Towards Real-time Object Detection for Safety Analysis in an ML-Enabled System Simulation

RTOD for Safety Analysis in an ML-Enabled System Simulation

Jubril Gbolahan Adigun⍟△♦, Patrick Aschenbrenner¶, Michael Felderer⍟♠⊥

⍟ Department of Computer Science, University of Innsbruck, Austria
Email: {firstname}.{lastname}@uibk.ac.at

¶ Institute for Astro- and Particle Physics, University of Innsbruck, Austria
Email: {firstname}.{lastname}@student.uibk.ac.at

△ Ainnov8 Technologies Ltd, Nigeria
Email: {firstname}@ainnov8.com

♦ Center for Artificial Intelligence (AI) Research Nepal, Nepal
Email: {firstname}.{lastname}@cair-nepal.org

♠ German Aerospace Center (DLR), Germany
Email: {firstname}.{lastname}@dlr.de

⊥ University of Cologne, Germany
Email: {firstname}.{lastname}@uni-koeln.de

Abstract — **Machine learning (ML)-equipped critical systems such as collaborative artificial intelligence systems (CAISs), where humans and intelligent robots work together in a shared space are increasingly being studied and implemented in different domains. The complexities of these systems raise major concerns for safety risks because decisions for controlling the dynamics of the robot during the interaction with humans must be done quickly driving the detection of potential risks in form of collision between a robot and a human operator using information obtained from sensors such as camera or LIDAR. In this work, we explore and compare the performance of two You Only Look Once (YOLO) models - YOLOv3 and YOLOv8 - which rely on convolutional neural networks (CNNs) for real-time object detection in a case study collaborative robot system simulation example. The preliminary results show that both models achieve high accuracy (≥ 98%) and real-time performance albeit requiring a GPU to run at such speed as 40FPS. The results indicate the feasibility of real-time object detection in a CAIS simulation implemented with CoppeliaSim software.**

Keywords - collaborative robot, object detection, simulation, machine learning, risk analysis

I. Introduction

Artificial intelligence (AI), inculcating its fast-growing branch, machine learning (ML) has become a mainstay in many aspects of human life. From its integration in simple non-intrusive systems such as a movie recommendation system to those deployed to reduce human effort, for example, GitHub Copilot[1] which can simplify tedious programming tasks. Moreover, AI models dynamically evolve by learning from large datasets obtained from sensor readings, text corpus as well as continuous interaction with humans. Therefore, their development varies significantly from conventional software systems [1].

This presents a new challenge as software engineers and researchers consider AI engineering as a standalone endeavour from software engineering process. Surely, software engineering is broader and involves tasks such as designing, developing, testing, and maintaining software applications across many application domains e.g., education, finance, healthcare, entertainment among others. Yet, AI engineering, which is a specialism within software engineering, is dedicated to the development of systems equipped with human-like intelligence and that are capable of learning from data and making informed decisions or predictions based on that data [2].

Furthermore, in conventional software systems, defects and malfunctions can and do occur, they are not harmful. However, malfunctions due to the application of ML solutions in safety-critical domains can have devastating impacts on lives, property and systems, like robotic systems that work together with humans in a shared physical space to reach a common goal i.e., collaborative artificial intelligence systems (CAISs) [3]. These systems rely on ML components to process external data, decide on the next action and learn from observations.

Several properties, such as safety, robustness and accuracy are crucial in CAISs [4]. Safety risk assessment in industrial collaborative robots is guided by the ISO/TS 15066 which specifies safety requirements for collaborative industrial robot systems and the work environment. The standard provides four modes of operation to ensure safety within the system: safety-rated monitored stop, hand guiding, speed and separation monitoring, and power and force limiting. The safety risk management of CAISs covered by the ISO 10218-1/ISO

[1] https://github.com/features/copilot

This work was partially supported by the Austrian Science Fund (FWF), under grant I 4701-N

102182/TS 15066 standards [5] aligns with the tenets of Industry 5.0 [6] which aims at a human-centric industry. This can be achieved with CAISs by ensuring that workers are supported by autonomous intelligent machines in a safe manner.

Testing and safety analysis of these complex systems requires new approaches that are both efficient and effective because testing takes up a large chunk of software system development budgets. Simulation testing [7] has been used extensively to this end to conduct quick and less costly analysis of different complex systems since simulators present a simpler and more inexpensive way to test systems.

To address the challenges above, we demonstrate an approach that leverages a simulated CAIS where a human operator works alongside a roof-mounted ML-enabled robotic arm. Since the simulation is within a virtually controlled environment, the experimentation can be done with no risk to human life. Compared to a real-world CAIS system, it is way cheaper, changes and modifications to the system can be added quickly and most importantly, there is no risk for real persons and the physical system. For safety risk management in the simulated system, we employ a safety-rated monitored stop in which the identification of a hazard – in the form of a human hand close to a moving robot arm – initiates an emergency stop (IEC 60204-1 category 2 stop and IEC 61800-5-2 SOS)[2].

