Embedded Selforganizing Systems

Evaluating YOLOv7 Deployment Performance on Diverse Platforms and Strategies for Real-time Object Detection

Fatima Abouzid
Electrical Engineering Department
Faculty of Science and Technology
University Sidi Mohamed Ben Abdellah
Fez, Morocco
Fatima.abouzid@usmba.ac.ma

Shadi Saleh
Department of Computer Engineering
Chemnitz University of Technology
Chemnitz, Germany
shadi.saleh@informatik.tu-chemnitz.de

Mohamed Salim Harras
Department of Computer Engineering
Chemnitz University of Technology
Chemnitz, Germany
mohamed-salim.harras@informatik.tu-chemnitz.de

Wolfram Hardt
Department of Computer Engineering
Chemnitz University of Technology
Chemnitz, Germany
wolfram.hardt@informatik.tu-chemnitz.de

Ghennioui Hicham
Electrical Engineering Department
Faculty of Science and Technology
University Sidi Mohamed Ben Abdellah
Fez, Morocco
ghennioui@gmail.com

***Abstract*— In the past few years, machine learning (ML) has evolved from academic research curiosity into a practical domain capable of addressing tangible business challenges. Deploying machine learning (ML) models to production with an equivalent degree of thoroughness and automation as conventional software systems has proven to be a complex endeavor, demanding additional attention and infrastructure to address the unique challenges involved. Building and training machine learning models is just one part of the process. Deploying models into production is a critical step that brings the potential of these models to life and allows them to deliver value to organizations and end users. It's the bridge between the theoretical work in the research and the practical application of machine learning in real-world scenarios. This research paper presents an original study focused on deploying a traffic sign detection and recognition system utilizing the YOLOv7 algorithm. The study evaluates the algorithm's performance across diverse platforms and determines a robust production deployment strategy for YOLOv7.**

Keywords —Traffic sign, Detection, Classification, Deep Neural networks, YOLOv7, Deployment

I. INTRODUCTION

Many researchers focus on building and training machine learning models, but it is important to note that having a model file on a local computer is just one part of the process. The true impact of machine learning is realized when these models are effectively deployed in production environments, where clients and end-users can use them to make real-time decisions and automate tasks[1].

This transition comes with challenges. Similar to other domains, there are notable distinctions between what proves effective in an academic environment and what a real-world system demands. The principal difficulties when deploying machine learning models in production include constraints on scalability, the absence of real-time interaction, and the challenge of providing user access to the models [2].

Conversely, the deployment of machine learning offers a multitude of benefits, such as improving user access to model predictions in practical scenarios, enabling real-time decision-making, and swiftly adjusting to dynamic circumstances. Additionally, machine learning can enhance model performance, leading to higher accuracy rates when compared to traditional methods.

Over the years, traffic sign detection and recognition systems have given extra value to driver assistance, leading to a more user-friendly driving experience and much improved safety for passengers[3]. It helps regulate the number of accidents and helps drivers who commit traffic mistakes, particularly those who fail to heed traffic signs[4].

Traffic signs have a dual role: the first one, they regulate the traffic and the second one is to indicate the state of the road by guiding and warning the drivers and pedestrians. Those who drive vehicles need to learn to detect and recognize all traffic signs for traffic safety reasons[5], drivers are required to process the knowledge of cyclist signs, pedestrian signs, mandatory signs and advisory signs to reduce some unwanted circumstances during driving especially in Morocco where the driving task is complex because of road and weather conditions[6].

This work provides a comparative study -in terms of speed- of the best known and most efficient traffic sign recognition deployment methods based on the YOLOv7 algorithm.

The paper is structured as follows: The second section delves into a review of prior work concerning traffic sign recognition systems and deployment approaches. The third section offers an in-depth survey of traffic sign detection for driver assistance using the YOLOv7 algorithm. Subsequently, the fourth part introduces the study's proposal and analyzes the primary experimental results to identify a resilient deployment strategy for YOLOv7. Moving forward, the fifth section

discusses the current landscape of this field and outlines potential avenues for future research. Lastly, the sixth section draws the paper to a conclusion.

II. Related Work

A. DevOps

The DevOps cycle is a continuous and iterative process that integrates development (Dev) and operations (Ops) in software development and deployment[7]. It is a software development methodology that streamlines the software lifecycle by integrating development and operations, increasing speed, efficiency, and output. By reducing the number of steps in the lifecycle, DevOps enables teams to work more collaboratively and deliver high-quality software more quickly[8]. It aims to enhance team productivity and outcomes by promoting collaboration, automation, and an iterative software development and operations management approach. By implementing various strategies, teams can streamline workflow and deliver high-quality software more efficiently.

1) DevOps Lifecycle

The DevOps lifecycle (See Fig. 2), also known as the DevOps pipeline, is a continuous software development process that incorporates DevOps best practices at every stage, from planning and building to deployment and monitoring. By providing continuous feedback, teams can improve the software and deliver high-quality results [7].

