ISBN: 978-981-18-5852-9
WCSE 2022 Spring Event: 2022 9th International Conference on Industrial Engineering and Applications
doi: 10.18178/wcse.2022.04.054

# Embedded AI-Box for Visual Detection and Inspection with Mobile Robots in ROS2

Atanas Tanev[1], J. Camilo Vasquez Tieck[1], Georg Heppner[1], Philip Keller[1], Arne Roennau[1], Rüdiger Dillmann[1]

[1] FZI Research Center for Information Technology, Karlsruhe, Germany
tanev@fzi.de, tieck@fzi.de, heppner@fzi.de, keller@fzi.de, roennau@fzi.de, dillmann@fzi.de

**Abstract.** Computer vision has made impressive progress in recent years due to the surge of deep learning (DL). Many DL approaches have been developed to detect, classify and segment many kinds of objects. Most of these models require a large amount of data and computing resources for training. This is specially inconvenient for mobile robots and embedded systems, because they have limited hardware and energy. We present a modular approach for a distributed system and an AI-Box for visual detection of objects and persons. The AI-Box has dedicated hardware for DL, a camera, and it runs ROS2 and Detectron2. The AI-Box can easily be installed onto a mobile robot to extend its capabilities. The trained model is optimized and reduced using pruning and quantization. Finally, the model is executed on the AI-box installed on the robot. We show how the AI-Box can be integrated onto a quadruped robot (Spot). The AI-Box can run ”compressed” DL vision models and provide detection capabilities to the robot. We also show that AI-Box can also be easily extended to run different models for detection, classification and segmentation. By using a system like this, mobile robots can be enhanced with powerful models trained offline for different capabilities.

**Keywords:** embedded AI; mobile robots; vision; object and person detection;

## 1. Introduction

Computer vision plays a crucial part in creating intelligent and autonomous robots. Being able to extract information from a visual source, such as a camera image, and understand how to act upon, is an important building block for complex robotic applications. The robot industry has witnessed great development, especially in the area of mobile robots (as in Fig. 1). There are different types of mobile robots in the market, and they are being used for many tasks, such as inspection, exploration and logistics. It is critical to provide mobile robots with the ability to see and understand the unstructured real-world environment in which they operate. This enables them to autonomously navigate through the environment avoiding col-lisions with obstacles, while searching for objects of interest or completing a task. With such capabilities, mobile robots could find even more practical use and impact the transformation of the industry, and even use them in dangerous and hazardous environments.

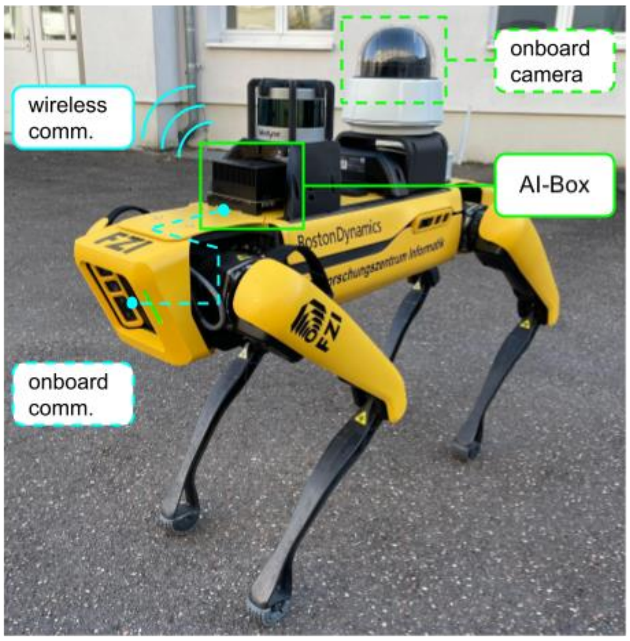


Fig. 1 The AI-Box installed on a mobile robot (Spot). It communicates using ROS2 with the robot using the onboard network, or with an external control station using a wireless network. The AI-Box can have its own camera or use the onboard camera using the ROS2 integration.