The robotic arm is equipped with a vision sensor mounted at its tip for obtaining continuous image streams and detecting potential risks for the human operator and stops the robot, if necessary, via a control script. As a risk measure in the simulation, the distance between the operator's hand and the tip of the robotic arm is used. Specifically, two different YOLO [8] models for object detection methods are applied and both rely on a deep convolutional neural network (CNN) architecture trained from scratch as well as via transfer learning i.e., using a testing a model pre-trained on real life data on data generated from the simulation and compared in terms of speed and prediction accuracy.

CONTRIBUTIONS. Through this paper, we contribute to the existing literature in the safety analysis on ML-enabled systems through the following:

- We highlight the need for real-time object detection as a safety function based on ISO 10218-1/ISO 10218-2/TS 15066 safety risk management standards in collaborative ML-enabled systems.
- We explore the potential of transfer learning of existing object detection models from real to virtual as way to quickly validate the models.
- We present an argument for the use of simulators in safety analysis of ML-enabled systems.
- We carry out an experimental campaign on a simulated industrial CAIS to demonstrate our approach which to the best of our knowledge is a pioneer example in the safety analysis of ML-enabled systems.
- We evaluate the performances of the selected object detection approaches.

The remaining part of the paper is structured as follows: Section II introduces the theoretical background of the applied methods, in Section III, our approach is explained in detail, a brief discussion of the preliminary results ensues in Section IV and finally, in Section V, a summary of this work and an outlook for the future are highlighted

II. BACKGROUND

Making sense of a scene, real or virtual, comes rather naturally to humans and they can decipher objects within an image more easily. Teaching a computer to identify objects usually requires thousands of training samples. With object detection, machines can localize objects within an image. Indeed, a high predictive performance and real-time discernment of objects are necessary for human-like object detection in computers. Accordingly, real-time means that the objects in an image must be detected within the time it takes until another frame surfaces (i.e., consecutive frames of a video stream [9]).

AI researchers have developed different approaches for object detection. Section II-A provides the basis for a popular network in artificial intelligence on which complex systems such as convolutional neural networks (Section II-B), learning and decision-making tasks are built. Section II-C introduces to the usage of (convolutional) neural networks for object detection

A. Neural network

The first proposal of artificial neurons predates the invention of electrical computers and goes back to McCulloch and Pitts [10]. Their neurons have a set of input and control signals, and one output signal. This simple neuron only works with binary data. It is possible to create logical AND, OR and NOT gates, which build a functionally complete set. Therefore, every Boolean function can be built from McCulloch-Pitts neurons.

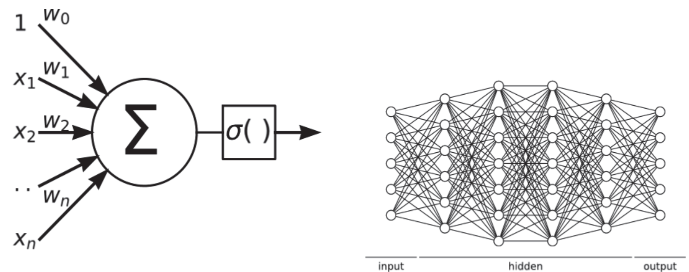

Figure 1. Neuron and fully connected neural network. *Left:* Model of a single neuron. The output of the neuron is the activation of the weighted sum of all inputs $gma(\Sigma_{i=0}^{n} x_i \omega_i)$. *Right:* Example of a fully connected neural network. Each circle represents a neuron. The network has one input layer, four hidden layers and one output layer.

A neural network can be constructed from multiple neurons by arranging them in layers and using the outputs from one layer as the input for the next layer. The first layer is called the input

[2] https://www.controlengeurope.com/article/109959/EN-61800-5-2--more-than-just-Safe-Torque-Off.aspx

layer, the last one output layer and all layers in between are hidden layers. More details about how neural networks work can be found in Werbos [11] and Rumelhart et al. [12].

A simple network is shown in Figure 1. It is called fully connected since every output of a layer is an input of every neuron in the next layer.

