

UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Technology
Robotics and Computer Engineering Curriculum

Maxence ROUCHOU

Development of an Adaptive Pipeline for Object Detection Training and Benchmarking

Master's Thesis (30 ECTS)

Supervisors: Ric Dengel
Quazi Saimoon Islam

Tartu 2024

Development of an Adaptive Pipeline for Object Detection Training and Benchmarking

Abstract:

This thesis presents the development of a highly modular pipeline designed for the training, testing and benchmarking of object detection models for future extraterrestrial surface exploration. The key focus for these models is to achieve an optimal balance between inference time, power consumption, and model precision, which is paramount given the limited computational capabilities and power restrictions of resource-constrained devices. Five models (FRCNNs and YOLOs) were chosen for their varied characteristics. They were then implemented in this pipeline and their accuracy, power consumption, inference time, and CPU & GPU usage were collected. With the collected metrics, Yolo11s was determined to be the most suitable for autonomous navigation out of the implemented models. The developed pipeline lays the foundation for further study and research into the domain and identifying models that could be deployed in future space applications.

Keywords: Neural Network Benchmark, Object Detection, Model optimization, Jetson Orin Nano

CERCS: T111: Imaging, image processing, T125: Automation, robotics, control engineering;

Kohandatava toru arendamine objektide tuvastamise koolitamiseks ja hindamiseks

Lühikokkuvõte:

Kohanduva treenimis- ja hindamisliini arendus objektide tuvastamise mudelitele. See lõputöö esitleb modulaarset süsteemi, et treenida, testida ja hinnata objektide tuvastamise mudeleid tulevaseks maavälise pinna avastamiseks. Töö fookuses on optimaalse tasakaalu leidmine mudeli järeldamisaja, nõutava arvutusvõimsuse ning mudeli täpsuse vahel, sest arvutusjõudlus ning volutarbimine on seadmetel piiratud. Valiti viis mudelit (FRCNN ja YOLO algoritmidel baseeruvad) nende varieeruvate karakteristikute poolest. Need mudelid implementeeriti treenimis- ja hindamisliini, mõõdeti nende täpsust, volutarbimist, järeldamisega ning protsessorite kasutust. Kogutud andmete põhjal leiti, et autonoomseks navigatsiooniks on implementeeritud mudelitest parim YOLO11. Tehtud töö loob aluse edasisteks valdkonnasisesteks uuringuteks kosmoserakendusteks kasutatavate mudelite valimisel.

Võtmesõnad:

CERCS: T111: Pilditehnika, T125: Automatiseerimine, robotika, juhtimistehnika,

Contents

1	Introduction	8
1.1	Background	8
1.2	Problem statement	9
1.3	Goals	9
2	Literature Review	10
2.1	Previous Work in Autonomous Navigation for Rovers	10
2.1.1	Evolution of Rover Navigation	10
2.1.2	Advances in Neural Network Architectures	10
2.1.3	Optimization Techniques for Resource Efficiency	11
2.1.4	Real-World Applications and Challenges	12
2.2	Future Possibilities in Neural Network Applications for Lunar Rover Navigation	12
2.2.1	Enhanced Terrain Classification	12
2.2.2	Integration of Spiking Neural Networks	13
2.3	Neural Networks Choice	13
2.4	Possible Target Devices	15
2.4.1	Difference between main Hardware Components	15
2.4.2	Nvidia Jetson	16
2.4.3	AMD Versal	18
2.4.4	Google Coral	19
3	Methodology	20
3.1	Chosen Models	20
3.2	Dataset	20
3.3	Chosen Target Device	21
3.4	Pipeline Overview	22
3.4.1	Configuration File	22
3.4.2	Virtual Environments	24
3.4.3	Model Training	24
3.4.4	Model Testing	25
3.4.5	Model & Config Copy	26
3.4.6	Model Benchmarking	26
4	Results and analysis	29
4.1	Input Variables	29
4.2	Results	30
4.2.1	Total Time and Individual Times	30
4.2.2	Inference Speed and Model Accuracy	32

4.2.3	Power Consumption	33
4.2.4	CPU & GPU	35
4.2.5	RAM	37
4.3	Analysis of the Results	37
5	Discussion	38
6	Conclusion	39
	References	44
	Glossary	45
	Tools	45
	Licence	46

List of Tables

1	Performance and Specifications of object detection models from PyTorch documentation [1]	14
2	Performance and Specifications of YOLO Models [2]	14
3	Performance Highlights of NVIDIA Jetson Devices [3]	16
4	Specifications of Versal AI Edge XA Models	18
5	Performance comparison of models based on Frames Per Second (FPS) and precision.	20
6	Specifications of Jetson Orin Nano	22
7	Configuration Variables Definition	23
8	Overview of metrics used for benchmarking.	27
9	Image Augmentations for Yolo models	29
10	Image Augmentations for PyTorch FRCNN models	30
11	First Pass Times for Object Detection Models	31
12	Performance Comparison of the 5 Benchmarked Models based on FPS and Model Score.	32
13	Performance Comparison of Models	37

List of Figures

1	Hardware efficiency [4]	15
2	Jetson AGX Xavier and Jetson AGX Orin Comparison [5]	17
3	Space Bunker	21
4	Input Image from the Dataset	21
5	Same Image with Annotations	21
6	Pipeline Diagram	22
7	Block Diagram of PyTorch Model Training	25
8	Block Diagram of TensorFlow 2 Model Training	25
9	Block Diagram of the Model Benchmarking	28
10	Inference Times for Each Iteration of the 5 Benchmarked Models	31
11	Power Consumption Of the 5 Benchmarked Models	33
12	Power Consumption of the 5 Benchmarked Models	34
13	CPU & GPU Usage of the 5 Benchmarked Models	35
14	CPU Frequency of the 5 Benchmarked Models	36

List of Acronyms

AI Artificial Intelligence

ASIC Application-specific integrated circuit

CNN Convolutional Neural Network

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

FOV Field Of View

FPGA Field Programmable Gate Arrays

FPS Frames Per Second

FRCNN Faster RCNN

GFLOPS Giga Floating Point Operations Per Second

GPU Graphics Processing Unit

JSON JavaScript Object Notation

MPU Microprocessor Unit

RL Reinforcement Learning

SL Supervised Learning

SNN Spiking Neural Network

SoC System-on-Chip

TPU Tensor Power Unit

UL Unsupervised Learning

XLA Accelerated Linear Algebra

YOLO You Only Look Once

1 Introduction

1.1 Background

Interest in lunar exploration has increased in recent years due to several significant factors. The first factor is that the Moon is an entryway for deeper space exploration [6] and is the perfect testing ground for technologies required for missions to Mars and beyond. More specifically, its proximity to Earth allows for shorter and less costly missions, making it the best place for research and development in space engineering. The second main factor was the discovery of water ice near the poles of the Moon [7]. This translates to a potential source of drinkable water. In addition, electrolysis of water separates the H_2O into O_2 (oxygen for breathable air) and H_2 (can be used for Rocket Fuel). Finally, the rise of new commercial space programs such as SpaceX has made lunar exploration more accessible, leading to a new era of space exploration [8].

Rovers play an important role in the increase in interest in lunar exploration. They are vital for conducting scientific investigations and acquiring data in harsh or remote areas [9]. Rovers equipped with advanced technologies are beginning to autonomously explore the rough terrain of the Moon, including craters, rocks, and varied surface textures, but currently still need a human operator to navigate these terrains [10].

Traditional autonomous navigation algorithms rely primarily on rule-based systems and manually created features to detect and avoid obstacles. Although these technologies have been extensively tested and are efficient in a controlled environment, they struggle with other planets' more varied and complex terrains, where obstacles and geological formations are more common [11]. Said algorithms lack the adaptability to handle these conditions, especially when adding variable lighting, shadows, and surface textures [12]. Neural networks, on the other hand, are based on how the human brain works. Their main advantage over standard algorithms is their ability to generalize concepts from a set of information. Starting from a huge number of training data, they can detect patterns and anomalies that traditional algorithms may miss, resulting in more accurate and reliable navigation [13].

However, neural network models have some disadvantages to take into account, the most important being their computational costs [14]; this cost is difficult to accommodate on resource-limited space systems. Luckily, recent advancements have made powerful Graphics Processing Unit (GPU) technology more accessible for space applications [15]. As miniaturization and energy efficiency of computing hardware improvements occur, this issue is becoming decreasingly problematic [16] but still needs to be addressed. Another constraint with missions in the space environment is the limitation in power resources, where energy conservation is vital for prolonged mission duration. Overall,

any neural network used in a space project must achieve a delicate balance between speed, accuracy, and power efficiency [17]. More specifically for rover applications, the goal is to deploy models that maximize detection capabilities, allowing for safe autonomous navigation while consuming the lowest amount of energy in challenging extraterrestrial environments.

To this end, this thesis aims to develop a benchmarking pipeline for neural network solutions that enables quick and easy evaluation of various neural network architectures and their performance on an edge device. With this tool, this research seeks to identify the best model and configuration that can accurately be used for navigation while operating within the constraints of computationally and energetically limited space systems.

1.2 Problem statement

Autonomous navigation for planetary rovers requires a careful balance of model precision, inference speed, and power consumption, which are essential for detecting obstacles in real-time. This balance is especially challenging on compact hardware, which has limited computational resources and must operate with restricted power.

