching 88.53%. Such

results suggest that VGG, even if it is the slowest, is a good candidate where memory consumption is the primary concern in the target system. In contrast, the Inception has the highest consumption among the base models. It consumed 97.27% of the device memory, the closest to the total memory available. Therefore, inception should be used only if the target system has sufficient extra memory.

The optimised versions of the models reached the ceiling of available memory on the device. The four models (MobileNET and NASNet in both optimisation options) consumed over 99% of system memory. It remained at this consumption level until the end of the inference stage. Such an increase is expected because TensorRT needs its entire software package to be loaded into memory. A feature likely to have been inherited from the NVIDIA desktop hardware environment was later adapted for edge devices. This effect of the optimisation engine is worrying, as the extremely constant memory usage can affect the rest of the system's performance and energy footprint. Nevertheless, the optimisation engine approach (as used by TensorRG) is *the facto* industry standard for edge computing applications.

Memory consumption has been the major identified barrier that prevented the VGG and Inception model optimised versions from executing the validation on the device platform successfully. In both cases, we repeated the experiment many times but could not execute such optimised models on the Jetson NANO. In all attempts to run the experiment with the TRT-FP32 and TRT-FP16 model versions, the system did not have sufficient memory to instantiate the tensor objects required for the inferences. This was a showstopper for analysing these two versions under the other metrics studied. Also, this suggests that users have to consider memory stress as a priority in their edge application deployments.

4.5 CPU and GPU usage

Evaluating computational resources (CPU-GPU usage) is vital to understanding model stress on the chosen deployment hardware. Table 7 reports the results of computational resource usage. Most models do not put significant pressure on processing cores (e.g. MobileNET and its variations). However, it can be observed that the optimised versions slightly decrease GPU usage but exercise a more significant load on the four CPU cores. This is because TensorRT optimisations allow some model operations to be offset to the CPU.

The workload that NASNetMobile imposes on the processing cores remains small. However, such a load is larger than the previous model by a small margin. Again, it is possible to extrapolate from the table that the optimised versions offset the computational burden from the GPU by transferring it to the CPU cores. Such GPU-CPU offset is more evident in this case, and the GPU usage drop is more noticeable. In contrast, the VGG again differs from the other models. The CPU cores usage is visibly lower and the lowest

Table 7: Average CPU and GPU usage. Columns are CPU cores and device GPU.

	CPU 1	CPU 2	CPU 3	CPU 4	GPU
MobileNET Base	24	20	32	21.5	25
MobileNET TRT-FP32	27	34	28	35	24
MobileNET TRT-FP16	34	28	36	31.5	24
NASNetMobile Base	27.5	28.7	36.6	23.5	37
NASNetMobile TRT-FP32	32.2	33	30	28.5	23
NASNetMobile TRT-FP16	30	30	34	30.5	28
VGG Base	20.5	19.9	18.5	18.4	91
Inception Base	30	22	27	22	42

for any model. However, this model consumes significant GPU resources. The high VGG-associated overheads have also been observed for the power consumption footprint.

Finally, Inception's last model presents a CPU usage close to that of the other models. However, the model utilises the specialised image processor (GPU resource) at a higher rate than MobileNET and NASNetMobile.

The models have not imposed too much pressure on the Jetson NANO system because the embedded device processors could execute the system processes that carry out the experiments. However, an interesting observation is that the VGG model is costly in terms of computational resource usage.

4.6 Performance and Costs Analyses

It is essential to establish which models are superior in terms of performance, guiding the choice of model for the application. But in the resource-scarce world of edge computing, it is even more critical to establish the costs and taxes related to such performance. A model whose performance is far superior to all others is not so useful if unable to execute correctly on the chosen platform. And especially in off-grid embedded systems, as the one envisaged in the use case scenario, ensuring that every *watt unit*is well used with minimal overhead becomes an important system design choice. It becomes clear that it is necessary to contrast performance results with resource usage data. We use the image throughput metric as the main performance indicator to achieve this. The ability to classify the most significant possible number of images in a time window is central to the applications. We analyse the cost of such performance through average power consumption values over each experiment. This is an indication of the power cost for the gains in the frame rate performance.