The evaluation of a fully connected neural network is done via forward propagation (i.e. evaluate the first layer, use the output to evaluate the second one and so on). The output $\mathbf{o_i}$ of layer i is calculated as follows:

$$\begin{aligned} o_1 &= \sigma_1\left(W^{(1)}x\right) \\ o_2 &= \sigma_2\left(W^{(2)}o_1\right) \\ &\dots \\ o_i &= \sigma_i\left(W^{(i)}o_{i-1}\right) \end{aligned} \tag{1}$$

Note that $W^{(i)}$ is the matrix obtained by stacking all the (row) weight vectors for each neuron ($\mathbf{w}$ in (1)) in the i-th layer and that gma$_i(\cdot)$ is applied element-wise. The activation function for different layers can be different. Typical nonlinear activation functions that are used are:

$$\begin{aligned} &\sigma(x) = \frac{1}{1+e^{-x}} : \text{logistic sigmoid} \\ &tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} : \text{hyperbolic tangent} \\ &\max(0, x) : \text{rectified linear unit (ReLU)} \\ &H(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} : \text{Heaviside step function} \end{aligned} \tag{2}$$

If a linear activation function is used at each layer, for example, the identity function, then one can always find an equivalent neural network without a hidden layer, since all the matrix multiplications can be simplified to one.

B. *Convolutional neural network*

The task of image recognition can also be performed with a fully connected network, where the different pixel values are used as input for the neural network. However, in images nearby pixels are stronger related than distant ones and object features are present in local parts of an image. Therefore, fully connected networks have many unnecessary connections. Furthermore, the exact position of an object within the image is most of the time irrelevant. Building those principles into a neural network was first done by LeCun et al. [13], where they trained a neural network to recognize handwritten digits in zip codes.

To extract the local features in an image, a convolutional layer is used. In difference to a fully connected layer, it is only connected locally in a regular pattern. The result after the convolution is a large feature vector. Since the exact location is often not important, a pooling layer can be used for simplification, where values are aggregated over a neighbourhood, e.g. via max- or mean-pooling. Neural networks which include one or more convolutional layers are called convolutional neural networks (CNNs).

Over the past decade, the accuracy of CNNs has been greatly improved by adding more layers, creating so-called deep CNNs. One of those models, AlexNet by Krizhevsky et al. [14], won the ImageNet Large Scale Visual Recognition Challenge [15] with a big lead over the second place. Since then, classification tasks are dominated by neural networks over the classical approaches.

C. *Object detection with convolutional neural networks*

Different approaches for object detection with CNNs have been developed and improved, each trying to outperform the other. Prominent examples are YOLO (you only look once) [8], SSD (single shot detector) [16] and RetinaNet [17]. Improvements made to YOLO (YOLOv3) [18] led to a huge performance increase, outperforming the other networks [18] as shown in Figure 2. Over the years, many modifications have been added to the YOLO architecture, with the most recent version being YOLOv8 introduced by Jocher et al. [19].

However, many implementations still use YOLOv3 due to its reliability.

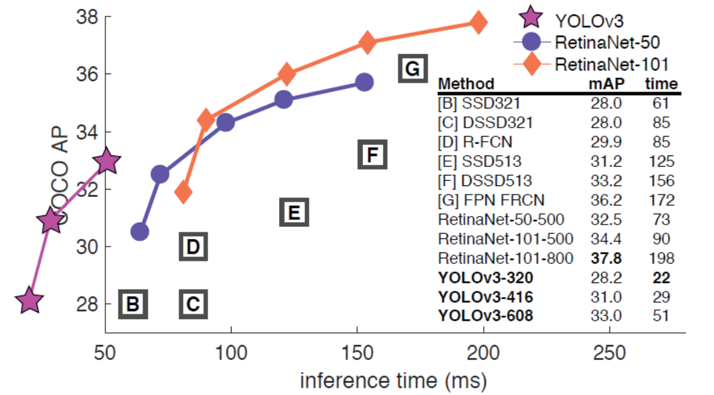

Figure 2. Runtime comparison of different CNN models for object detection. Figure taken from [18].

Object detection is that the latter finds the bounding box of an object within the image. In the case of YOLO, the bounding box annotations are defined in a text file, each line contains 5 numbers (without <>):

$$< label > \quad < c_x > \quad < c_y > \quad < w_x > \quad < w_y > \tag{3}$$

The *label* is an integer starting from 0. It is up to the user to keep track of what real label (e.g. dog, cat, hand, ...) the number corresponds to. Usually, all labels are defined in an extra text file, in the first line is the name of the 0*th* object, in the second line of the 1*st* object and so on. The floating-point numbers c_x and c_y are the bounding box centre coordinates, normalized by the image width and image height respectively. w_x and w_y are floating point numbers of the bounding box width and height, again normalized by the image width and image height respectively.

III. APPROACH

The workflow applied in this work is illustrated in Figure 3. As shown, the tester (domain expert) defines relevant domain features that would be serve as parameters for a test case generator (a random search algorithm). Next, the search algorithm creates a set of randomly generated test data that is fed into the simulator to create different scenarios for which the vision sensor my capture different image frames of the scene. When the simulation is run, the vision sensor captures images of within its point-of-view (POV) and saves them. From the images, a training dataset is created which is used to train a model for object detection. This model is then used to stop the simulation if an emergency is detected.