This thesis aims to develop a modular pipeline to easily experiment with different neural network models and configurations for object detection of lunar rocks. By building a flexible benchmarking system, the project seeks to help find the optimal trade-off between speed, accuracy, and power efficiency for rock detection.

1.3 Goals

In response to the identified challenges, this project aims to achieve the following goals:

- Develop a highly modular pipeline that aims to train, test, and benchmark different neural networks with variable configurations.
- Identify the most relevant metrics to accurately benchmark the trained deep learning models.
- Determine which model and which configuration achieves the best balance between inference speed, accuracy, and power consumption.

2 Literature Review

2.1 Previous Work in Autonomous Navigation for Rovers

Over the years, the number of studies related to Autonomous navigation especially focusing on deep learning has skyrocketed. This section will be diving into the evolution of autonomous navigation for rovers, then the improvements that were made related to neural networks, followed by various optimization techniques for models to be lighter and faster, and finally the challenges related to applying the neural networks into real-life extraterrestrial exploration.

2.1.1 Evolution of Rover Navigation

Autonomous navigation was already implemented as early as the first Mars Exploration Rover with a self-driving technology called AutoNav [18]. Opportunity, one of the two Mars Exploration Rovers, only traveled 5% of its total distance with this technology. This system was improved over the years for subsequent missions: Curiosity and especially Perseverance, which has autonomously traveled more than 85% of its total distance with a record 699.9 meters without human review [19]. Despite this impressive feat, the maximum distance traveled per day remains limited, the maximum attained by Perseverance being only 347.7 meters, and regular human intervention is still needed.

2.1.2 Advances in Neural Network Architectures

In the field of neural networks, there are three main learning paradigms:

- Supervised Learning (SL): Training a neural network to predict values based on labeled data.
- Unsupervised Learning (UL): Training a neural network to distinguish patterns, without labeled data.
- Reinforcement Learning (RL): Training a neural network where its interactions with its environment awards rewards or penalties.

A first approach towards autonomous navigation using SL would be through the use of Convolutional Neural Network (CNN). This type of neural network is specifically aimed for computer vision, in that sense, their use is paired with navigation. Their use in the context of extraterrestrial rovers was studied for image classification of terrain types and terrain features [20]. It was shown that the use of CNN leads to high accuracy for spatial but most importantly for us, ground-based images.

Another approach towards autonomy using RL was also studied for navigation tasks [21]. Generally, RL neural networks have a very long training time and may lead to dead ends during navigation. In this paper, a new framework was developed, combining traditional navigation and RL, keeping the advantages of RL while mitigating its weaknesses. Their novel approach beat both end-to-end RL approaches and traditional navigation processes. This approach, even though never implemented in extraterrestrial environments could be implemented in future studies.

2.1.3 Optimization Techniques for Resource Efficiency

For autonomous navigation in resource-constrained systems, it is mandatory to maximize accuracy and detection speed while having the lowest power consumption possible. When working with neural networks, multiple techniques were developed to improve model inference speed while very slightly affecting precision, such as low-rank factorization or knowledge distillation [22]. However, the two most effective techniques are model pruning and quantization [23]. There are different algorithms to perform pruning on models, but the aim remains the same: delete some neurons of the neural network alongside its connections to other neurons, leading to lighter models. Similar to pruning, there are many different types of quantization, either pre or post-training. The idea is to reduce the precision of neuron weights, usually from float of 32 or 16 bits to integers of 8 bits, leading to faster models, as less bits means smaller computation. It was demonstrated that the implementation of these techniques greatly improved the inference speed of tested models without any significant decrease in model accuracy [23].

While pruning and quantization are techniques applied to models, light models are also being developed, aiming for the lowest inference time while keeping a high accuracy. It is the case of the last version of the MobileNet models, MobileNetV3 [24]. The main technique used in this architecture called Depthwise Separable Convolutions, breaks down the standard convolution into two steps: Depthwise convolution and Pointwise Convolution. This results in a lower computational complexity and a lower amount of parameters, leading to faster inference time while keeping high accuracy. This type of lightweight model is perfect for low-power devices and is a good contender for our lunar rover application.

Now that we know how to create a small and efficient model with lightweight architecture, pruning, and quantization, we can dig into inference optimizers such as TensorRT and Accelerated Linear Algebra (XLA). TensorRT is a high-performance deep learning inference library developed by NVIDIA that optimizes trained models for deployment specifically on NVIDIA GPU. It applies various techniques such as layer fusion, precision calibration, and kernel auto-tuning to maximize inference speed, making it perfect for deployment on devices that require real-time inference [25].

On the other hand, XLA is a compiler for linear algebra that optimizes computations by generating efficient machine code tailored to specific hardware. By utilizing XLA optimization, model speed can significantly improve through graph optimizations and reduced memory overhead [26]. The choice of one of these two optimization techniques can significantly enhance the efficiency of neural networks.

2.1.4 Real-World Applications and Challenges

The use of neural networks still has some downsides that have already been studied. When training neural networks with a low amount of data, the risk of overfitting is high [27]. This problem is particularly important for planetary rovers which do not have an unlimited amount of data. However, it was demonstrated that using pre-trained models and fine-tuning them on a limited amount of data from the Martian environment would lead to pretty accurate models [28], thus enhancing the feasibility of deploying neural networks in extraterrestrial scenarios.

Moreover, sensor fusion techniques that combine data from various sources, such as adding LiDAR and odometry values to camera input, were developed to improve navigation performance under challenging conditions [29]. This study showed that it might be wiser to rely on multiple sensor inputs than only the output of neural networks.

2.2 Future Possibilities in Neural Network Applications for Lunar Rover Navigation

As the field of autonomous navigation for lunar rovers continues to evolve, several new possibilities emerge. In this field, the integration of neural network architectures and innovative techniques is a hot topic. This section explores potential advancements, including enhanced terrain classification, and the application of Spiking Neural Network (SNN).

2.2.1 Enhanced Terrain Classification

The ability to accurately classify terrain is vital for ensuring safe navigation and effective mission planning on the Moon. Recent advancements have shown promise in using deep learning techniques for automated image classification of lunar features. For example, a study focused on using CNN demonstrated high accuracy in identifying geological features from lunar imagery captured by robotic rovers [30]. This study shows that the implementation of neural networks for extraterrestrial environments is not only theoretically possible but even already possible to implement.

It is possible to imagine the future of more robust and more accurate deep learning models that could be created from bigger datasets due to the renewed interest in lunar

exploration. This paired with transfer learning and domain adaptation could enable models trained on Earth-based data to be effectively adapted for lunar environments, enhancing their robustness in on-field applications.

2.2.2 Integration of Spiking Neural Networks

Spiking Neural Networks are a significant evolution compared to traditional neural network models. The idea behind SNN is to resemble the human brain more closely, by processing information with spikes and adding a time dimension to these networks. Research showed that this type of neural network requires less power to be run, and is faster than CNN [31].

However, multiple challenges are currently linked to the use of SNN.

- Training SNN is more complex than training traditional neural networks. As they are based on spikes, these signals are non-differentiable so traditional backpropagation by gradient descent is non-applicable.
- Only a few amount of hardware (such as Intel Loihi [32]) currently support SNNs. This reason is the main issue regarding its use in this pipeline.
- Only a few amount of research was done about the use of SNN with object detection, which is the task performed in this study.

Given these limitations, while SNN is promising for future implementation and wider use, further research is necessary to mitigate the aforementioned issues.

2.3 Neural Networks Choice

In the vast world of neural networks, new models emerge regularly, making the selection process quite fastidious. While it is possible to develop a custom model from scratch, this approach can be time-consuming and does not guarantee impressive results, which could lead to a waste of time. As already covered previously, a better strategy is to use pre-trained models which can then be fine-tuned for specific tasks. In this section, we will explore several object detection models and evaluate their suitability for our study.

Tables 1 and 2 show us key metrics for different object detection models. Model accuracies are designated by their mAP score (the higher the better) and their computational cost is represented by the number of Giga Floating Point Operations Per Second (GFLOPS) (the higher, the more computationally intensive the model is)

When considering model performance, it is essential to benchmark both highly accurate but slower models -such as Faster RCNN (FRCNN) with ResNet50 backbone, RetinaNet with Resnet50 backbone, or Yolo11x models- but also fast and less accurate ones -such as SSDLite320, FRCNN with MobileNet Backbone or Yolo11n.

Model	Box MAP	Params	GFLOPS
FCOS_ResNet50_FPN	39.2	32.3M	128.21
FasterRCNN_MobileNet_V3_Large_320_FPN	22.8	19.4M	0.72
FasterRCNN_MobileNet_V3_Large_FPN	32.8	19.4M	4.49
FasterRCNN_ResNet50_FPN_V2s	46.7	43.7M	280.37
FasterRCNN_ResNet50_FPNs	37.0	41.8M	134.38
RetinaNet_ResNet50_FPN_V2	41.5	38.2M	152.24
RetinaNet_ResNet50_FPN	36.4	34.0M	151.54
SSD300_VGG16	25.1	35.6M	34.86
SSDLite320_MobileNet_V3_Large	21.3	3.4M	0.58

Table 1. Performance and Specifications of object detection models from PyTorch documentation [1]

Model	mAPval 50-95	Params	GFLOPS
YOLO11n	39.5	2.6 M	6.5
YOLO11s	47.0	9.4 M	21.5
YOLO11m	51.5	20.1 M	68.0
YOLO11l	53.4	25.3 M	86.9
YOLO11x	54.7	56.9 M	194.9

Table 2. Performance and Specifications of YOLO Models [2]

The more accurate models are amazing at detecting objects but often have highly increased inference time, which is not ideal for real-time applications. Conversely, the lightweight models offer faster processing speeds, which is important for real-time detection but comes with a trade-off in precision.