4.6.1 MobileNetV2

We initially reported the results on the performance/cost for the MobileNET model and its variants (Table 8). To this end, we organise the results as follows: (a) Total Avg Power is the average power consumption across the entire experiment, (b) Inference Avg

Power is the average power in the inference step only, (c) The Total Performance-Watt (in FPS/W) represents the energy cost of the achieved performance for the entire experiment, (d) and the Inference Performance-watt (FPS/W), the performance-watt value covering only the power consumption for the inference.

Table 8: Power Consuption Results of the **MobileNetV2** model and variations.

	Base Model	TRT-FP32	TRT-FP16
Total Avg Power (mW)	3386	3185	3229
Inference Avg Power (mW)	6125	5711	5860
Total Performance-watt (FPS/W)	12.10	25.74	26.01
Inference Performance-watt (FPS/W)	6.69	14.35	14.33

The base version's average power consumption is $3386mW$ and $6125mW$ for the entire experiment and inference phase, respectively. Such a promising result led to an acceptable performance-watt 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. The TRT-FP32 version reached $3185mW$ across the experiment and $5711mW$ in the inference step, and the optimised model outperforms the base one by a small margin. The enhanced performance-watt (FPS/W) has a superior outcome compared to the base model. The TRT-FP32 experiment achieved $25.74FPS/W$ and $14.35FPS/W$ in the inference step. The optimised model at least doubled the final performance-watt compared to the base model. This result is highly favourable and is derived from the joint result between the slight gain observed in the energy profile and the optimisation engine doubling the model's frame rate.

The TRT-FP16 model outperformed the base model by only a small margin under the power consumption analysis. It reaches an average consumption of $3229mW$ and $5860mW$ during the inference stage. This model's performance surpassed the base model. However, it lagged behind the TRT-FP32 model in the final metric. The performance-watt reached $26.01FPS/w$ for the entire experiment and $14.33FPS/w$ in the inference step, a negligible difference between the optimised model experiments. Such results may indicate an improved performance for each watt spent for MobileNET-optimised models.

4.6.2 NASNetMobile

In the NASNetMobile base model, average power consumptions were $4236mW$ and $6282mW$, respectively, for the entire experiment and inference step. The model achieved $9.44FPS/w$ (entire experiment) and $6.36FPS/w$ (inference only). The NASNetMobile is slightly less efficient than the other models when considering the performance-watt metric. However, the results of this model were still within acceptable limits, considering the return per watt.

The TRT-FP32 version reached $3624mW$ across the experiment and $6470mW$ in the

Table 9: Power Consumption forthe **NASNetMobile** model and variants.

	Base Model	TRT-FP32	TRT-FP16
Total Avg Power (mW)	4236	3624	3641
Inference Avg Power (mW)	6282	6470	6409
Total Performance-watt (FPS/W)	9.44	22.90	23.89
Inference Performance-watt (FPS/W)	6.36	12.82	13.57

inference step. The analysis of power consumption is tricky for this model. The optimised model outperformed the base model in the Total Avg Power by a safe margin. However, it falls short if we consider only the inference step. In this metric, the TRT-FP32 was inferior to the base model. The joint analysis performance-watt is key to understanding the behaviour of this model. For instance, it reached $22.90FPS/w$ (entire experiment) and $12.82FPS/w$ (inference step); thus, the performance-watt result in the whole experiment doubled, resulting in high gains in frame rate. Therefore, despite the inferior outcome in Inference Avg Power, the TRT-FP32 version outperformed the base model in the final metric due to the already mentioned gains in frame rate. The TRT-FP16 version outperforms the base model in the first power-only metric ($3641FPS/w$). However, it falls short of the other optimised version by a tiny margin. Inference Avg Power surpasses the previously optimised model but does not beat the base model ($6409FPS/w$). Finally, as the performance-watt metric, it exceeds both other versions of the NASNetMobile in the two metrics by reaching $23.89FPS/w$ across the whole experiment and $13.57FPS/w$ during the inference step. The NASNetMobile results reinforce the evidence that using TensorRT allows developers to use less energy to power up more efficient models.