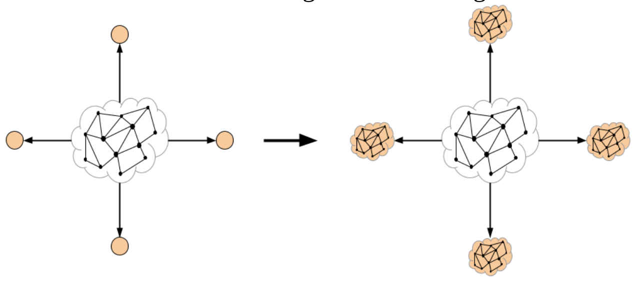


Fig. 2 Distributing centralized, powerful and expensive computing resources for processing AI models, into smaller modular nodes.

Computer vision has made impressive progress in recent years due to the surge of deep learning (DL). Deep learn- ing (DL) has been dominating the space of machine vision with outstanding results in image and video processing for object detection, classification and segmentation. Open-source libraries such as Yolov5 [1], MaskRCNN [2], Detectorn2 [3] offer ready to use pre-trained models. Yolov5 from ultralyt- ics written in Pytorch[4] has different architecture models trained on COCO dataset [5] for object detection. MaskRCNN implemented in Keras [6] and TensorFlow [7] have Feature Pyramid Network (FPN) architecture with a ResNet101 back- bone and offer pre-rained models on COCO [5] dataset for object-detection and in addition generates segmentation masks. Detectron2 [3] developed by Facebook AI as a successor of MaskRCNN is a state-of-the art library for DL algorithms in machine vision. It is implemented in Pytorch and has a wide variety of NN architectures and models pre-trained on different data sets. It can be used for object detection, segmentation, keypoint skeleton tracking and panoptic segmentation. Addi- tionally, it offers compatibility with further research projects such as Denspose [8], Cascade R-CNN [9], PointRend [10], among others.

Most DL models require a large amount of data and computing resources for training. The success of DL has been accompanied by the advances in computing hardware. DL models often require powerful GPUs or TPUs which consume a lot of energy. Nevertheless, this is specially inconvenient for mobile robots and embedded systems, because they have limited hardware and energy resources. This is often solved by transferring the data to an external computing unit which processes the captured data, while resulting in large data traffic and communication latency issues. A promising solution to this problem is to use pre-trained compressed DL models. Using embedded AI algorithms, large DL models can be optimized and deployed on a low-power embedded devices, such as system on chip (SoC) devices with ARM based CPU architecture like the RaspberryPi and Nvidia Jetson boards. With this approach, sensory data is processed locally on the robot, while resulting in shorter latency by significantly reducing the communication bandwidth with the external computing power. There are two common approaches for compressing large DL models and optimizing them for SoC: pruning and quantization [11]. The methods of pruning analyse the weights, and it focuses on suppressing and eliminating weak connections to reduce the size of the neural network. Pruning [12] can be made in a structured and unstructured way. The methods for quantization simplify the NN models by transforming the weight values from float type to integer. Quantization [13] can be made on a trained model known as static quantization, or during the training phase known as dynamic quantization. This results in model reduction but could compromise the accuracy. Embedded AI techniques could be applied to compress powerful DL models and provide mobile robots with advanced vision capabilities.

We present a modular approach for a distributed system and an AI-Box for visual detection of objects and persons (see Fig. 1). Because of this large flexibility, we use Detectron2 in our work. The AI-Box has dedicated hardware for DL, a camera, and it runs ROS2. We use low-power embedded hardware to deploy and execute the trained NN models, which makes it possible to be deployed on any mobile robot. We have developed an approach for deploying, optimizing and using open-source Detectron2 framework for machine vision tasks, together with ROS2. As a communications framework, ROS2 [14] ensures safe and secure transfer of information onboard with the robot, and wireless to a control station. The AI-Box can easily be installed onto a mobile robot to extend its capabilities. It can be integrated as an add-on module, or integrated using the robot cameras and power source The learning takes place in a central powerful computer. Then the trained model is optimized and reduced using pruning and quantization. Finally, the model is executed on the AI-box installed on the robot.