To strike a balance between accuracy and speed, we can also consider two models from the You Only Look Once (YOLO) family: YOLO11s and YOLO11l. The choice of Yolo11 was made as it is the newest version of this model, but most importantly it is faster and more precise than its predecessors [2]. The YOLO11s model achieves a reasonable mAP score while seeming to have rapid inference times, making it an excellent choice for our trade-off problem. Meanwhile, YOLO11l provides higher accuracy than YOLO11s while still being faster than the more complex FRCNN with ResNet50 backbone. By benchmarking these models, we can decipher which configuration would be more suitable for an autonomous navigation application.

2.4 Possible Target Devices

The selection of appropriate target devices for benchmarking is crucial for the performance evaluation of neural network models in real-world applications, especially in resource-constrained environments. The choice of hardware directly impacts the efficiency, speed, and accuracy of the object detection algorithms deployed on these platforms.

2.4.1 Difference between main Hardware Components

Before digging into which device we could use, it is interesting to think about the reasons why some devices are better than others.

First, we can compare different hardware in terms of energy efficiency. To this end, figure 1 was created, showing the number of operations per energy consumed (GOP/J), over the years [4].

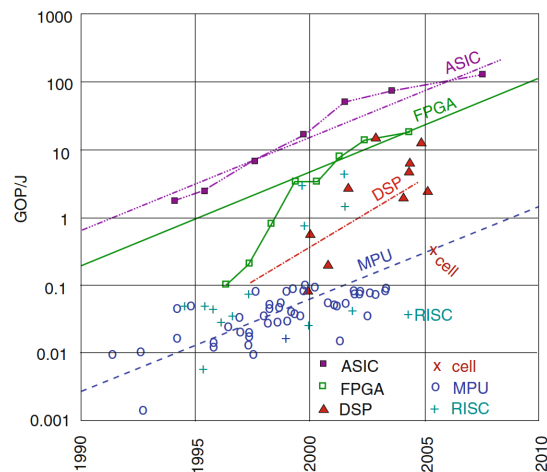


Figure 1. Hardware efficiency [4]

In this figure, multiple technologies are compared but the ones that we are interested in are Application-specific integrated circuit (ASIC) and Field Programmable Gate Arrays (FPGA). Digital Signal Processors (DSP) are designed for signal processing, which is not in the scope of this thesis, and Microprocessor Unit (MPU), cell and Reduced Instruction Set Computer (RISC) are designed for general purposes.

ASICs are custom chips designed for a specific task. Due to this and as can be seen in figure 1, they are very energy efficient. This makes them the primary target device for

this thesis. Coming in close second in terms of energy efficiency are FPGAs. These integrated circuits are reprogrammable which makes them less efficient than ASICs. Their high efficiency still makes them an interesting target to consider. The primary difference between those two hardware is that ASIC logic is set on a silicon level (Very efficient, but not modifiable) while FPGA have circuits that can take on any logic (reprogrammable but adds an overhead) [33].

In this thesis, the aim is to train, test, and benchmark object detection models, and these models are all based on CNN. With this type of neural network, the calculations are made using tensors. For example, in CNNs, input images are often stored in four-dimension tensors of shape [B, C, H, W]. B represents the Batch size, C is the number of channels in the image (3 for RGB Images), and H and W are the Height and Width of the images. In a nutshell, we need to find devices similar to ASICs or FPGA, designed for tensor calculations.

2.4.2 Nvidia Jetson

Nvidia Jetson is a series of low-power embedded computing boards designed for Artificial Intelligence (AI) applications. They are System-on-Chip (SoC), which means that they integrate a graphical computational hardware that is called GPU and a more general one called Central Processing Unit (CPU). Their GPU contains Compute Unified Device Architecture (CUDA) cores, designed for parallel computing and cores specialized for tensor calculations. This specialized GPU part can be considered as an ASIC. Knowing that and from our previous study, these devices should be very energy efficient for our deep learning application. Table 3 lists the different generations of Jetson devices (Nano, TX2, Xavier, and Orin) as well as the models from these generations.

Jetson Device	AI Performance	GPU Cores	RAM	Power (W)
Jetson Nano	0.472 TOPS	128	4 GB	5 - 10
Jetson TX2	1.26 TOPS	256	4 - 8 GB	7.5 - 20
Jetson Xavier NX	21 TOPS	384	8 - 16 GB	10 - 20
Jetson AGX Xavier	30 - 32 TOPS	512	32 - 64 GB	10 - 40
Jetson Orin Nano	34 - 67 TOPS	512 - 1024	4 - 8 GB	7 - 25
Jetson Orin NX	117 - 157 TOPS	1024	8 - 16 GB	10 - 40
Jetson AGX Orin	200 - 275 TOPS	1792 - 2048	32 - 64	15 - 75

Table 3. Performance Highlights of NVIDIA Jetson Devices [3]

From this table, it is possible to make some conclusions. As we are searching for a power-efficient device, it is important to compare devices with similar power consumption.

- Jetson Nano is a low-power, entry-level model but its very low AI performance is not interesting for this study.
- Jetson TX2 is the oldest generation out of the one compared which makes it pretty outdated. It can be seen with its low AI performance while having a power consumption relatively similar to the Jetson Xavier NX or the Jetson Orin Nano.
- Jetson Xavier Nx and Jetson Orin Nano have a similar power consumption, however, Jetson Orin Nano has a better AI performance which makes it more interesting for the benchmark. It is still important to denote that the Xavier has twice the RAM as the Orin
- Similarly Jetson AGX Xavier and Jetson Orin NX have comparable power consumption. Also, the same remark can be made as the previous point, in the sense that the Orin has a way higher AI Performance while the Xavier has twice as much RAM.
- Finally Jetson AGX Orin is the device with the higher power consumption and higher AI performance.

Experimental studies were conducted to verify these Nvidia marketing metrics such as the one depicted in figure 2.

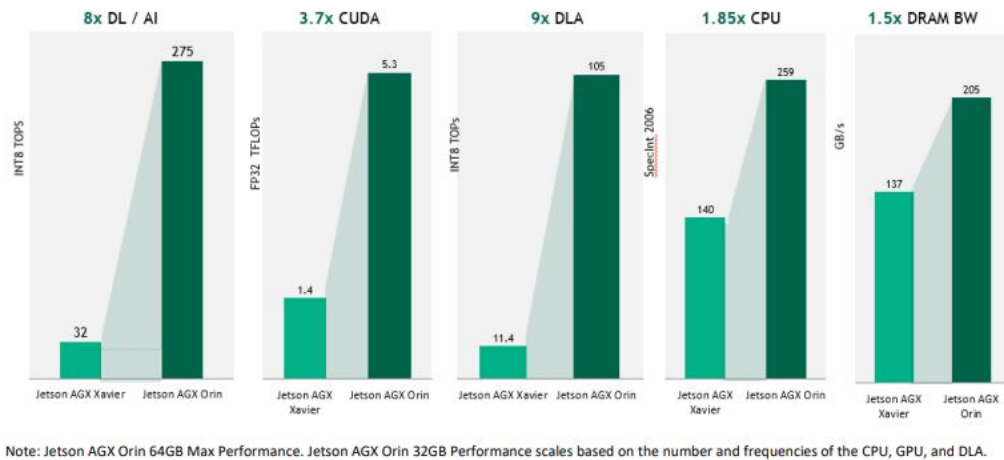


Figure 2. Jetson AGX Xavier and Jetson AGX Orin Comparison [5]

This figure confirms that the newer Orin generation is way more computationally powerful than the older Xavier, by comparing a Jetson AGX Xavier and a Jetson AGX Orin. The Orin can produce ~8x more operations per second than the Xavier while being

only ~1.5x more energy consuming.

For neural network benchmarking, AI performance has way more importance than RAM. It means that the three models from the Orin generation are the more interesting devices of the Jetson family. Then, the choice between those three devices will be determined by the power budget of the mission on which the device will be used. In the case of a small lunar rover like KuupKulgur, the AGX Orin might be too high in power consumption so the Orin Nano and NX are better options.

2.4.3 AMD Versal

Just like Nvidia Jetsons, AMD Versal are SoC devices. The Versal family that is the most relevant to our study is the AMD Versal AI Edge, as its name suggests, is designed for AI applications. They are composed of an FPGA part, a certain number of specialized "AI Engine", and a CPU. There are currently three generations to this family, the first generation, the second generation, and the XA, a more specialized version of the second generation [34]. Generation 2 is advertised as way more powerful and efficient than Gen 1 [35], so there is no need to dig into the first generation. Moreover, as the XA models are the most optimized, it is the ones with the best power efficiency so we should consider these models.

Table 4 presents the specifications of the different models of the AMD Versal AI Edge XA Series.