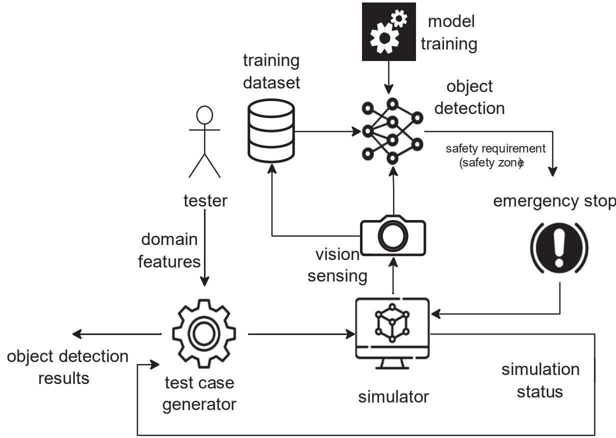

Figure 3. Illustration of the workflow used to implement the real-time object detection campaign in a simulator.

Overall, there are two major parts, the simulator (explained in Section III-B) where the simulation itself runs and the control package, from where the flow of the simulation is controlled. In the latter the object detection is implemented, see Section III-C for details.

A. *Runtime System Specification*

All the programs and libraries used for the implementation are available for Windows and Linux operating systems. Table I shows the system properties and libraries inculcated in the implementation used. NOTE: the libraries may depend on additional requirements.

Table I. System and Simulation Environment Properties

Feature	Description/Component
Operating system	Windows 10 Pro (64bit)
CPU	Intel Core i7-2600, 3.40GHz
GPU	NVIDIA GeForce GTX 970 (4GB)
Simulation software	CoppeliaSim Edu 4.4.0
Control environment	Python 3.9.0; NumPy1.24.2; OpenCV 4.7.0.72; PyTorch 2.0.1+cu118; ImageAI 3.0.3; PyZMQ 25.0.0; Ultralytics 8.0.111

B. *Simulation*

The simulation itself is implemented with the robot simulation software CoppeliaSim [20]. A snapshot of the working example of the simulated system is shown in Figure 4. The CAIS in the simulation consists of two protagonists, the human operator (called Bill) and a robotic arm. Both have the task to grab a green cube that lies on the conveyor belt. The robotic arm has a visual sensor at its tip and takes images with a resolution of 256 × 256 px.

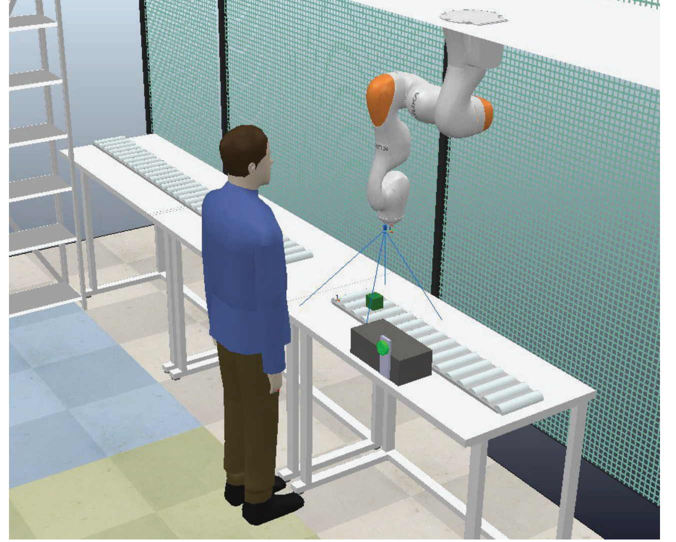

Figure 4. Overview of the simulated CAIS. Bill, the human operator stands next to a table with a conveyor belt. A green cube is placed on top of it. On the ceiling is a robotic arm mounted. At its tip, a vision sensor is placed. The viewing frustum of the vision sensor is outlined by the blue lines.

1) Simulation environment: The simulation environment typically has properties and parameters defined as part of a scene and runs in discrete time steps that may be altered e.g., dt = 50ms. By default, a scene must have a main script (programmed with Lua [21]) and contains the fundamental code that allows a simulation to run through a collection of four callback functions: *sysCall init*, *sysCall actuation*, *sysCall sensing* and *sysCall cleanup*. The main script can interact with a simulation "control" script also directly programmed within the simulation scene as a child script or as an external script. The two main actors in the simulation are Bill and the robotic arm.

Both actors aim to pick and move the cube away from the collaborative space, as they proceed on each timestep. In total, seven (7) free parameters (domain features) in the simulation are randomly assigned values from an automated control script:

1: Actuation delay for Bill
2: Hand movement speed for Bill
3: Actuation delay for the robotic arm
4: Movement speed of the robotic arm
5-7: RGB colour values of the ambient light

2) Control system: The simulation parameters and evaluation of a run are implemented in Python programs. To connect to CoppeliaSim the ZeroMQ[3] remote API is used.