In the following section, we describe our approach for de- ploying Detectron2 functionalities together with ROS2. Then we evaluate the performance on different embedded devices, while taking the different optimization steps into account. At the end, based on our conclusions, we share our ideas for further development.

## 2. Approach

The main goal of our approach is to increase the comfort of practical use of mobile robots in various vision related applications by making them more autonomous.

Robots depend on visual understanding of the environment they interact with. For that purpose DL algorithms with artificial neural networks are used in processing and detecting abstract representation from visual information such as camera image. This come at the cost of using powerful computing units such as GPUs that require large energy resources. Since mobile robots have an energy resource limited by their battery

capacity, implementing such hardware units could significantly reduce the robot's operating cycle under one charge. A so- lution for this is to use external computing resources and bidirectionally transfer large amounts of data. This approach is possible under the assumptions that a stable and strong wireless connection is ensured, otherwise it could lead to large latencies and even in not operational mode. In this way, the autonomy of the robot can significantly be compromised.

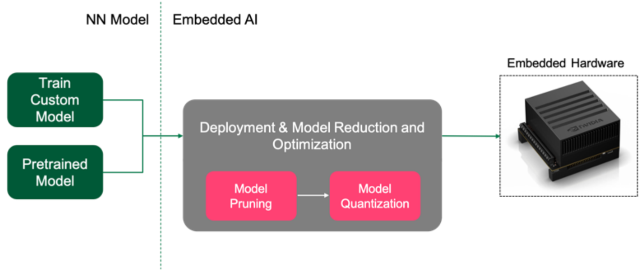


Fig. 3 The three stages of our approach. At the beginning, a model is either trained on a custom data set, or a pre-trained model from Detectron2 library is being used. Following that, the model is optimized using algorithms such as pruning and quantization. At the end, the model is deployed in a ROS2 environment with a custom developed wrapper on a low power embedded device. Nvidia Xavier AGX image was taken from [15].

In our paper we are to address this issues by focusing on the following aspects:

- Decentralization of NN model computing (see Fig. 2) with the use of Embedded AI algorithms, where robots using low-power embedded hardware can locally run the computationally expensive DL algorithms for image processing, while avoiding strong dependence on external communication and computing resources.
- Flexibility in implementation and simple upgradeability of NN models with Detectron2, where the framework ensures compatibility with a large NN model library for various computer vision use cases.
- Modular design and secure communication protocol with ROS2, which as a communication framework ensures encrypted information transfer and compatibility among different hardware units, such as different robots, cameras and embedded hardware.

As depicted in Fig. 3, we split our approach in three separate stages: model-creation, -optimization and -deployment. In the model-creation stage, there are two options for the user: firstly to train a custom model with user-specific classes and labelled dataset, or secondly to use a pre-trained model. Both of these options are accomplished within Detectron2 framework. The user can easily choose from a large library with various DL-architectures developed and validated by Detectron2 and train its own model. Due to the high de- gree of transfer learning and the efficiency of these DL- architectures, the training is much faster [3] in comparison to
other computer vision frameworks. Furthermore, the user can simply choose a ready to use model, which has for example been trained on: COCO[5], Pascal-Voc, LVIS [16] and more. These models can be used for solving tasks such as object detection, segmentation, human keypoint skeleton tracking, panoptic segmentation, etc. In addition, the user can choose from many research projects, such as Densepose[8], Cascade R-CNN [9], PointRend [10], etc. The Detectron2 API ensures the model compatibility and allows the user to even go deeper and modify the functionalities to the specific use case, but still keep the simple efficiency with high-level usage. Furthermore, as an open-source library, it has a large developer community for support.