Model	AI Performance	AI Engine-ML Tiles	Power Consumption
XAVE2002	5 TOPS	8	6-9 W
XAVE2102	7 TOPS	12	7-10 W
XAVE2202	15 TOPS	24	15-20 W
XAVE2302	22 TOPS	34	22 W
XAVE2602	89 TOPS	152	50-60 W
XAVE1752	92 TOPS	304	50-60 W
XAVE2802	171 TOPS	304	75 W

Table 4. Specifications of Versal AI Edge XA Models

Similarly to the model selection, it is complicated to compare the AI Performance of these models to the Jetson devices, as the operation is not the same for these two tables. Nonetheless, it is still interesting to compare these devices in terms of power consumption. The devices XAVE2102 and XAVE2202 have a power consumption and an AI Performance in the same order of magnitude as the Jetson Orin Nano. This

would suggest a similar power efficiency, to this device, so these two devices would be interesting target devices for this study.

2.4.4 Google Coral

Finally, another possible device could be the Google Coral Tensor Power Unit (TPU). As its name suggests, it is an ASIC designed for tensor calculation. It is pretty powerful for deep learning applications with 4 TOPS and nicely power efficient at 2 TOPS per watt [36]. However, as studies were made [37], it is more comparable to the Jetson Nano rather than the more power-consuming Jetson Orin, which makes this device less interesting than the latter family.

3 Methodology

This section outlines the methodology employed in this study, detailing the selection of models, the dataset utilized for training and testing, the target device for benchmarking, and finally the structure of the pipeline.

3.1 Chosen Models

In this study, a choice of four models was made based on their performance metrics. As previously mentioned, choosing models with diverse characteristics helps determine which one is best suited for the application of this thesis. The selected models are:

- The high accuracy and slow model: FRCNN with ResNet50 backbone.
- The fast and low accuracy model: FRCNN with MobileNet backbone
- The balanced models: YOLO models Yolo11s and Yolo11l

Table 5 presents the expected results for these four models, using data from tables 1 and 2. For better visualization, positive attributes (High Frame rate and High precision) are colored in green while detrimental behaviors (Low Frame rate and Low Precision) are colored in red. It is important to note that, since tables 1 and 2 were taken from different sources, these models were tested in different conditions so these results may vary from our experimental results.

Models	Frames Per Second (FPS)	Precision
FRCNN - ResNet50	Low	High
YOLO11l	Medium	Very High
YOLO11s	High	High
FRCNN - MobileNet	Very High	Low

Table 5. Performance comparison of models based on FPS and precision.

3.2 Dataset

For training and testing the selected models, a dataset has been created with images from the Lunar Analogue environment at the Observatory of Tartu. The bunker, depicted in figure 3, has a 7.5m x 8.5m sandbox operational area designed to simulate realistic lunar surface conditions [38].



Figure 3. Space Bunker

The dataset consists of 314 images collected by the KuupKulgur rover at the Lunar analog facility of Tartu Observatory, all captured with an IMX219-160 camera that has a resolution of 720x1280 pixels and a 160° Field Of View (FOV) [39]. All the images in this dataset were annotated using the Roboflow Annotate application [40]. Figure 4 depicts an image from the dataset while figure 5 represents the same image with annotated bounding boxes of rocks.



Figure 4. Input Image from the Dataset



Figure 5. Same Image with Annotations

3.3 Chosen Target Device

For this study, a choice between two devices had to be made due to availability: a Jetson Orin Nano or a Jetson Xavier NX. As discussed in the section 2.4, the Jetson Orin Nano is more relevant for our purposes due to its higher AI Performance for comparable power

consumption, meaning better power efficiency. In this study, we are equipped with a Jetson Orin Nano 8 GB, which detailed specs are detailed in table 6

Specification	Details
AI Performance	67 TOPS
GPU	1024-core NVIDIA Ampere architecture GPU with 32 Tensor Cores
CPU	6-core Arm Cortex-A78AE
Memory	8GB LPDDR5
Power	10W

Table 6. Specifications of Jetson Orin Nano

3.4 Pipeline Overview

This section describes the developed pipeline alongside the different metrics chosen to best assess the benchmarked object detection models. Figure 6 depicts the high-level execution of this pipeline.

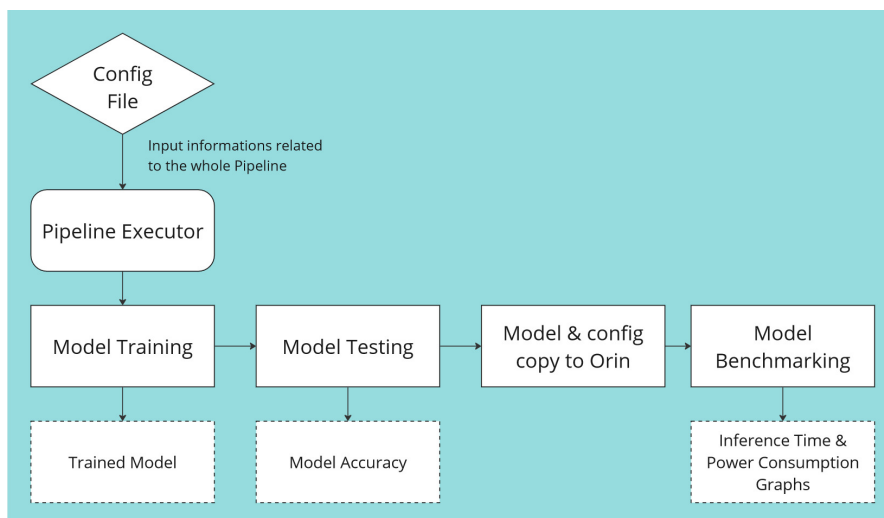


Figure 6. Pipeline Diagram

3.4.1 Configuration File

This pipeline begins with a configuration file in JavaScript Object Notation (JSON) format, where users can input all necessary variables for the pipeline execution. Table 7 presents the list of these variables.

Table 7. Configuration Variables Definition

Variable Name	Definition
train	Toggle the execution of the train module
test	Toggle the execution of the test module
copy	Toggle the execution of the copy module
benchmarking	Toggle the execution of the benchmarking module
framework	Choice of the framework of the pipeline execution
application	Choice of which model will be run
Folders	Input of the different folder locations
conda_env_tf2	Name for the TensorFlow2 conda environment
conda_env_pytorch	Name for the PyTorch conda environment
Training Variables	List of the variables for model training
Benchmarking Variables	List of the variables for model benchmarking
Jetson Variables	IP and Username of the Benchmarking device
Titan Variables	IP and Username of the Training and Testing device

The most important variables in this file are the different module variables, the framework, and the application.

First, the user can choose which module will be run. The different modules will be explained in the next sections. As can be seen in figure 6, if all the modules are selected, first is executed training, then testing, then copy, and finally benchmark. The upside that this pipeline proposes is that it is possible to run fewer modules depending on the interest of the user. For example, the user can do a benchmark on a model that was already trained and tested by this pipeline, without retraining a new model.

Then, the choice of framework has to be made. The user has to choose between TensorFlow 2 or PyTorch framework. These two frameworks are the main ones used for deep learning models, and the choice was made to incorporate them both in the Pipeline. However, during the time of this thesis, the four chosen models were only integrated in PyTorch while only the FRCNN - Resnet50 model was integrated in TensorFlow 2.

The final most important variable is the application, which simply is the choice of which model will be used in this pipeline.

Finally, all the other variables were implemented to have the most modular pipeline possible. For example, by changing the IP and user of the Jetson, it is possible to perform the benchmarking on another Jetson device.

Once all the variables in this configuration file are completed, the pipeline can be launched, and one or more of the following modules will be executed.

3.4.2 Virtual Environments

As previously explained, both TensorFlow and PyTorch frameworks can be used in this Pipeline. Unfortunately, these two frameworks cannot coexist together in the same environments.

To remedy this issue, a common practice is to use isolated environments. The two main ways are to either use Python virtual environment (venv) or conda. In this project, conda was chosen due to its ability to manage complex dependencies easily, a feat that was very advantageous due to the numerous libraries that were used.

3.4.3 Model Training

The first module is model training, where the chosen model is fine-tuned based on the specified parameters in the configuration file. Figures 7 and 8 depict the block diagrams of how the training is made for these two frameworks. At the end of the training process, the resulting model is saved for future use. To know which model was trained the last, the current timestamp is appended to the model filename. Augmentation was implemented with PyTorch images as a way to improve the generalization of the models. It can be noted that due to time constraints, image augmentation was not implemented for TensorFlow models; also these models require square images, which is not mandatory for PyTorch models.