4.6.3 VGG and Inception

The first model obtained an average consumption of $7216mW$ throughout the experiment and $7500mW$ considering only the inference. Regarding the performance-watt metrics, the VGG results are well below the others, reflecting the low FPS that the model achieved. It reaches $2.35FPS/w$ in the whole experiment and $2.26FPS/w$ considering only the last step. Such results reinforce the VGG as a model with high overhead. It requires much power to deliver questionable performance, easily surpassed by the other validated models.

Table 10: Power Consumption: **VGG** and **Inception**.

	VGG	Inception
Total Avg Power (mW)	7216	4633
Inference Avg Power (mW)	7500	6700
Total Performance-watt (FPS/W)	2.35	1.07
Inference Performance-watt (FPS/W)	2.26	0.74

The latter model has better results on the power-only metrics by reaching only

$4633mW$ and $6700mW$, the entire experiment and inference step, respectively. However, it demonstrates the worst results in performance-watt metrics among all models. Inception reaches only $1.07FPS/w$ considering the experiment from beginning to end, already a result far below the required for applications. And for the final metric, its results sit at $0.74FPS/w$, less than a unit, which indicates that the Inception, even not having a high energy overhead, does not deliver good performance results for each watt used. This result may be explained by the model not so good performance (extremely slow in FPS), as reported in the image throughput analysis.

It is important to emphasise that the deployment of the models at the edge kept accuracy at similar levels to their counterparts in the cloud. Furthermore, after optimisation, such accuracy levels have not changed.

TensorRT's ability to double the frame rate in the models where it is applied is remarkable. In their optimised versions, the two successfully tested models presented much higher throughput than the base versions. The data collected from the experiments and graphical analysis suggest that TensorRT can make models more energy efficient using the transformations, thus reducing the energy overhead of the model. This overall drop in power consumption could be related to the optimisation engine's effect on the device's CPU and GPU usage. GPUs well know for being power-hungry devices, so shifting some of the workloads away from the GPU is always an energy-saving opportunity.

Memory consumption is the main adverse effect observed in the TensorRT system. Optimised model versions required more memory to execute. The shortcoming is that the TensorRT involves a load of its entire stack into the system memory to run the inferences, a significant limitation for the applications developed for edge deployment. The results of the performance-watt analysis gave a better behaviour view for the models running on the edge device. Such results generally reinforce that TensorRT optimisation positively affects the deployed models. Furthermore, the two models successfully tested in their optimised forms exhibited real gains in terms of classified images and the energy used by the system. Such a gain in performance-watt impacts the system positively so that more performance can be achieved for every watt spent running the classification process.

MobileNET has been the model that showed the best results in most selected metrics, especially the TRT-FP16 version, which achieved a performance-watt ratio of $26.01FPS/w$. This model version also scored well in terms of CPU and GPU usage. However, the model suffers from the memory ceiling problem. In terms of accuracy, the model achieved reasonably good levels. Applications have diverse requirements that need to be considered individually for bringing cloud models to edge devices. But considering the context established in the experimental work, under the specific conditions set, the MobileNET TRT-FP16 has emerged as a good candidate for edge deployment for the case study application: phytoplankton image classification as part of the ASTRAL project.

4.7 Key Findings

The experimental regime and the studied models generated interesting results. Below we summarise the key findings of this work:

- Transitioning the ML to a mid-range edge device such as Jetson NANO has a minimal impact on the model quality as measured by the traditional accuracy metric. This is reassuring because quality loss is always a critical risk in edge computing.
- TensorRT through the model compression optimisations can lead to substantial image throughput improvements. For example, we observed the FPS doubling for some models evaluated.
- TensorRT decreases the model's overall power consumption by moving part of the GPU workload to the CPU cores. The magnitude of this decrease varies from model to model, but it can be easily observed from the experimental data.
- When transitioning ML models to the edge, memory consumption is a significant system limitation. TensorRT performance is disappointing in this area. This is a conflicting situation as one should expect the opposite as quantisation in the optimisation pipeline tens to decrease memory usage (data compression). However, the TensorRT system needs to load extra software libraries, diminishing the actual gains delivered by model compression techniques. This is a significant concern that deserves further investigation, and initial work at Harvard University has already shed some light in quantifying all the extra overheads often invisible as part of *AI taxes accounting* [6].