[3] https://zeromq.org/

ZeroMQ is a high-performance asynchronous messaging library which is available for many different programming languages and operating systems. The simulation software opens a TCP port (default 23000) which can then be accessed to get the state of the simulation, set parameters and advance the time steps. Since the communication is TCP based, the control environment does not have to run on the same system as the simulation; an own dedicated machine can be used.

C. Implementation

Two versions of the YOLO object detection models[4] were deployed in our work. They were implemented as part of the control script in Section III-B2 and linked to the simulation from where continuous images are sent from a vision sensor connected to the tip of the robotic arm.

1) Dataset: The first step towards training both methods was to generate two datasets of images. One for training and one for testing. To generate the images, the simulation was run 500 times with random parameters and the images captured by the vision sensor were saved. Then the images were sorted manually into the ones with a hand present and those without a hand. From the total number of available images, 2960 images were selected for the training set and 500 for the test set each with and without a hand respectively.

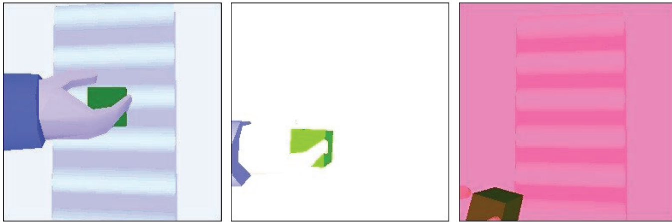

Figure 5. Three example images from the training for different light conditions and hand positions. The images have a resolution of 256 × 256px.

The neural network approach only needs the training samples with a hand, called positive samples. However, those images must be labelled according to the YOLO label format (see (3)). LabelImg [22] was used to label the images since it provides a convenient graphical user interface (GUI) to quickly label a set of images. A rectangular box can be drawn around the object one wants to label, and the label of the object can be selected - in our case, the object of focus is Bill's hand within an image frame.

The labels are saved into a text file with the same name as the image (except for the file extension). During the labelling process, a few misclassified images were noticed and removed from the data set. This resulted in the datasets listed in Table II. Examples of the images are shown in Figure 5. In total, the labelling took roughly 6 hours.

Table II. Number of images in the training dataset and the testing dataset.

	Images with hand	**Images without hand**
Training dataset	2960	2960
Testing dataset	500	500

2) Object detection using YOLOv3: For YOLOv3, the ImageAI [23] Python library was used and relies on PyTorch [24]. The library implements the YOLOv3 architecture and has pre-trained models available. There are different "sizes" of YOLO available with different numbers of weights. In this work, only the tiny-YOLOv3 version was used because GPU resource constraints. For the training process, the dataset is required to follow a strict directory structure where the generated images are named similarly as their annotations as in *imagexxx.jpg* should have a corresponding *imagexxx.txt.*

The names of the parent directory and the images do not matter. However, the annotations must have the same name as the images, except for the file extension. Note that the images for training and validation are both from the training dataset described in Section III-C1, 20% of the images are taken for the validation process and 80% for training. The tiny-YOLOv3 model was trained with an upper limit of 1000 epochs with a batch size of 16.

The YOLOv3 source code was modified to stop model training since there is no method for early stopping (and to prevent overfitting) of the training process in the ImageAI implementation of YOLOv3 if the model performance on the validation set no longer increases for 50 consecutive epochs. While training a neural network from scratch can take quite some time, transfer learning may be used leverage the knowledge of already trained networks by initializing the weights of a new network with the solution of a different network (that solves a similar problem) [25] - in our case, moving from real to virtual. A more recent discussion can be found in the work of Zhuang et al. [26]. To investigate the different behaviours, the networks were both trained from scratch and as well implemented in a transfer learning paradigm.

3) Object detection using YOLOv8: YOLOv8 is the most recent version of the YOLO model available in a Python library from Ultralytics [19]. Its implementation is similar to YOLOv3, as described in the previous section. Two small differences for the training process are, that the directory annotations must be renamed to labels and that the location of the training images must be specified in a separate YAML file [27]. Otherwise, the same training images, batch size and number of training epochs were used. YOLOv8 also comes in different model sizes: nano, small medium, large and extra-large. In this work, the nano and small versions were trained, both from scratch and with transfer learning.

[4] The object detection models have been packaged as a Python library at https://github.com/PatrickAschenbrenner/cais_rtod.