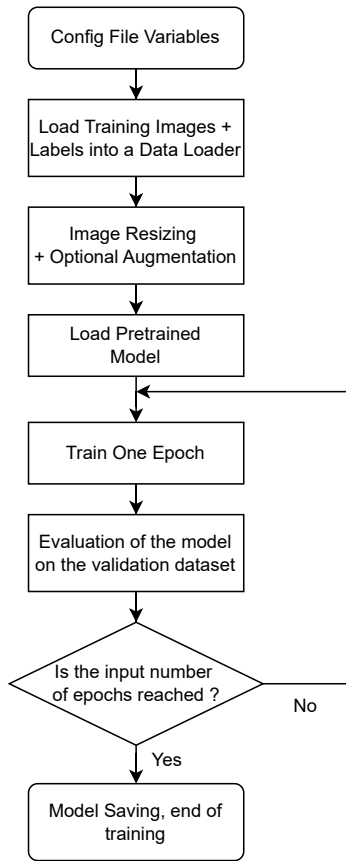


Figure 7. Block Diagram of PyTorch Model Training

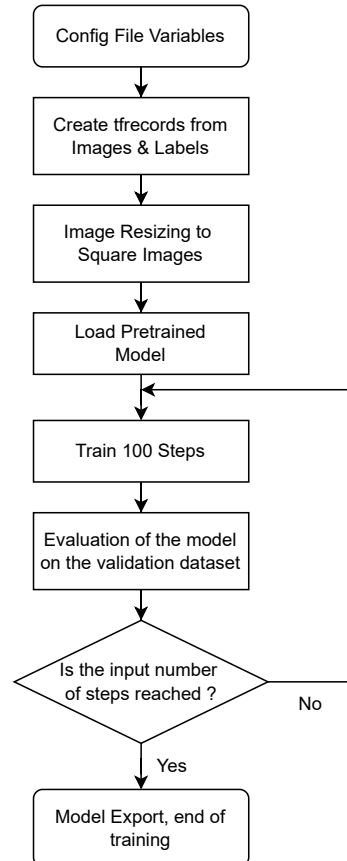


Figure 8. Block Diagram of TensorFlow 2 Model Training

3.4.4 Model Testing

Following the training process, the testing module can be executed. This module evaluates the performance of the fine-tuned model and computes an accuracy score. The chosen accuracy metric is the IoU score. The formula for calculating the IoU score of a model is as follows, with TP representing the True Positives, FP the False Positives and FN the

False Negatives :

$$\text{Model IoU score} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} \quad (1)$$

To determine whether a predicted bounding box is a true positive, a false positive, or a false negative, we calculate its IoU relative to all the ground truth boxes of the predicted image using the formula 2.

$$\text{IoU} = \frac{\text{Intersection Area}}{\text{Union Area}} \quad (2)$$

Then, there are two possibilities :

- If the predicted bounding box has an IoU score of more than 0.5 with any ground truth, it is considered a true positive.
- Otherwise, it is considered a false positive.

At the same time, a dictionary keeps track of which ground truth boxes have already been paired with true positives from the ones that have not. The sum of the unpaired ground truth is considered false negatives.

With all of that, we can finally calculate the IoU score of the tested model using the equation 1. Also, to keep track of the model accuracy, its IoU score is prepended to its filename.

3.4.5 Model & Config Copy

Next, the copy module transfers both the configuration file and the trained model to the Jetson Orin Nano, ensuring that all necessary components are available for benchmarking.

3.4.6 Model Benchmarking

Finally, the benchmark module conducts inference over a designated number of images across multiple iterations, a number defined in the configuration file.

Inference is done differently between TensorFlow and PyTorch frameworks.

For TensorFlow, the model has to be loaded and the according model signature (serving_default, by default) has to be selected to then perform inference. The TensorFlow model is run with XLA runtime optimization, optimization explained in section 2.1.3.

For PyTorch, the model needs to be loaded and, alongside the inferred images, has to be manually sent to the GPU. These models were not run with any runtime optimization

unlike TensorFlow.

During benchmarking, various performance metrics are collected, which are detailed in table 8:

Metric	Unit	Description
Total Inference Time	s	Total time taken from the start to the end of the benchmarking loop.
Individual Inference Times	s	Time taken by the model to predict each image, stored individually.
Total Power Consumption	mW	Overall power consumption of the device.
CPU & GPU Power Consumption	mW	Power consumption of the 6 CPU cores & the GPU
SoC Power Consumption	mW	Power consumption of the core components of the SoC.
GPU Usage	%	Percentage of utilization of the GPU.
CPU Core Usage	%	Percentage of utilization of each CPU core.
CPU Frequency	MHz	Frequency of each CPU core.
RAM Usage	MB	Amount of RAM utilization.

Table 8. Overview of metrics used for benchmarking.

With the total inference time, it is possible to calculate the throughput time. The calculation is detailed in equation 3, it represents the average time taken by the model, including all the necessary operations, such as image loading. Also, by taking the inverse of the throughput time, it allows us to calculate the throughput fps.

$$\text{Throughput Time} = \frac{\text{Total Time}}{\text{Iterations} \times \text{Number of images}} \quad (3)$$

With the inference time for each image, it is possible to calculate the number of frames that can be predicted per second with the equation 4:

$$\text{FPS} = \frac{1}{\text{Mean(Inference Times)}} \quad (4)$$

The power and computational consumption of the model cannot be summarized with only one value. Instead, a graph of the evolution of these metrics is plotted using tegrastats, which reports memory usage and processor usage for Tegra-based devices [41] such as the Jetson Orin Nano. These plots are then transferred back to the main

device that launched the pipeline.

Figure 9 summarizes the steps during model benchmarking.

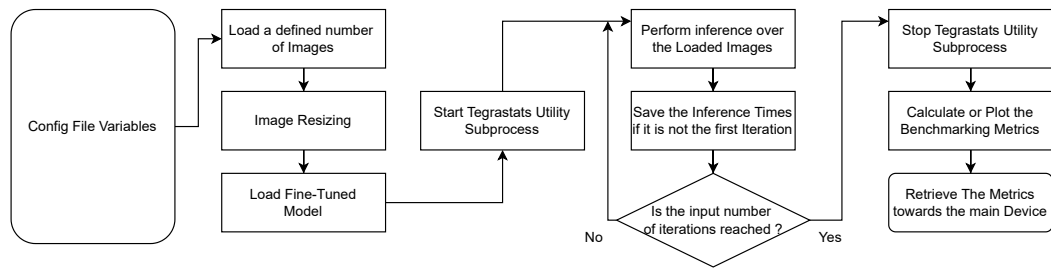


Figure 9. Block Diagram of the Model Benchmarking

4 Results and analysis

This section will present the results obtained using the described pipeline, with the four chosen models under the PyTorch framework: FRCNN with ResNet50 Backbone, FRCNN with MobileNet Backbone, Yolo11l and Yolo11s, as well as FRCNN with ResNet50 Backbone under TensorFlow framework.

4.1 Input Variables

For the PyTorch models, the training and input images have dimensions of 360x640 pixels. These smaller dimensions were chosen to achieve faster inference time while conserving as much information as possible.

For the TensorFlow model, as explained in section 3.4.3, images have a square shape of 512x512 pixels. This ratio is slightly higher in terms of total pixels than the PyTorch images (262144 pixels for TensorFlow and 230400 pixels for PyTorch).

The following results were obtained after 25 training epochs for the PyTorch models, and 4000 epochs for the TensorFlow model, with a batch size of 8, and only the bounding boxes with a confidence score of over 0.5 were kept.

The augmentations for the YOLO models are listed in table 9, each with a 1% probability of affecting each image.

Augmentation	Description
Blur	Creates blur by averaging over a kernel of size between 3x3 to 7x7 pixels.
Median Blur	Similar to Blur but replaces the center pixel with the median of the kernel.
ToGray	Transforms the image into grayscale.
CLAHE	Enhances the contrast of small regions of the images by applying histogram equalization.

Table 9. Image Augmentations for Yolo models

The augmentations of the PyTorch FRCNN models are listed in table 10.

Augmentation	Description
Horizontal Flip	Flips an image horizontally with a probability of 50%.
RandomBrightnessContrast	Randomly adjusts the brightness and contrast of the image with a probability of 20%.
GaussianBlur	Similar to standard blur, it averages pixel values over a kernel, but with a weighted average (center pixels have a bigger weight), with a probability of 10%.
RandomShadow	Creates shadow effects on random parts of the image, with a probability of 10%.
Rotate	Randomly rotates images between -30° and 30° , with a probability of 50%.

Table 10. Image Augmentations for PyTorch FRCNN models

The benchmarking loop made the models perform inference over 50 Images, for 10 iterations.

4.2 Results

4.2.1 Total Time and Individual Times

During benchmarking, it was discovered that the first inference of the first iteration loop was always longer than the following iterations. For clarity, the inference times, excluding the first-pass time, are visible in figure 10. The first inference times were compiled in table 11.

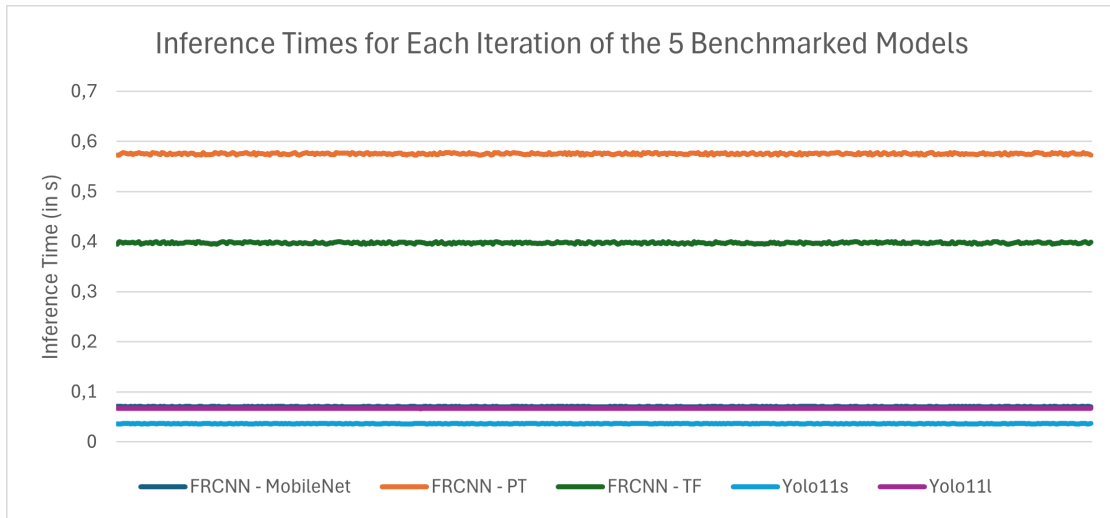


Figure 10. Inference Times for Each Iteration of the 5 Benchmarked Models

Model	First Pass Time (s)
FRCNN - MobileNet	1.60
FRCNN - PT	2.32
Yolo11s	3.98
Yolo11l	4.28
FRCNN - TF	15.19

Table 11. First Pass Times for Object Detection Models

As can be seen in the above figures, the first iteration is always longer than all the next ones. It is because during the first pass, all the elements linked to the model being run such as cache, memory on the GPU, and graph optimization are being set up.