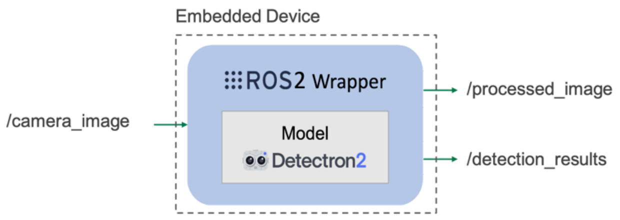


Fig. 4 An illustration of the Detectron2 NN model nested in the custom developed ROS2 wrapper. The model receives an image by subscribing to the topic of the camera /camera_image. The captured image is processed by the model and the results published on two topics. One is /processed_image containing an image with the visual representations of bounding-boxes, segmentation masks, detected class names. The second one /detection_results is a custom ROS2 message containing the intrinsic information of the output, which can later be used and processed for a specific use case. Detectron2 logo was taken from [3].

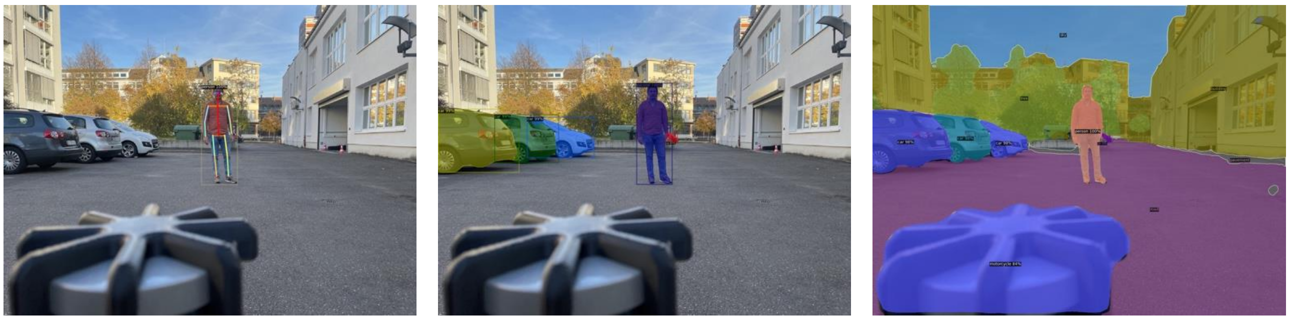
After the model-creation as illustrated in Fig. 3 in the following stage the model is being optimized and prepared for deployment on a low-power embedded hardware. We have developed an automated pipeline using PyTorch that compresses the trained model with the following optimization approaches:

- **Pruning** in unstructured way, by suppressing the weak connections between the neurons and setting them to zero. In this way, less computation is required for the connections that don’t influence the model’s output.
- **Quantization**, as static approach, by converting the weights of the trained model from float into int8. This is supported for both ARM and x86 based CPU architectures. It results in significant model reduction, but at the cost of the model’s confidence.

At the end, the optimized model is deployed within the ROS2 environment. For that purpose, we have developed a ROS2 wrapper that translates the model’s output from Detectron2 into ROS2 compatible format. As illustrated in Fig. 4, the wrapper relies on three main ROS2-Topics. Firstly, the wrapper subscribes to an image topic /camera_image, which contains the visual information to be processed by the model. As next, the image is propagated through the neural network, where the output information is being split into two ROS2-Topics. One topic called /processed_image con- tains the processed image with visualization such as bounding boxes, skeleton, segmentation masks, class-names, confidences etc. The second topic called /detection_results holds intrinsic information about the output as a custom ROS2- Message. This message has the following structure:

- **BoundingBox**, as a list of all detected objects in the frame, with the position and the site of the box in the image.
- **ClassID**, as a list of all detected classes, their names and confidence.
- **SegmentationMask**, as list of image ros-messages, and represents the masks of the segmented objects.
- **Keypoints**, as a list of all detected body parts per detected person, with their position in the image.

In this way, the user can easily select and further process the output depending on the desired use case. Furthermore, using ROS2 as communication framework, enables the user to use the wrapper in a modular way. As an example, any ROS2 compatible camera can be used as a visual sensory input, either directly connected with embedded device, or the robot's board PC. It can also be implemented in different mobile robot platforms, as ROS2 ensures the communication compatibility. In addition to the modularity, it also provides fully encrypted communication through the DDS layer of ROS2 [14].