D. Object Detection Integration

The A simple example of how different detection methods is shown in Listing 1.

```
1 from cais_rtod.detector import YOLOv3, YOLOv8
2
3 img_file = "random/image.jpg"
4 # image can also be an in-memory opencv image
5 # img_file = cv2.imread("random/image.jpg")
6
7 # YOLOv3
8 yolov3_detector = YOLOv3()
9 prediction = yolov3_detector.predict(img_file)
10 if prediction == 1:
11         print("Hand detected with YOLOv3")
12
13 # YOLOv8
14 yolov8_detector = YOLOv8()
15 prediction = yolov8_detector.predict(img_file)
16 if prediction == 1:
17          print("Hand detected with YOLOv8")
```

Listing 1. Example of how the different detectors can be used in Python to predict if a hand is present in an image. Note that the detectors should only be called once (lines 8 and 14 in the code above) if multiple images are analysed. The library can also be used to draw bounding boxes around the detected hands, an example can be found in the repository containing the Python library.

IV. PRELIMINARY EVALUATION

In this section, we demonstrate how the different approaches performed with the testing dataset using common evaluation metrics [28], [29] according to the following image classification definitions:

- True positive (TP): Number of images with hand, correctly classified
- True negative (TN): Number of images with no hand, correctly classified
- False positive (FP): Number of images with no hand, incorrectly classified
- False negative (FN): Number of images with hand, incorrectly classified

The values resulting from determining how the different images are classified by the object detection models are then to calculate the metrics in (4).

$$\begin{gathered}\text{True negative rate (TNR): } \frac{TN}{TN+FP} \\ \text{Precision: } \frac{TP}{TN+FP} \\ \text{Recall (True positive rate (TPR)): } \frac{TP}{TP+FN} \\ \text{Accuracy: } \frac{TP+TN}{TP+FP+TN+FN}\end{gathered} \tag{4}$$

The above metrics are useful for classification tasks. However, in object detection, one wants to also know how well the predicted bounding box fits. To compare the true box with the predicted one, the intersection over union (IoU) is used. It is defined as the ratio of the intersection of the two bounding boxes over the area of their union.

A common metric to measure how well bounding boxes are predicted is the mean average precision (mAP, mAP@50 if IoU ≥ 0.5 is chosen as the detection threshold) over all classes. A detailed explanation of how it is calculated can be found in [30]. However, since our focus is on assessing safety risk, it is sufficient that the models identify an arm fully or in part within an image frame to trigger an emergency stop.

A. YOLOv3

The YOLOv3 network was trained from scratch and with transfer learning. For the former, the network was trained from scratch for 2000 epochs without early stopping. This number was chosen as it is significantly higher than the number chosen for early stopping and training could still be completed in a reasonable amount of time. However, training for more epochs decreased the accuracy of the network. The training time for early stopping was below 1h, while the full 2000 epochs took 30h. The evaluation of the full test dataset was about 22.3 ± 0.3s on the GPU and 129.1 ± 0.6s on the CPU. For the latter, the weights were initialized with those from a pre-trained model on the COCO dataset [31] provided by ImageAI[5].

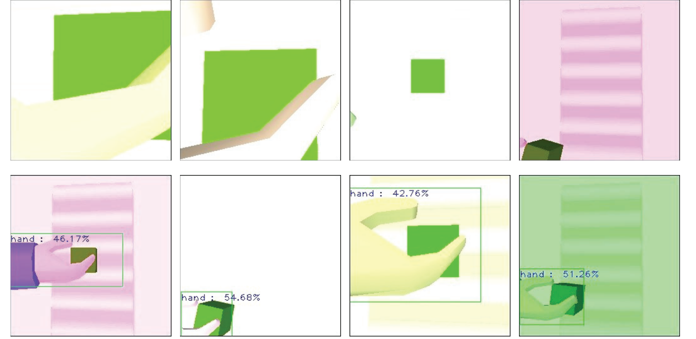

Figure 6. YOLOv3 example images. *Top:* Four false negatives from the test sample. *Bottom:* Selection of correctly identified images.

The results for the different models are listed in Table III. The minimum confidence level for detection is set to 40%. The best performance is obtained from the transfer learning model. From the test set only five images with a hand were misclassified (FN = 5) and none of the images without a hand (FP = 0). Examples of detected hands are shown in Figure 6.

B. YOLOv8

Similar to YOLOv3, the YOLOv8 networks were trained from scratch and with transfer learning, both with early stopping enabled. The nano and small implementations of the YOLOv8 architecture were used to achieve the best framerate. For transfer learning the pre-trained model provided by Ultralytics, which was originally trained on the COCO dataset, was used. The minimum confidence level for detection is set to 40%. Compared to YOLOv3 the training time per epoch is faster. The

[5] https://github.com/OlafenwaMoses/ImageAI/releases/download/3.0.0-pretrained/tiny-yolov3.pt

evaluation of the complete test dataset for nano-YOLOv8 (12.0 ± 0.1s) and small-YOLOv8 (13.0 ± 0.1s) models are approximately twice as fast as that of the tiny-YOLOv3 model as shown in Table III. Expectedly, the nano models have a higher frame rate than the small models. The small pre-trained model achieved the best accuracy, with *FN* = 3 and *FP* = 7. In Figure 7 some example images are shown.