It is interesting to note that the setup time is similar between the two PyTorch FRCNNs and the two Yolo models. It supports the idea that the same kind of model setup is made for similar framework architectures as FRCNNs and Yolo come from two different sources (PyTorch main documentation and Ultralytics).

For the TensorFlow model, more optimizations are occurring such as XLA runtime optimization, which leads to a longer overhead but aims for faster inference afterwards.

4.2.2 Inference Speed and Model Accuracy

First, let us study the first interesting metrics: Inference speed (denoted by the number of Frames per second inferred) and Model accuracy (with the IoU score defined in section 3.4.4). The results can be seen in table 12 and are interesting due to their differences compared to the theoretical values of table 5. From now on, the FRCNN - ResNet50 model run with PyTorch framework will be addressed as FRCNN - PT while this same model with TensorFlow framework will be called FRCNN - TF.

Models	FPS	IoU Score
Yolov11s	27.564	0.525
Yolov11l	14.907	0.557
FRCNN - Mobilenet	14.202	0.3535
FRCNN - TF	2.529	0.3442
FRCNN - PT	1.745	0.7265

Table 12. Performance Comparison of the 5 Benchmarked Models based on FPS and Model Score.

With this table, a few remarks can be denoted:

- FRCNN - MobileNet model was expected to be the fastest which is not the case. In contrast, the Yolo models demonstrated a much better inference speed than predicted.
- Linked with the previous remark, GFLOPS is not the only metric that is needed to determine models with faster inference time.
- FRCNN - PT model performed significantly better than all the other models in terms of accuracy. Comparatively, the Yolo models performed slightly worse than expected.
- FRCNN - TF had a way lower model score than the FRCNN - PT. This is due to some factors such as the lack of augmentation, the lack of hyperparameter tuning, and the need for padding that might have resulted in information loss. However, its speed of inference is nearly 50% higher. This might partially be due to the XLA optimizer, even though more testing -such as running the Yolo models with this optimization- should be achieved to confirm this hypothesis.

4.2.3 Power Consumption

Now, let us focus on the power consumption of the Jetson Orin Nano's modules during inference. The three modules listed using tegrastats are the Chip, the CPU & GPU, and the SoC. For clearer visualization, only the steady-state power output was considered in figure 11.

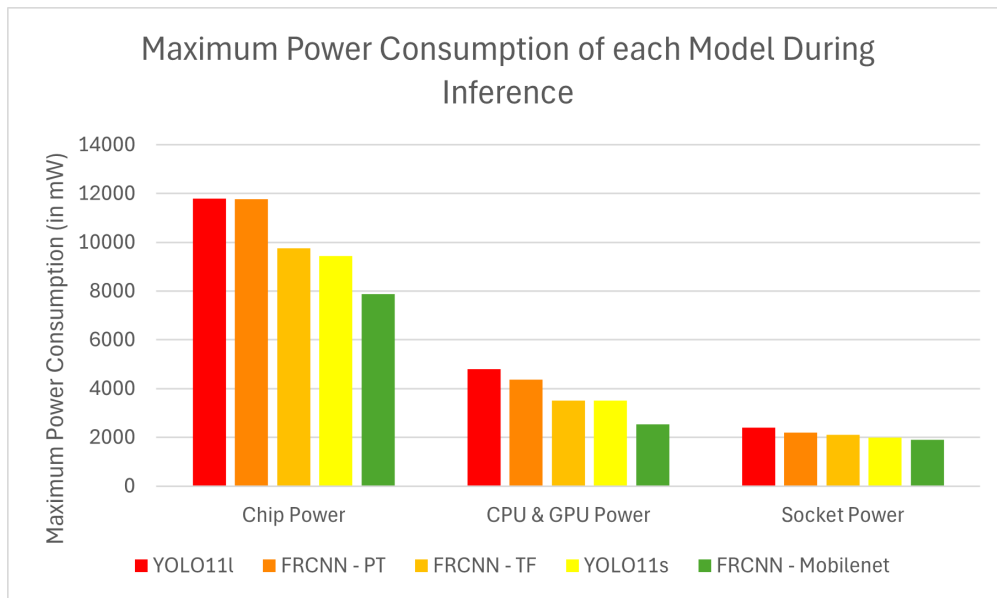
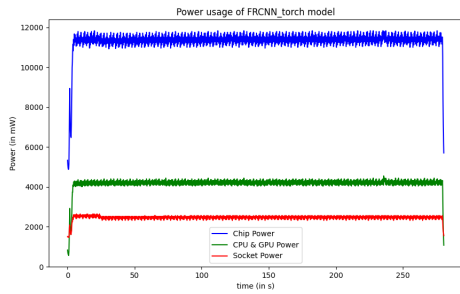


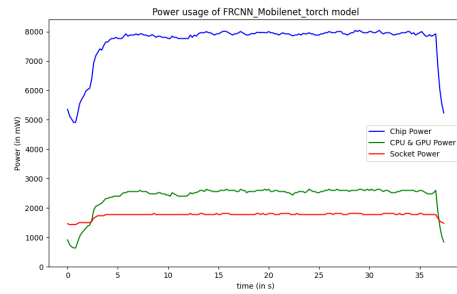
Figure 11. Power Consumption Of the 5 Benchmarked Models

From this table, we can observe that the Yolo11l and the FRCNN - PT require the most power. They are followed by the FRCNN - TF and by Yolo11s models, and finally, FRCNN - MobileNet is the least power-consuming.

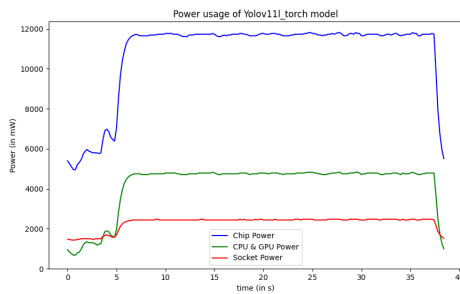
Figure 12 details the power consumption of these 5 models during the whole benchmarking process.



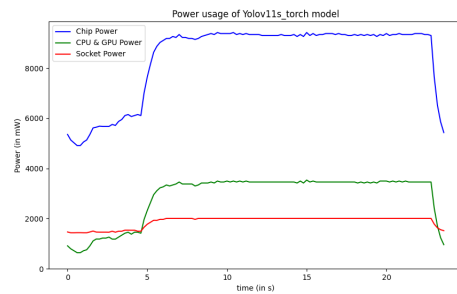
(a) FRCNN - PT



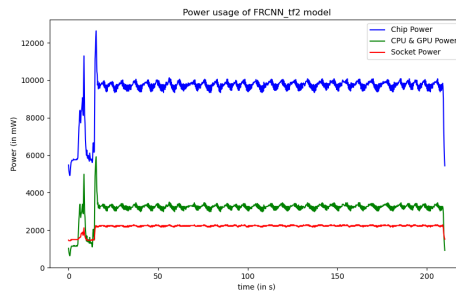
(b) FRCNN - MobileNet



(c) Yolo11l



(d) Yolo11s



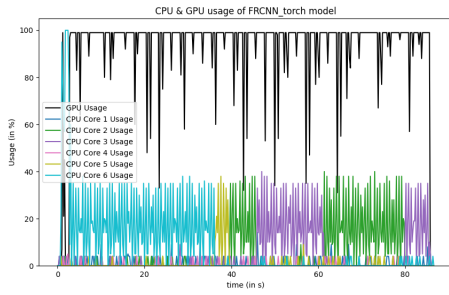
(e) FRCNN - PT

Figure 12. Power Consumption of the 5 Benchmarked Models

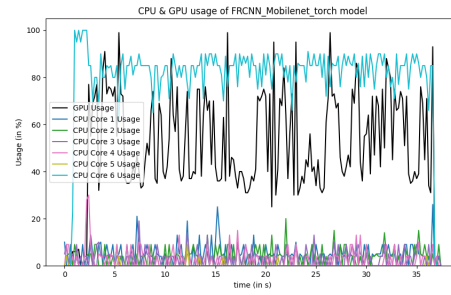
It is interesting to compare these power consumption figures with figure10. We can see that for the duration of the first inference, when the model is being set up and compiled, the GPU and CPU power consumption is pretty low, and then quickly increases to a steady value, before coming back down to a low value after the end of the benchmarking loop.

4.2.4 CPU & GPU

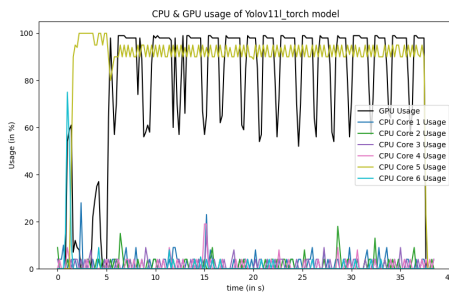
For this part, two graphs per model were created. One is the usage (in %) of the GPU and the 6 CPU cores, visible in figure 13. The other is the frequency (in MHz) of each of the 6 CPU cores in figure 14.