(a) Human skeleton keypoints (b) Object detection and segmentation (c) Panoptic segmentation

Fig. 5 Visualization of the information from /processed_image topic. Three different use case models from Detectron2 were deployed within the ROS2 wrapper for the on-field experiments with Spot.

It can also be implemented in different mobile robot platforms, as ROS2 ensures the communication compatibility. In addition to the modularity, it also provides fully encrypted communication through the DDS layer of ROS2 [14]. In this way, no external influence can manipulate the information from the camera and the embedded device, which is crucial for the safety of the robot and the environment.

## 3. Results and Experiments

Our experimental setup was constructed in a way to be able to evaluate our approach regrading different hardware and software configurations. We have implemented our developed pipeline on two embedded hardware devices:

- **RaspberryPi4** with 8GB of RAM and Quad-Core CPU with Cortex-A72 (ARM v8)-64 Bit, which was running on Ubuntu Server 20.04. This device offers no GPU unit and also no CUDA compatibility. It was installed with the latest long term support (LTS) version of ROS2 Galactic.
- **Nvidia Xavier AGX** which has 512-core Volta GPU with Tensor Cores, and 8-core ARM v8.2 64-bit CPU
  with 32GB RAM. This device was running on JetPack 4.6 with underlining Ubuntu 18.04. Because of this, we had to install an older and only compatible version of ROS2 Eloquent, which is without LTS and sometimes with unstable behaviour.

Furthermore, we have used RGB usb-cameras running on the ROS2 usb-cam-driver [17], compatible with ROS2 Galactic and Eloquent. As a mobile robot platform, we have used Spot from BostonDynamics (see Fig. 1), which was equipped with a tilt-pan camera from offering a 360 degrees field of view of the surrounding. The camera was directly connected with the embedded device and running and publishing image topics over ROS2 (see Fig. 5).

At first, we evaluated different NN model architectures from the Detectron2 model-zoo [3], regarding their backbone, head and training schedule. By analyzing the processing speed and reading the output frame rate of the deployed NN model on the Nvidia Xavier AGX board. For this purpose we compared three different models from:

- **R50-C4 x1**, backbone network with ResNet in 50 layer variations, with a head that is the fourth module from ResNet [18] and trained with the 1x (12 COCO epochs) schedule [3].
- **R50-FPN x3**, backbone network with ResNet in 50 layer variations, with a head of the second half from the 3x (37 COCO epochs) schedule [3].
- **X101-FPN x3**, backbone network with RasNeXt in 101 layer variations, with a head of the second half from feature pyramid network (FPN)[19] and trained with the 3x (37 COCO epochs) schedule [3].

The NN models were deployed without any optimization steps on Nvidia Xavier embedded device. An usb-camera connected directly to the board and running on ROS2 is publishing captured images with an average frame rate of 6 Hz. The NN model with the ROS2 wrapper is subscribing to the input image topic, as illustrated in Fig. 4. When an image is processed by the model, the wrapper publishes a message on the processed-image topic. By reading out the publishing frequency of this output topic, we can evaluate the processing speed of each model individually. In Fig. 6 The performance results of three evaluated models can be found. X101-FPN reaches an average frame rate of 0,254Hz, while R50-C4 is slightly faster with 0,354Hz on average. Almost five times faster is the R50-FPN model with an average of 1,4Hz. In respect to the results from Fig. 6 we continued evalu- ating the best preforming model R50-FPN in regard to the optimization pipeline illustrated in Fig. 3 by deploying it on two different embedded devices.

We considered comparing four scenarios (see Fig. 7):

1. **No optimization**, the pre-trained model from the Detectron2 library was deployed without modifications,
2. **Pruning**, the model was optimized with the unstructured pruning approach,
3. **Quantization**, the model was compressed using the static quantization approach.
4. **Quantization with pruning**, is a combination of the two optimization approaches.