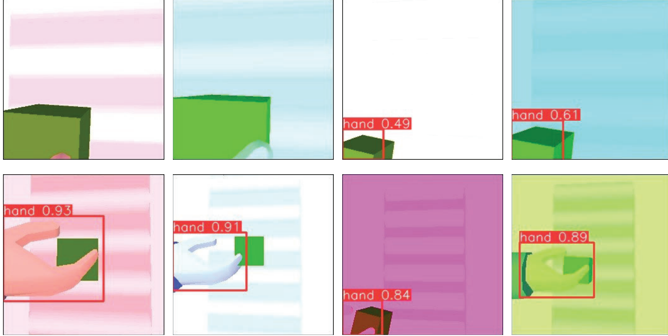

Figure 7. YOLOv8 example images. *Top:* Misclassified images. From left to right are two false negatives and then two false positives *Bottom:* Selection of correctly identified images.

C. *Implication of the Results*

In terms of the actual performances of the models, the results are consistent with findings from a similar work on real-world datasets which compares variants of YOLOv3 and YOLOv5 [32]. Although in our implementation there weren't significant differences in the accuracy, precision and recall of the models, the differences are more noticeable in terms of speed of detection whereby YOLOv8 variants have faster detection rates that YOLOv3.

The results in the previous sections showcase promising potential for proactive identification of potential safety hazards before deploying the CAIS in the real world allowing for adjustments to be made to the system or the environment to mitigate risks. For instance, if the performances of the models were to be low in general or lower than a certain set threshold, additional models can quickly be implemented, or the existing models modified and tested until a desired outcome is obtained.

Additionally, the simulation testing process can be extended for real world systems with minimal cost implications either financially or in terms of human effort needed. For instance, while in this work, we have relied on using transfer learning of object detection models trained on real-world data, for use in a virtual domain, the reverse is also possible i.e., training the models on synthetic data and then transferring them for use in real-world systems with little adjustments made to system.

Furthermore, it is important to bear in mind that simulated environment may not perfectly reflect the real world. Several environmental and system factors like lighting variations, sensor noise, and unexpected object appearances may be difficult to obtain in a simulator. Therefore, discrepancies may occur between simulation results and real-world performance, and these must be accounted for.

V. CONCLUSION & OUTLOOK

In this work, we argue for the need for safety risk analysis in critical ML-enabled systems using simulations. Accordingly, we present two different YOLO models for real-time object detection: YOLOv3 and YOLOv8 which rely on a deep CNN in a CAIS simulation. The models have been composed into an easy-to-use Python library and can be added to an existing simulation control environment without the need to rewrite the existing code. The simulation showcases a collaboration between a human operator and a robotic arm equipped with vision sensor detect a potential risk for the operator (Bill), completing a simple pick and move task.

Both models achieved similar accuracy (≥ 98%) and precision (≥ 96%). However, YOLOv8 achieves higher detection rate of about 83FPS (12.0ms inference time) nearly twice the evaluation speed of YOLOv3 with 44FPS (22.7ms inference time) although both required dedicated GPU for training and testing. For early stopping, both models were trained in approximately 1-2 h. Noticeably, longer training did not result in better performance.

For future use cases, the implementations must be tested for robustness. This work only considered 7 domain features and the models were trained based on the assigned of random values to the various features. Additional parameters and changes in the scenery may alter the look of the captured images to an extent where they differ too much from the training images.

One may also use the size information from the predicted bounding box to estimate the distance between the robotic arm and the hand of the operator so that a more fine-grained risk estimation can be applied in combination with applicable safety assessment standards. For example, a speed and separation monitoring (SSM) based risk management process may be used instead of the safety-rated monitored stop.

Future work would explore if a model trained on synthetic data only can be tested on real-world data by initially training with a large set of simulated data, and afterwards, only a small training sample from the real world is needed. However, additional parameters such as shadows and occlusion which are practical phenomena in the real world must be considered.

ACKNOWLEDGMENT

This study is partially supported by the Austrian Science Fund (FWF) for the project SafeSec, grant agreement No: I 4701 Internationale Projekt.