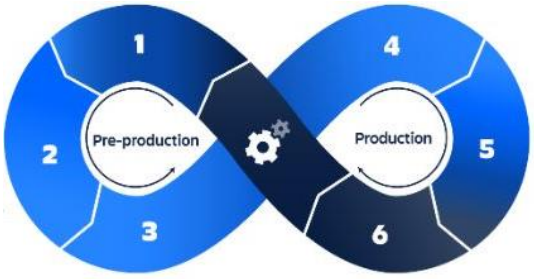

Fig. 2. DevOps Lifecycle [6]

The DevOps lifecycle typically consists of the following stages [6]:

- ***Plan***: Define the project goals, requirements, and success criteria.
- ***Code***: Write the code for the application or feature.
- ***Build***: Compile the code into a deployable artifact.
- Test: Run automated tests to ensure the code works as expected.
- ***Deploy***: Release the code into the production environment.
- ***Monitor***: Continuously monitor the application for performance, errors, and other issues.
- ***Feedback***: Collect feedback from users and stakeholders to improve the application.

The DevOps lifecycle is designed to be iterative and continuous, with each stage feeding into the next. By automating as many steps as possible, teams can reduce the time it takes to deliver new features and improve the quality of their software[9].

2) DevOps for Machine Learning (MLOps):

MLOps, short for "Machine Learning Operations", is a set of practices, principles, and tools that combine machine learning (ML) and artificial intelligence (AI) with the principles of DevOps to enable the end-to-end management of ML models and workflows. MLOps aims to streamline and automate the deployment, monitoring, and management of machine learning models in production environments, similar to how DevOps does for software development and IT

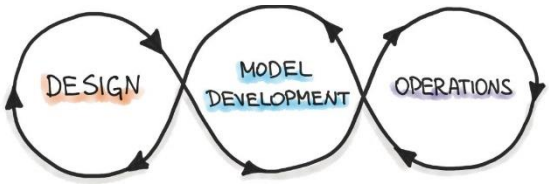

Fig. 1. MLOps Pipeline [8]

operations [10]. The process is highlighted in Fig. 1.

MLOps is a collaborative, continuous process that emphasizes the operationalization of data science by treating statistical, data science, and machine learning models as reusable, highly available software artifacts. This is achieved through a repeatable deployment process focusing on managing these models as a cross-functional team.

The three main stages of the MLOps process are "Designing the ML-powered application", "ML Experimentation and Development," and "ML Operations"[11]. The first stage comprises tasks linked to digesting the data, building the software with machine learning capabilities, and understanding the business objectives. Then, by building a Proof-of-Concept for an ML model, the 'ML Experimentation and Development' phase is focuses on verifying that machine learning is feasible for the specified issue. The main goal of the 'ML Operations' phase is to implement the previously developed ML model in a real-world setting. Using well-established DevOps techniques—including testing, version control, continuous delivery, and continuous monitoring [11].

B. YOLOv7 Architecture

YOLOv7 is one of the models in the YOLO (You Only Look Once) series of object detection. The YOLO framework has three main components: Backbone, head and neck (Fig. 3).

The Backbone mainly extracts essential features of an image and feeds them to the Head through Neck. The neck collects feature maps extracted by the Backbone and creates feature pyramids. Finally, the head consists of output layers that have final detections [12].

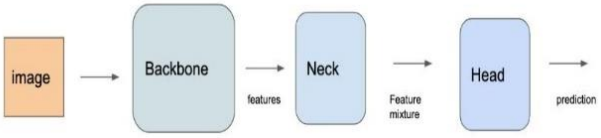

Fig. 3. YOLO Architecture [10]

In 2022, the YOLOv7 [13]algorithm is making big waves in the computer vision and machine learning communities. Its performance was evaluated based on previous YOLO versions

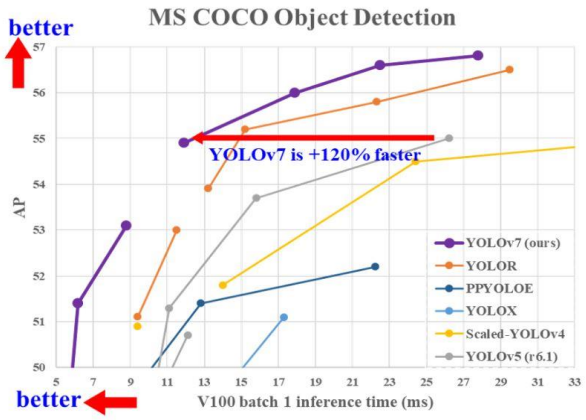

Fig. 5. COCO Benchmark for YOLO Models [11]

(YOLOv4 and YOLOv5) and YOLOR as baselines. However, it surpasses all previous object detectors in terms of both speed and accuracy, ranging from 5 FPS to as much as 160 FPS [14]. As seen in Fig. 5, the benchmark showcases the difference in average precision in relation with time.