Regarding the hardware performance, Nvidia Xavier AGX performs more than four times faster than the low power RasperryPi4, where the deployed model without any optimiza- tion steps reaches an average frame rate of 0.32Hz on the Pi4. This is mainly as a result of the GPU units running with CUDA, that can accelerate the computation for the Nvidia Xavier AGX.

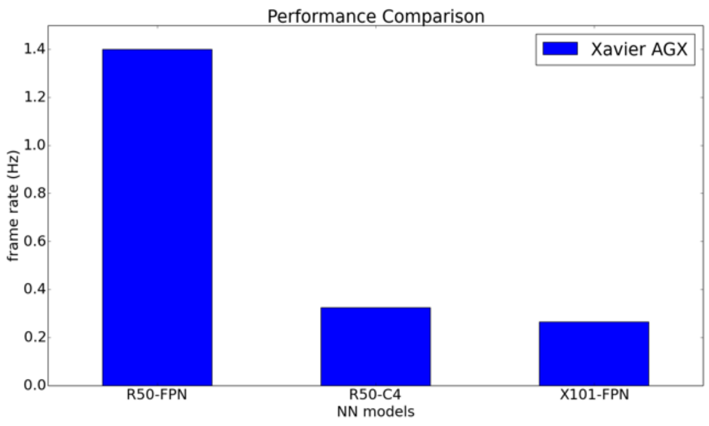


Fig. 6 Evaluation results of different NN model architecture. We compare three different models trained for object-detection and -segmentation, deployed on Nvidia Xavier AGX. We measure the speed of the model by analyzing the publishing frame-rate of the processed-image topic. The results show that the R50-FPN model, with average frame-rate of 1.4Hz, is up to five times faster than the R50-C4 and X101-FPN models.

Furthermore, the results in Fig. 7 show that the applied optimization algorithms to the R50-FPN model have decreased its efficiency. The pruned model on the AGX reduced its effi- ciency up to 50% with an average of 0.65Hz, where on the Pi4 with a minor reduction and an average of 0.29Hz output frame-rate. Similar performance with a slight underperformance was noted for the quantized models on the both embedded device. This optimization step also resulted in a reduced confidence of the detected classes. At the end, pruning and quantitation were combined, which resulted in performance increase for the AGX with an average of 1.1Hz, but the frame-rate of the Pi4 stayed relatively the same.

The results from Fig. 7 imply that the optimization steps, such as unstructured pruning and statical quantitation, don't improve the performance of the Pi4 and especially of the AGX in regard to the R50-FPN model from Detectron2, which could also signify that is sufficiently optimized for its performance.

Nvidia Xavier AGX would probably show significantly improved results and performance, by using TensorRT [20] acceleration framework, developed by Nvidia for their line of hardware devices. Unfortunately, Detectron2 models are still not compatible with TensorRT, which hopefully will change in the next TensorRT version.

We validated our Detectron2 with ROS2 pipeline for ob- ject detection, segmentation, keypoint skeleton tracking and panoseptic segmentation by implementing the Nvidia AGX on Spot-Robot with a tilit-pan RGB camera (see Fig. 1). The results from the on-field experiment are depicted in Fig. 5 and demonstrate three different use cases of the Detctron2 models with the developed ROS2 wrapper such as: human skeleton keypoint detection in Fig. 5a, object detection or segmentation in Fig. 5b, and at the end a panoptic segmentation in Fig. 5c.

## 4. Conclusions and Future Work

We presented a modular approach for a distributed AI system and an AI-Box to extend the capabilities of a mobile robot. The AI-Box has dedicated hardware for DL, a camera, and it runs with ROS2. The trained DL model is compressed and optimized using pruning and quantization. Then deployed on embedded hardware such as in our case a RaspberryPi4 and Nvidia Xavier AGX. We showed how the AI-Box can be installed onto a mobile robot to extend its capabilities. Additionally, it was possible to switch the executed model on the AI-Box to have either skeleton-detection, object-detection, or -segmentation.

The experiments showed that the AI-Box is able to run different vision models using Detectron2. We compared the same configuration between R50-FPN, R50-C4 and X101- FPN models. These models have different backbone and head structure. Based on the benchmarks we performed, the R50- FPN model provides the best frame-rate performance for this application.