5 CONCLUSION AND FUTURE WORK

This work explored the transition challenge whereby the cloud-based ML model is transferred to an embedded system device (edge). The work attempted to expose and understand the performance and resource usage limitations (mainly image throughput and power consumption) when deploying the models in a resource-constrained edge environment. In this way, our main interests in the research presented are the answers to the following 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 microscopic image analyser devices? And what is the impact that TensorRT has on the deployed models in terms of performance-watt?

To develop the work, we followed an evaluation methodology centred on several system stress experiments to gather information for key evaluation metrics. The work used the TensorRT tool as an optimisation engine for the underlying embedded hardware and analysed possible overheads and bottlenecks in the underlying system.

The experimental work evaluated possible trade-offs between model performance, accuracy, and resource usage. The analysis of the results has not established a clear-cut relationship between frame rate performance (image throughput) and high power consumption. The fastest high-performant models are not the most power-hungry. Furthermore, the deployment of the models at the edge did not affect their accuracy (related to RQ1).

TensorRT engine optimised a MobileNet model trained in the cloud (base model), resulting in a fine-tuned model for an embedded edge system (NVIDIA Jetson Nano). Optimised model versions were evaluated using float-point quantisation of 16 and 32 bits. Evaluation results of the power-consumption profile, resource usage, classification performance and image throughput suggest that TensorRT is an efficient option for machine learning edge computing (related to RQ2). TensorRT boosted the *performance-watt* of both models successfully tested. In addition, the optimisation engine doubled the image

throughput while enhancing its energy efficiency, improving the overall performance-watt trade-off. Results indicate that the library also transfers a portion of the GPU workload to the CPU thanks to the transformations made to the model's graph. However, application developers following these guidelines should take caution as the observed high memory usage presents a system limitation that must be carefully considered. The overheads outside the main inference pipeline must be taken into account. This is a strong message.

The objectives of the present work derive directly from the research questions we set to address. The work objectives were enumerated in the introductory chapter and will now be revisited to close the validation loop.

- **S01 - To investigate the accuracy of cloud-based models once deployed into edge devices [RQ1].**
  This work reports accuracy for all selected models in the cloud and edge devices. It also reports the accuracy results of the optimised models running in the target edge device, the Jetson NANO. For instance, the MobileNetV2 ACC results drop 0.01% as the model is embedded. Also, we observe no change with the optimised versions, which might indicate no over-fitting. The same decrease has been observed with the NASNetMobile. However, in this case, we observe another small drop with the optimised versions, in both cases 0.03%.

- **S02 - To investigate possible performance gain (in image FPS) in model post-optimisation [RQ1].**
  We report the results regarding image throughput for each model and variation. The selected models vary significantly in this metric. Nevertheless, the experiments confirm the supposed gains that TensorRT promises. A significant increase was observed in all cases where we could run the optimised models, mostly around doubling the base model result. MobileNetV2 and NASNetMobile double their image throughput, and the gains of the later model are slightly higher, jumping from 40 to 83 FPS. It is important to highlight the slight difference between optimisation options. The TRT-FP16 option reaches an increased FPS for both models. This quantisation achieved the best results for the NASNetMobile, 87 FPS.

- **S03 - To conduct a deep analysis of system power consumption [RQ2]**
  The present Master's dissertation carried out extensive analytical work on the power consumption results. We split the experiments into sections and delve into each section's energy profile. All models and variants' results were analysed. This way, we could compare the base models and between each model's base version and its optimised versions. First, we observe a substantial decrease in power consumption for all optimised versions compared to the base models. The more significant difference we observe is in the NASNetMobile model, going from $4236mW$ to around $3600mW$ in both optimised versions.