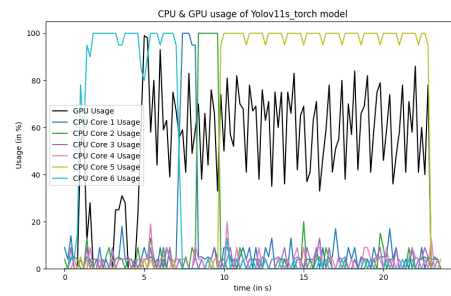
(a) FRCNN - PT



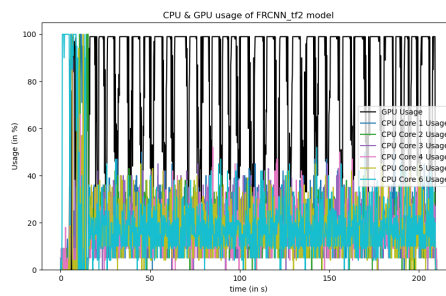
(b) FRCNN - MobileNet



(c) Yolo11l

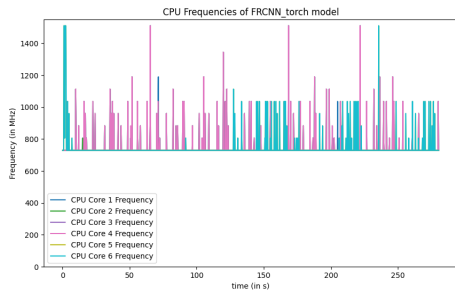


(d) Yolo11s

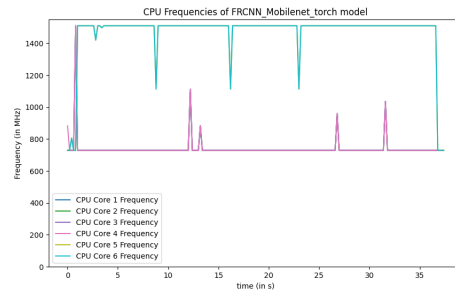


(e) FRCNN - TF

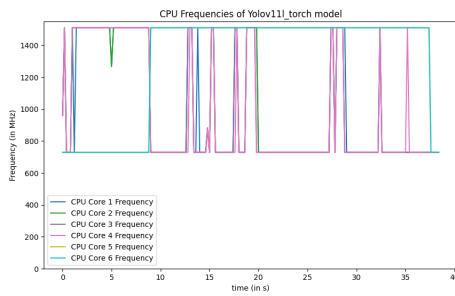
Figure 13. CPU & GPU Usage of the 5 Benchmarked Models



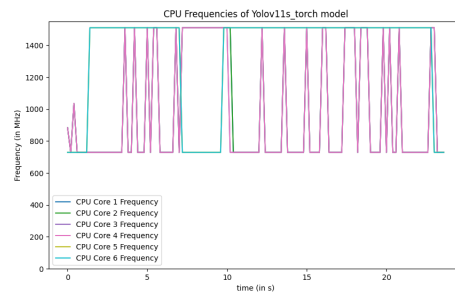
(a) FRCNN - PT



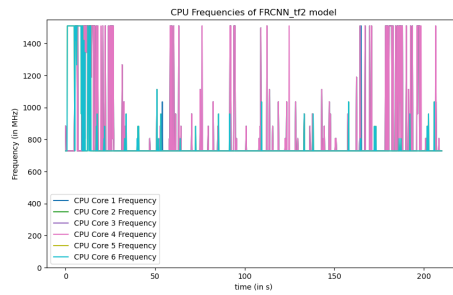
(b) FRCNN - MobileNet



(c) Yolo11l



(d) Yolo11s



(e) FRCNN - TF

Figure 14. CPU Frequency of the 5 Benchmarked Models

To analyze GPU usage we have to look at figure the black line in 13. For the to CPU usage and frequency, we need to study figures 13 and 14, each colored line representing one core.

First, we can again see the effect of the model compilation before the first inference. In these graphs we can observe that model loading is mostly computed by the CPU. Then, as the inference starts, GPU usage rises as it is performing the tensor calculations

required to perform inference.

The two bigger models (FRCNN - PT and FRCNN - TF) are each using the GPU almost at 100% during the whole inference, and the CPU cores usage stays relatively low.

Yolo11l model also uses a great percentage of the GPU while still having a high GPU usage.

Surprisingly, the two smaller models (Yolo11s and FRCNN - MobileNet) do not fully utilize the GPU, while having a fairly high CPU usage. In theory, if the GPU usage was higher, as it is specialized in tensor calculation, it would lead to an even faster inference time. However, it would seem that there are some bottlenecks in the benchmarking process, meaning that images might not be transferred fast enough to the GPU, so it has to remain slightly idle while waiting for images to process.

4.2.5 RAM

Finally, RAM usage was fluctuating during each run of the same model. As this usage relies on multiple exterior factors to the model we are choosing, such as garbage collection or caching mechanisms, it was decided to not incorporate this metric in the final analysis between these models.

4.3 Analysis of the Results

To better analyze all the metrics that were benchmarked using the developed pipeline, table 13 was created, compiling the previously detailed results. For visibility, cells of this table were colored, red colors representing undesirable performances while green represents positive characteristics.

Model	FPS	Model Accuracy	Power Cons.	CPU Usage	GPU Usage
FRCNN - PT	Very Low	Very High	High	Low	High
FRCNN - MobileNet	High	Low	Low	High	Low
YOLO11l	High	Medium	High	High	Medium
YOLO11s	Very High	Medium	Medium	High	Low
FRCNN - TF	Low	Very Low	Medium	Low	High

Table 13. Performance Comparison of Models

5 Discussion

With table 13, we can more clearly decipher which models are the best suited to particular situations.

- FRCNN - PT is a model performance made when the only focus is model accuracy, with no regard toward inference speed or power consumption.
- FRCNN - MobileNet has an edge in cases where the requirement is the lowest power consumption possible.
- Yolo11l has a good balance in most of the domains but is overall worse in inference speed and power consumption for a low accuracy increase over Yolo11s
- Yolo11s balances well all the collected metrics, with a very high inference speed which makes it a perfect choice for our goal.
- FRCNN - TF does not have any use case in its current form, further development would lead to pretty similar results as FRCNN - PT with a possibility of faster inference time.

While comparing these four models, Yolo11s appears to be a great compromise and suits well our problem of balancing model accuracy with inference speed and power consumption.

With this pipeline, we can extensively study different models, with a complete list of their metrics. It was discovered that the experimental results were different than the theoretical ones, so it highlighted the importance of having a unified benchmarking process. From the presented models, one model was highlighted as the best balance for an autonomous driving neural network, the Yolo11s.

This analysis shows that this pipeline did allow us to effectively train, test, and benchmark 5 models, and make a choice on which model is the most adapted to a certain situation, which was the goal of this whole study.

6 Conclusion

In conclusion, the goal of this study was to develop a pipeline designed for training testing and benchmarking object detection models for autonomous navigation. The primary focus was to make this pipeline as easily adaptable and modifiable as possible.

The pipeline modality goal was well achieved, as it is possible to quickly add new pre-trained models, such as the one listed in tables 1 and 2, and new modules such as the implementation of TensorRT optimization or a pruning/quantizing module can easily be implemented in a limited time.

Currently, the TensorFlow side of the Pipeline is underdeveloped compared to the PyTorch one, for future directions it will be interesting to develop it further. To have the highest model accuracy, a dataset containing more images would be needed, however, image labeling is time-consuming so the dataset was kept to a minimal sufficient size for accurate training. Finally, model optimizations that were discussed in section 2.1.3 should be integrated into this pipeline due to their benefits of improving model performances.

With the work done during this thesis, the developed pipeline is at a great working and stable first iteration. It still is not perfect, so by adding the mentioned improvements, this pipeline would become an even more efficient tool to train, test, and benchmark object detection models.

Acknowledgments

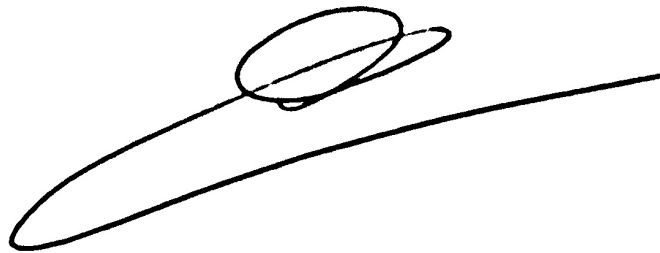
First, I want to express my heartfelt gratitude toward my two supervisors Ric and Saimoon. Your help during my projects with the KuupKulgur team has been more than valuable. The discussions I had with you always helped me navigate the encountered problems of this thesis, and your feedback encouraged me to improve my work and expand my knowledge.

I am thankful for the resources provided by the Observatory of Tartu, allowing such an interesting project as KuupKulgur to be able to exist.

A special thanks to Dmytro Fishman and the lecturers of the Machine Learning and Deep Learning for Computer Vision courses. These classes were not only interactive but also made me learn and grow my knowledge in the field of AI. Your teaching was the reason why I decided to pursue this field in my thesis.