TABLE III. Summary of the different metrics for the YOLOv3 & YOLOv8 implementation. The models are trained with transfer learning and from scratch. The training stopped if the performance (mAP@50 of the validation dataset) did not increase for 50 epochs. For comparison, one YOLOv3 model was trained for 2000 epochs.

	tiny-YOLOv3 pretrained	**tiny-YOLOv3 pretrained**	**tiny-YOLOv3 scratch**	**nano-YOLOv8 scratch**	**small-YOLOv8 pretrained**	**nano-YOLOv8 scratch**	**small-YOLOv8 scratch**
Training time	54min	55min	30.6h	0.81h	1.2h	1.9h	1.7h
Epochs	59	55	2000	117	118	274	177
Model size	33.1MB	33.1MB	33.1MB	5.9MB	21.4MB	5.9MB	21.4MB
FPS (GPU)	44.8	44.8	44.8	83.1	76.7	83.1	76.7
FPS (CPU)	7.7	7.7	7.7	18.5	8.8	18.5	8.8
TNR	1	0.996	0.972	0.968	0.986	0.976	0.98
Precision	1	0.996	0.972	0.969	0.986	0.976	0.98
Recall	0.99	0.982	0.99	0.998	0.994	0.996	0.99
Accuracy	0.995	0.989	0.981	0.983	0.99	0.986	0.985

REFERENCES

[1] L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson, and I. Crnkovic, "A taxonomy of software engineering challenges for machine learning systems: An empirical investigation," in Agile Processes in Software Engineering and Extreme Programming (P. Kruchten, S. Fraser, and F. Coallier, eds.), (Cham), pp. 227–243, Springer International Publishing, 2019.

[2] Jellyfish, "AI Engineer vs. Software Engineer," 2023.

[3] M. Camilli, M. Felderer, A. Giusti, D. T. Matt, A. Perini, B. Russo, and A. Susi, "Risk-Driven Compliance Assurance for Collaborative AI Systems: A Vision Paper," pp. 123–130, 2021.

[4] J. G. Adigun, M. Camilli, M. Felderer, A. Giusti, D. T. Matt, A. Perini, B. Russo, and A. Susi, "Collaborative artificial intelligence needs stronger assurances driven by risks," Computer, vol. 55, no. 3, pp. 52– 63, 2022.

[5] Universal Robots, "New Technical Specification on Collaborative Robot Design," 2018.

[6] E. Commission, D.-G. for Research, and Innovation, Industry 5.0: human-centric, sustainable and resilient. Publications Office, 2021.

[7] C. Birchler, S. Khatiri, B. Bosshard, A. Gambi, and S. Panichella, "Machine learning-based test selection for simulation-based testing of self-driving cars software," Empirical Software Engineering, vol. 28, p. 71, Apr. 2023.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 7, pp. 138–148, 2015.

[9] M. A. Santana, R. Calinescu, and C. Paterson, "Risk-aware realtime object detection," in 2022 18th European Dependable Computing Conference (EDCC), pp. 105–108, 2022.

[10] W. McCulloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," Bulletin of Mathematical Biophysics, vol. 5, pp. 127–147, 1943.

[11] P. Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. PhD thesis, Harvard University, Cambridge, MA, USA, 1974.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Nature, vol. 323, no. 6088, pp. 533– 536, 1986.

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," Neural Computation, vol. 1, no. 4, pp. 541–551, 1989.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Communications of the ACM, vol. 60, no. 6, pp. 84–90, 2012.

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," International Journal of Computer Vision (IJCV), vol. 115, no. 3, pp. 211–252, 2015.

[16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," European Conference on Computer Vision, vol. 9905, pp. 21–37, 2016.

[17] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for´ dense object detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 2, pp. 318–327, 2018.

[18] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[19] G. Jocher, A. Chaurasia, and J. Qiu, "YOLO by Ultralytics," Jan 2023.

[20] Coppelia Robotics, "CoppeliaSim."

[21] R. Ierusalimschy, "Lua," 1993.

[22] Tzutalin, "LabelImg." Git code, 2015.

[23] Moses, "Imageai, an open source python library built to empower developers to build applications and systems with self-contained computer vision capabilities," mar 2018–.

[24] 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, HighPerformance Deep Learning Library," in Advances in Neural Information Processing Systems 32 (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alche Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran´ Associates, Inc., 2019.

[25] S. Bozinovski and A. Fulgosi, "The influence of pattern similarity and transfer learning upon the training of a base perceptron B2.," Proceedings of Symposium Informatica, vol. 3-121-5, 1976.

[26] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A Comprehensive Survey on Transfer Learning," 2020.

[27] O. Ben-Kiki, C. Evans, and B. Ingerson, "YAML ain't markup language (YAML) (tm) version 1.2," 2009.

[28] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," Information Processing Management, vol. 45, no. 4, pp. 427–437, 2009.

[29] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A survey on performance metrics for object-detection algorithms," in 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), pp. 237–242, 2020.

[30] D. Shah, "Mean Average Precision (mAP) Explained: Everything You Need to Know," 2022.

[31] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollar, "Microsoft coco:´ Common objects in context," 2014.

[32] K. Liu, H. Tang, S. He, Q. Yu, Y. Xiong, and N. Wang, "Performance validation of yolo variants for object detection," in Proceedings of the 2021 International Conference on Bioinformatics and Intelligent Computing, BIC 2021, (New York, NY, USA), p. 239–243, Association for Computing Machinery, 2021.