- **S04 - To analyse the resources used [RQ2].**
  This work explored the use of computational resources by running ML models. Results were reported from monitoring software for memory consumption and CPU and GPU usage. Such results were essential in determining which model and variants could be feasible for deployment in the target device. Unfortunately, not all models could be adequately assessed. For example, the optimised versions of VGG and Inception exhibited excessive memory-hungry behaviour to the point of crashing the system on every attempt.

- **S05 - To understand the performance-watt [RQ2].**
  The present work uses image throughput and power consumption results to create a complete cross-metric analysis. This performance-watt approach enables a comprehensive full picture of the model's overheads. And by this measure, we can understand how much it costs for each model to reach its performance. Data from this cross-analysis indicate MobileNetV2 in the TRT-FP16 version as the best performance-watt, with 26.01 FPS/W for the whole experiment and 14.33FPS/W considering only the inference time.

The present work contributed to the profiling and understanding of issues and limitations associated with taking ML-based phytoplankton classification models to a representative edge device system.

The main contribution of this work is the analysis of the performance and energy footprint (and their collective impact) on the overall edge system. The performance-watt study is revealing because it shows that the trend in system evaluation is better off using a two-dimensional metric rather than exploring those individually. Edge computing devices need to be low-cost and low-power but still deliver some performance to the application. The optimisation across such metrics is non-trivial and will inevitably lead to system trade-offs. Nevertheless, measuring and visualising energy usage and performance is a good practice for future developers.

This work left some questions to be explored in the future. First, it is important to investigate the mechanics of cold inference on edge devices more closely. The available literature led us to include a standardised warm-up step in our experiments. And the experimental stage showed the advantages of considering this phenomenon and counting on its existence. However, the experiments also demonstrated that each model experiences cold inference differently. So studying this phenomenon in depth in edge models and understanding the best warm-up range can lead to valuable findings.

Another critical point that can be explored in future works is to expand the range of models. Our methodology proved very useful for studying models deployed on the edge, generating valuable results for developing systems. This way, including more models than those used in our application, can create a useful dataset for edge system designers.

Finally, the most natural next step for the present work is to continue the development process of the embedded system as a whole. The studied models demonstrated their capabilities for the embedded application of phytoplankton classification. This beneficial result will be capitalised on in the upcoming research activities in the ASTRAL project.

REFERENCES

[1] Ahmed, E., Yaqoob, I., Hashem, I. A. T., Khan, I., Ahmed, A. I. A., Imran, M., and Vasilakos, A. V. (2017). The role of big data analytics in internet of things. *Computer Networks*, 129:459–471.

[2] Anderson, D. M., Andersen, P., Bricelj, V. M., Cullen, J. J., and Rensel, J. J. (2001). *Monitoring and management strategies for harmful algal blooms in coastal waters*. Unesco.

[3] Bianco, S., Cadene, R., Celona, L., and Napoletano, P. (2018). Benchmark analysis of representative deep neural network architectures. *IEEE access*, 6:64270–64277.

[4] Braque, M., De Nul, L., and Petridis, A. (2021). *Industry 5.0 : towards a sustainable, human-centric and resilient European industry*. European Commission, Directorate-General for Research and Innovation, Publications Office, Brussels.

[5] Brosnahan, M. L., Velo-Suárez, L., Ralston, D. K., Fox, S. E., Sehein, T. R., Shalapyonok, A., Sosik, H. M., Olson, R. J., and Anderson, D. M. (2015). Rapid growth and concerted sexual transitions by a bloom of the harmful dinoflagellate alexandrium fundyense (dinophyceae). *Limnology and Oceanography*, 60(6):2059–2078.

[6] Buch, M., Azad, Z., Joshi, A., and Reddi, V. J. (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)*, pages 96–106. IEEE.

[7] Buskey, E. J. and Hyatt, C. J. (2006). Use of the flowcam for semi-automated recognition and enumeration of red tide cells (karenia brevis) in natural plankton samples. *Harmful Algae*, 5(6):685–692.

[8] Campbell, L., Henrichs, D. W., Olson, R. J., and Sosik, H. M. (2013). Continuous automated imaging-in-flow cytometry for detection and early warning of karenia brevis blooms in the gulf of mexico. *Environmental Science and Pollution Research*, 20(10):6896–6902.