Additionally, we compared further optimized versions of R50-FPN. Our conclusion is that the models available are already very good and optimized for Detectron2. In this sense, we learned to not waste time further improving or optimizing them. The quantization and pruning are very useful for the models that are custom-made. Detectron2 has a big pool of pre-trained models, that can be used to extend the AI-box functionality for other applications.

For future work, we would like to convert the models to real-time tensors with TensorRT and take advantage of the GPU units with CUDA. So far, a direct conversion did not work, using the tool to convert from PyTorch to tensor. An alternative is to convert to an "onnx" model, as Detectron2 has the option, and then convert to real time tensor "rtt". There are also some problems with the jetpack version from Nvidia. It is still for Ubuntu 18.04 and this version is not very stable with ROS2. Hopefully, with the new version for Ubuntu 20.04, we can upgrade jetpack and Xavier together with ROS2. A modular system such as the AI-Box that can extend the capabilities of a mobile robot, and can be upgraded or reconfigured any time, provides another dimension for sensory systems. With this, mobile robots can be reconfigured and extended based on the latest technologies or the requirements of the task.

## 5. Acknowledgment

This work was supported by the Competence Center Karl- sruhe for AI Systems Engineering (CC-KING, https://www.ai- engineering.eu) sponsored by the Ministry of Economic Af- fairs, Labour and Housing Baden-Württemberg.

## 6. References

[1] G. Jocher, A. Stoken, J. Borovec, NanoCode012, ChristopherSTAN, L. Changyu, Laughing, tkianai, A. Hogan, lorenzomammana, yxNONG, AlexWang1900, L. Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, F. Ingham, Frederik, Guilhen, Hatovix, J. Poznanski, J. Fang, L. Yu, changyu98, M. Wang, N. Gupta, O. Akhtar, PetrDvoracek, and P. Rai, “Ultralytics/yolov5: V3.1 - bug fixes and performance improvements,” Zenodo, Oct. 2020.

[2] W. Abdulla, “Mask R-CNN for object detection and instance segmen- tation on Keras and TensorFlow,” 2017.

[3] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, “Detectron2,” 2019.

[4] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An imperative style, high- performance deep learning library,” in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlch ́e-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[5] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Doll ́ar, “Microsoft COCO: Common objects in context,” 2015.

[6] F. Chollet et al., “Keras,” 2015.

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Man ́e, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Vi ́egas, O. Vinyals, P. Warden, M. Wat- tenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015.

[8] R. A. G ̈uler, N. Neverova, and I. Kokkinos, “Densepose: Dense human pose estimation in the wild,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 7297–7306.

[9] Z. Cai and N. Vasconcelos, “Cascade r-cnn: Delving into high quality object detection,” 2017.

[10] A. Kirillov, Y. Wu, K. He, and R. Girshick, “PointRend: Image segmen- tation as rendering,” 2020.

[11] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” arXiv preprint arXiv:1510.00149, 2015.

[12] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, “Pruning and quantization for deep neural network acceleration: A survey,” 2021.

[13] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, “A survey of quantization methods for efficient neural network infer- ence,” 2021.

[14] L. Puck, P. Keller, T. Schnell, C. Plasberg, A. Tanev, G. Heppner, A. Roennau, and R. Dillmann, “Distributed and synchronized setup towards real-time robotic control using ros2 on linux,” in 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE), 2020, pp. 1287–1293.

[15] “Nvidia developer,” https://developer.nvidia.com/embedded/ jetson-agx-xavier-developer-kit.

[16] A. Gupta, P. Doll ́ar, and R. Girshick, “Lvis: A dataset for large vocabulary instance segmentation,” 2019.

[17] A. Klintberg, “ROS2 USB camera node.”

[18] K. He, G. Gkioxari, P. Doll ́ar, and R. Girshick, “Mask r-cnn,” in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2961–2969.

[19] S. Xie, R. Girshick, P. Doll ́ar, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” 2017.

[20] Nvidia, “TensorRT open source software.”