I would like to mention all of my friends back in France, as well as all the new ones I made here in Tartu. To Julian, Alexandre, Antoine, Loïc, Daniel, and all the others that I cannot cite individually, thank you for being part of my life, I am proud of having you as friends.

Finally, I want to warmly thank my parents, for your continuous support and encouragement throughout my entire studies. Without you, I would not be where I currently am. Your love and guidance shaped me and made me able to embark on a journey such as this thesis.

A handwritten signature in black ink, consisting of a long horizontal stroke that curves upwards at the end, with a smaller, more complex loop above it.

References

- [1] Pytorch, “Models and pre-trained weights.” <https://pytorch.org/vision/main/models.html#models-and-pre-trained-weights>. Accessed: 2024-11-19.
- [2] Ultralytics, “Yolov11 models documentation.” <https://docs.ultralytics.com/models/yolo11/>. Accessed: 2024-12-06.
- [3] NVIDIA, “Jetson modules.” Accessed: 2024-12-17.
- [4] P. Marwedel, *Embedded system design: embedded systems foundations of cyber-physical systems, and the internet of things*. Springer Nature, 2021.
- [5] A. Startups, “Jetson agx orin: The hot new thing in edge computing & ai,” Apr. 2022. Accessed: 2024-12-09.
- [6] J. Flahaut, C. H. van der Bogert, I. A. Crawford, and S. Vincent-Bonnieu, “Scientific perspectives on lunar exploration in europe,” *npj Microgravity*, vol. 9, no. 1, p. 50, 2023.
- [7] S. Li, P. G. Lucey, R. E. Milliken, P. O. Hayne, E. Fisher, J.-P. Williams, D. M. Hurley, and R. C. Elphic, “Direct evidence of surface exposed water ice in the lunar polar regions,” *Proceedings of the National Academy of Sciences*, vol. 115, no. 36, pp. 8907–8912, 2018.
- [8] J. Mehta, “Why explore our moon and how we’re going back like never before.” <https://jatan.space/why-explore-the-moon/>, Dec. 2023. Accessed: 2024-12-04.
- [9] J. Lai, Y. Xu, R. Bugiolacchi, X. Meng, L. Xiao, M. Xie, B. Liu, K. Di, X. Zhang, B. Zhou, *et al.*, “First look by the yutu-2 rover at the deep subsurface structure at the lunar farside,” *Nature communications*, vol. 11, no. 1, p. 3426, 2020.
- [10] C. Wong, E. Yang, X.-T. Yan, and D. Gu, “Adaptive and intelligent navigation of autonomous planetary rovers—a survey,” in *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pp. 237–244, IEEE, 2017.
- [11] D. Arce, J. Solano, and C. Beltrán, “A comparison study between traditional and deep-reinforcement-learning-based algorithms for indoor autonomous navigation in dynamic scenarios,” *Sensors*, vol. 23, no. 24, p. 9672, 2023.
- [12] H. Qin, S. Shao, T. Wang, X. Yu, Y. Jiang, and Z. Cao, “Review of autonomous path planning algorithms for mobile robots,” *Drones*, vol. 7, no. 3, p. 211, 2023.

- [13] Y. Wang, C. Gong, J. Gong, and P. Jia, “Motion planning for off-road autonomous driving based on human-like cognition and weight adaptation,” *Journal of Field Robotics*, 2024.
- [14] Built In, “Disadvantages of neural networks,” 2024. Accessed: 2024-11-12.
- [15] W. Powell, M. Campola, and T. Sheets, “Commercial off-the-shelf gpu qualification for space applications,” technical report, NASA, 2018.
- [16] R. L. Davidson and C. P. Bridges, “Error resilient gpu accelerated image processing for space applications,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 9, pp. 1990–2003, 2018.
- [17] K. AI, “How object recognition powers autonomous vehicles.” <https://keylabs.ai/blog/how-object-recognition-powers-autonomous-vehicles/>, 2023. Accessed: 2024-12-04.
- [18] J. Carsten, A. Rankin, D. Ferguson, and A. Stentz, “Global path planning on board the mars exploration rovers,” in *2007 IEEE Aerospace Conference*, pp. 1–11, 2007.
- [19] V. Verma, M. W. Maimone, D. M. Gaines, R. Francis, T. A. Estlin, S. R. Kuhn, G. R. Rabideau, S. A. Chien, M. M. McHenry, E. J. Graser, A. L. Rankin, and E. R. Thiel, “Autonomous robotics is driving perseverance rover’s progress on mars,” *Science Robotics*, vol. 8, no. 80, p. eadi3099, 2023.
- [20] B. Rothrock, R. Kennedy, C. Cunningham, J. Papon, M. Heverly, and M. Ono, “Spoc: Deep learning-based terrain classification for mars rover missions,” in *AIAA SPACE 2016*, p. 5539, 2016.
- [21] X. Wang, Y. Sun, Y. Xie, J. Bin, and J. Xiao, “Deep reinforcement learning-aided autonomous navigation with landmark generators,” *Frontiers in Neurorobotics*, vol. 17, p. 1200214, 2023.
- [22] H. Cai, J. Lin, Y. Lin, Z. Liu, H. Tang, H. Wang, L. Zhu, and S. Han, “Enable deep learning on mobile devices: Methods, systems, and applications,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 27, Mar. 2022.
- [23] G. Menghani, “Efficient deep learning: A survey on making deep learning models smaller, faster, and better,” *ACM Comput. Surv.*, vol. 55, Mar. 2023.
- [24] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, *et al.*, “Searching for mobilenetv3,” in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1314–1324, 2019.

- [25] Y. Zhou and K. Yang, “Exploring tensorrt to improve real-time inference for deep learning,” in *2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pp. 2011–2018, 2022.
- [26] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, L. Gan, G. Yang, and D. Qian, “The deep learning compiler: A comprehensive survey,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 708–727, 2020.
- [27] X. Ying, “An overview of overfitting and its solutions,” *Journal of Physics: Conference Series*, vol. 1168, p. 022022, feb 2019.
- [28] J. Liu, S. Liu, Y. Shao, X. Wan, and H. Zhao, “Mars terrain semantic segmentation using zhurong rover imagery based on transfer learning of historical mission data,” in *2022 International Conference on Service Robotics (ICoSR)*, pp. 139–144, IEEE, 2022.
- [29] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, “Sensor and sensor fusion technology in autonomous vehicles: A review,” *Sensors*, vol. 21, no. 6, p. 2140, 2021.
- [30] J. Shohag, “Automated lunar surface image classification using deep convolutional neural networks for geological feature detection,” *American Journal of Neural Networks and Applications*, 2024.
- [31] J. K. Eshraghian, M. Ward, E. O. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Benamoun, D. S. Jeong, and W. D. Lu, “Training spiking neural networks using lessons from deep learning,” *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023.
- [32] C.-K. Lin, A. Wild, G. N. Chinya, Y. Cao, M. Davies, D. M. Lavery, and H. Wang, “Programming spiking neural networks on intel’s loihi,” *Computer*, vol. 51, no. 3, pp. 52–61, 2018.
- [33] I. Kuon and J. Rose, “Measuring the gap between fpgas and asics,” in *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, pp. 21–30, 2006.
- [34] AMD, “Versal ai edge xa series.” Accessed: 2024-12-20.
- [35] AMD, “Versal ai edge series gen 2.” Accessed: 2024-12-20.
- [36] Google, “Coral dev board.” Accessed: 2024-12-19.

- [37] M. Suryavansh, “Google coral edge tpu board vs nvidia jetson nano dev board — hardware comparison.” <https://towardsdatascience.com/google-coral-edge-tpu-board-vs-nvidia-jetson-nano-dev-board-hardware-comparison>. Apr. 2019. Accessed: 2024-12-09.
- [38] T. Group, “Tartu observatory space missions simulation center.” Accessed: 2024-12-10.
- [39] KuupKulgur, “Payload demonstrator.” Accessed: 2024-12-11.
- [40] Roboflow, “Roboflow annotate.” Accessed: 2024-12-20.
- [41] N. Corporation, “tegrastats utility.” Accessed: 2024-06-12.
- [42] OpenAI, “Chatgpt.” <https://www.openai.com>, 2024. Accessed: December, 2024.

Glossary

Tools

- **Arxiv:** Information source.
- **ChatGPT-4:** [42] Used as a programming and debugging assistant, as a first learning source for new knowledge and LaTeX commands.
- **Conda:** System to create virtual environments and install packages. Particularly important as the TensorFlow and PyTorch environments cannot coexist in the same environment.
- **Draw.io:** Block diagrams designing tool. Used to build figures 7, 8, and 9.
- **Elsevier:** Information source.
- **GitLab:** Project hub. All codes were saved on different branches.
- **GlobalProtect:** VPN. Enables remote access to the Titan Desktop for model training and testing, and to the Jetson Orin Nano for benchmarking.
- **Grammarly:** Writing assistance tool. Grammar internet plugin.
- **IEEE:** Information source.
- **Notions:** Blog tool.
- **ResearchGate:** Information source.
- **Roboflow:** Image Labelling tool for the dataset creation
- **Sci-Hub:** Article access tool. Unlocks part of the pay-to-read papers.
- **Stack Overflow:** Debugging tool. Mostly used after errors or dependency issues.
- **Wikipedia:** Information tool. First approach to new topics.

Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Maxence ROUCHOU**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Development of an Adaptive Pipeline for Object Detection Training and Benchmarking,

supervised by Quazi Saimoon Islam and Ric Dengel.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Maxence ROUCHOU

23/12/2024