[9] Campbell, L., Olson, R. J., Sosik, H. M., Abraham, A., Henrichs, D. W., Hyatt, C. J., and Buskey, E. J. (2010). First harmful dinophysis (dinophyceae, dinophysiales) bloom in the us is revealed by automated imaging flow cytometry 1. *Journal of Phycology*, 46(1):66–75.

[10] Cao, K., Liu, Y., Meng, G., and Sun, Q. (2020). An overview on edge computing research. *IEEE access*, 8:85714–85728.

[11] Chen, Y., Wen, X., Zhang, Y., and He, Q. (2022). Fpc: Filter pruning via the contribution of output feature map for deep convolutional neural networks acceleration. *Knowledge-Based Systems*, 238:107876.

[12] Cho, D., Tai, Y.-W., and Kweon, I. S. (2018). Deep convolutional neural network for natural image matting using initial alpha mattes. *IEEE Transactions on Image Processing*, 28(3):1054–1067.

[13] Dapena, C., Bravo, I., Cuadrado, A., and Figueroa, R. I. (2015). Nuclear and cell morphological changes during the cell cycle and growth of the toxic dinoflagellate alexandrium minutum. *Protist*, 166(1):146–160.

[14] Deng, L., Yu, D., et al. (2014). Deep learning: methods and applications. *Foundations and trends® in signal processing*, 7(3–4):197–387.

[15] DeSA, U. (2015). *World population prospects: The 2015 revision, key findings and advance tables*. United Nations.

[16] Diez-Olivan, A., Del Ser, J., Galar, D., and Sierra, B. (2019). Data fusion and machine learning for industrial prognosis: Trends and perspectives towards industry 4.0. *Information Fusion*, 50:92–111.

[17] Ferdowsi, A., Challita, U., and Saad, W. (2019). Deep learning for reliable mobile edge analytics in intelligent transportation systems: An overview. *IEEE Vehicular Technology Magazine*, 14(1):62–70.

[18] (Food, F. and Organization), A. (1997). Bangkok fao technical consultation on policies for sustainable shrimp culture.

[19] Ge, W., Sun, J., Xu, Y., and Zheng, H. (2021). Real-time object detection algorithm for underwater robots. In *2021 China Automation Congress (CAC)*, pages 7703–7707. IEEE.

[20] Grosan, C. and Abraham, A. (2011). *Machine Learning*, pages 261–268. Springer Berlin Heidelberg, Berlin, Heidelberg.

[21] Guterres, B., Khalid, S., Pias, M., and Botelho, S. (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*, volume 2021. NeurIPS.

[22] Hallegraeff, G. M., Anderson, D. M., Cembella, A. D., and Enevoldsen, H. (2004). *Manual on harmful marine microalgae*. Unesco.

[23] Han, S., Pool, J., Tran, J., and Dally, W. (2015). Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28.

[24] Harred, L. B. and Campbell, L. (2014). Predicting harmful algal blooms: a case study with dinophysis ovum in the gulf of mexico. *Journal of plankton research*, 36(6):1434–1445.

[25] Henrichs, D. W., Sosik, H. M., Olson, R. J., and Campbell, L. (2011). Phylogenetic analysis of brachidinium capitatum (dinophyceae) from the gulf of mexico indicates membership in the kareniaceae 1. *Journal of Phycology*, 47(2):366–374.

[26] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

[27] Huang, S., Ankit, A., Silveira, P., Antunes, R., Chalamalasetti, S. R., El Hajj, I., Kim, D. E., Aguiar, G., Bruel, P., Serebryakov, S., et al. (2021). Mixed precision quantization for reram-based dnn inference accelerators. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 372–377. IEEE.

[28] Joshi, P., Afli, H., Hasanuzzaman, M., Thapa, C., and Scully, T. (2022). Enabling deep learning for all-in edge paradigm. *arXiv preprint arXiv:2204.03326*.

[29] Jošilo, S. and Dán, G. (2019). Selfish decentralized computation offloading for mobile cloud computing in dense wireless networks. *IEEE Transactions on Mobile Computing*, 18(1):207–220.

[30] Khan, W. Z., Ahmed, E., Hakak, S., Yaqoob, I., and Ahmed, A. (2019). Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235.

[31] Khelifi, H., Luo, S., Nour, B., Sellami, A., Moungla, H., Ahmed, S. H., and Guizani, M. (2019). Bringing deep learning at the edge of information-centric internet of things. *IEEE Communications Letters*, 23(1):52–55.

[32] Lai, Q. T., Lee, K. C., Tang, A. H., Wong, K. K., So, H. K., and Tsia, K. K. (2016). High-throughput time-stretch imaging flow cytometry for multi-class classification of phytoplankton. *Optics Express*, 24(25):28170–28184.

[33] Larsen, K. R. and Becker, D. S. (2021). *Automated machine learning for business*. Oxford University Press.

[34] Lee, M.-S., Park, K.-A., Chae, J., Park, J.-E., Lee, J.-S., and Lee, J.-H. (2020). Red tide detection using deep learning and high-spatial resolution optical satellite imagery. *International Journal of Remote Sensing*, 41(15):5838–5860.

[35] Lehman, P., Kendall, C., Guerin, M., Young, M., Silva, S., Boyer, G., and Teh, S. J. (2015). Characterization of the microcystis bloom and its nitrogen supply in san francisco estuary using stable isotopes. *Estuaries and Coasts*, 38(1):165–178.

[36] Lehman, P., Marr, K., Boyer, G., Acuna, S., and Teh, S. J. (2013). Long-term trends and causal factors associated with microcystis abundance and toxicity in san francisco estuary and implications for climate change impacts. *Hydrobiologia*, 718(1):141–158.

[37] Li, F.-F., Fergus, R., and Perona, P. (2006). One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611.

[38] Li, Z., Gong, Y., Ma, X., Liu, S., Sun, M., Zhan, Z., Kong, Z., Yuan, G., and Wang, Y. (2020). Ss-auto: A single-shot, automatic structured weight pruning framework of dnns with ultra-high efficiency. *arXiv preprint arXiv:2001.08839*.

[39] Min, X., Li, W., Yang, J., Xie, W., and Zhao, D. (2022). Self-supervised graph neural network with pre-training generative learning for recommendation systems. *Scientific Reports*, 12.

[40] Moore, R. E. (2011). *FlowCAM® Manual Version 3.0*. Fluid Imaging Technologies, Inc.

[41] Munoz, A. (2014). Machine learning and optimization. *URL: https://www. cims. nyu. edu/˜ munoz/files/ml_optimization. pdf [accessed 2016-03-02][WebCite Cache ID 6fiLfZvnG]*.

[42] Namiot, D., Ilyushin, E., and Chizhov, I. (2021). Military applications of machine learning. *International Journal of Open Information Technologies*, 10(1):69–76.

[43] Nilsson, N. J. and Nilsson, N. J. (1998). *Artificial intelligence: a new synthesis*. Morgan Kaufmann.

[44] Olson, R., Vaulot, D., and Chisholm, S. (1985). Marine phytoplankton distributions measured using shipboard flow cytometry. *Deep Sea Research Part A. Oceanographic Research Papers*, 32(10):1273–1280.

[45] Olson, R. J. and Sosik, H. M. (2007). A submersible imaging-in-flow instrument to analyze nano-and microplankton: Imaging flowcytobot. *Limnology and Oceanography: Methods*, 5(6):195–203.

[46] Ongsulee, P. (2017). Artificial intelligence, machine learning and deep learning. In *2017 15th international conference on ICT and knowledge engineering (ICT&KE)*, pages 1–6. IEEE.

[47] O'Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

[48] Pollina, T., Larson, A. G., Lombard, F., Li, H., Colin, S., de Vargas, C., and Prakash, M. (2020). Planktonscope: Affordable modular imaging platform for citizen oceanography. *bioRxiv*.

[49] Pouyanfar, S., Sadiq, S., Yan, Y., Tian, H., Tao, Y., Reyes, M. P., Shyu, M.-L., Chen, S.-C., and Iyengar, S. S. (2018). A survey on deep learning: Algorithms, techniques, and applications. *ACM Comput. Surv.*, 51(5).

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

[51] Reisizadeh, A., Mokhtari, A., Hassani, H., Jadbabaie, A., and Pedarsani, R. (2020). Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*, pages 2021–2031. PMLR.

[52] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

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

[54] Seichter, D., Köhler, M., Lewandowski, B., Wengefeld, T., and Gross, H.-M. (2021). Efficient rgb-d semantic segmentation for indoor scene analysis. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13525–13531. IEEE.

[55] Shadrin, D., Menshchikov, A., Somov, A., Bornemann, G., Hauslage, J., and Fedorov, M. (2019). Enabling precision agriculture through embedded sensing with artificial intelligence. *IEEE Transactions on Instrumentation and Measurement*, 69(7):4103–4113.

[56] Shafi, O., Rai, C., Sen, R., and Ananthanarayanan, G. (2021). Demystifying tensorrt: Characterizing neural network inference engine on nvidia edge devices. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*, pages 226–237. IEEE.

[57] Shailaja, K., Seetharamulu, B., and Jabbar, M. (2018). Machine learning in healthcare: A review. In *2018 Second international conference on electronics, communication and aerospace technology (ICECA)*, pages 910–914. IEEE.

[58] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[59] Siswanto, E., Ishizaka, J., Tripathy, S. C., and Miyamura, K. (2013). Detection of harmful algal blooms of karenia mikimotoi using modis measurements: A case study of seto-inland sea, japan. *Remote Sensing of Environment*, 129:185–196.

[60] SOFIA, F. (2018). The state of world fisheries and aquaculture 2018-meeting the sustainable development goals. *Fisheries and Aquaculture Department, Food and Agriculture Organization of the United Nations, Rome*.

[61] Su, L., Yang, X., Cao, B., Wang, Y., Li, X., and Lu, W. (2021). Development and application of substation intelligent inspection robot supporting deep learning accelerating. In *Journal of Physics: Conference Series*, volume 1754, page 012170. IOP Publishing.

[62] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[63] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[64] Tao, L., Hong, T., Guo, Y., Chen, H., and Zhang, J. (2020). Drone identification based on centernet-tensorrt. In *2020 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pages 1–5. IEEE.

[65] Tilman, D. and Clark, M. (2014). Global diets link environmental sustainability and human health. *Nature*, 515(7528):518–522.

[66] Traller, J. C. and Hildebrand, M. (2013). High throughput imaging to the diatom cyclotella cryptica demonstrates substantial cell-to-cell variability in the rate and extent of triacylglycerol accumulation. *Algal Research*, 2(3):244–252.

[67] Wahab, O. A., Mourad, A., Otrok, H., and Taleb, T. (2021). Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems. *IEEE Communications Surveys & Tutorials*, 23(2):1342–1397.

[68] Wells, M. L., Trainer, V. L., Smayda, T. J., Karlson, B. S., Trick, C. G., Kudela, R. M., Ishikawa, A., Bernard, S., Wulff, A., Anderson, D. M., et al. (2015). Harmful algal blooms and climate change: Learning from the past and present to forecast the future. *Harmful algae*, 49:68–93.

[69] Yang, D., Yu, W., Mu, H., and Yao, G. (2021). Dynamic programming assisted quantization approaches for compressing normal and robust dnn models. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pages 351–357.

[70] Yi, R., Cao, T., Zhou, A., Ma, X., Wang, S., and Xu, M. (2022). Understanding and optimizing deep learning cold-start latency on edge devices. *arXiv preprint arXiv:2206.07446*.

[71] Yue, W., Wang, Z., Chen, H., Payne, A., and Liu, X. (2018). Machine learning with applications in breast cancer diagnosis and prognosis. *Designs*, 2(2):13.

[72] Zhao, J. and Ghedira, H. (2014). Monitoring red tide with satellite imagery and numerical models: A case study in the arabian gulf. *Marine pollution bulletin*, 79(1-2):305–313.

[73] Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., and Zhang, J. (2019). Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762.

[74] Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710.