C. *Deployment Strategies*

Deep learning computations may be run on a variety of devices, including edge servers (ESs), cloud servers, and edge devices (EDs). This determines the following computing paradigms [12].

1) Cloud computing:

Cloud computing is a paradigm for extensive distributed computing that uses of technologies like virtualization, grid computing, and service orientation. It provides the ability to quickly acquire and release customizable computing resources from a shared pool of infrastructure, with little participation from the server infrastructure provider [15].

2) Edge computing:

Moving processing resources physically closer to the location where data is generated, often a sensor or Internet of Things device, is known as edge computing. Edge computing, is so named because it brings compute capacity to the network or device's edge, enables quicker data processing, more bandwidth, and guaranteed data sovereignty[15].

Many enterprises deploy their AI applications using edge computing, which refers to processing that happens where data is produced. Edge computing handles and stores data locally in an edge device instead of cloud processing in a distant, centralized data reserve[16]. Consequently, cloud and edge computing may cooperate and have a range of advantages and use cases.

III. EXPERIMENTS

This section describes the efforts undertaken during our research. first, we commence by assembling the dataset and initiate the training process, in order to select the most suitable model for deployment.

A. *Model Selection and Training:*

There are several steps involved in the applied ML process. Although the steps are the same, the descriptions may change in what they call for them. We can define the process using four high-level steps: define the problem, prepare data, evaluate models, and finalize the model [17].

1) Data Preparation:

Once we have identified the problem and collected the data useful in traffic sign detection, we initiate the essential second step in ML process, data preparation. Since the predictions made by ML systems can only be as good as the data on which they have been trained.

The designs of traffic signs are standardized through laws but differ across the world. In Morocco as in other countries, shapes and colors are used to categorize different types of signs: circular red signs are prohibitions, circular blue signs are obligations, triangular red signs are warnings, and rectangular blue signs are recommendations and traffic lights. The recognition algorithm requires an extensive database with several images of traffic signs, but in a Moroccan context, it was evident that there is a lack of databases with Moroccan traffic signs.

The dataset comprises approximately 2000 images in different conditions: at night and during bad weather to prove the robustness of the model that will be chosen. The dataset is partitioned into distinct subsets to facilitate the model's development and assessment. Specifically, 70% of the data is allocated for training, 20% for testing, and the remaining 10% for validation.

Fig. 4. Dataset Samples

We ensure balance within our dataset across the five classes.: obligation, recommendation, warning, light, and prohibition. This is showcased in the class distribution in Fig. 6.

2) Model Selection and Training

Fig. 6. Dataset Class Distribution

While training a machine learning model using the dataset prepared, we opted to work with the YOLOv7 algorithm due to its exceptional speed and accuracy. The model learns to predict and classify traffic sign panels during this process.

The model underwent training on Google Colab, utilizing a Tesla T4 GPU. The training process was divided into three segments: the first involved 55 epochs, the second extended to 110 epochs, and the third encompassed 126 epochs (See Table I).

TABLE I. TRAINING HYPERPARAMETERS

Trainings	YOLOv7 Hyperparameters	
1st Training	Epoch	55
	Batch size	16
2nd Training	Epoch	110
	Batch size	16
3rd Training	Epoch	126
	Batch size	16

The third section, consisting of 126 epochs and a batch size of 16, produced weights that demonstrated strong performance, particularly in terms of precision, recall, and mAP@0.5. This is the key reason why we decided to choose this section for the deployment phases.

B. Model Deployment

Model deployment represents a challenging phase within the MLOps cycle. It necessitates careful planning, robust infrastructure, and collaboration among multidisciplinary teams.

Hence, efficient deployment is a critical requirement to ensure that machine learning models consistently deliver optimal performance in production environments and achieve their expected value. It demands constant focus, proactive observation, and a dedication to change and advancement.

In this section, we initiated the model deployment process. This part consists of several steps, commencing with the selection of deployment strategies, followed by the identification of the resources employed and the deployment frameworks utilized.

1) Deployment Strategies and Resources

Many businesses choose to implement edge computing for their AI applications due to its performance, security, cost-effectiveness, and adaptability benefits. It is a popular choice across diverse use cases, which is why we decided to deploy our machine learning model on the edge.

We chose to deploy our model across various resources to ensure optimal machine learning deployment performance (Table II). Initially, we conducted deployments on our local PC, followed by Nvidia Jetson Nano, and finally, on Google Colab. This approach allowed us to thoroughly evaluate the model's performance and inference times across different resources, helping us identify the most robust option.

2) Deployment Frameworks

Several frameworks available for deploying machine learning, each with its own distinct features and capabilities. In our study, we decided to deploy the machine learning model across various frameworks to identify the one that demonstrates the most incredible robustness and compatibility, particularly with YOLOv7.

TABLE II. RESOURCE ANALYSIS

	Pc	Jetson nano	Google Colab
CPU	Intel® Core™ i5-6th 2.5 GHz	ARMv8 rev 1 (v8l) *4	Intel Xeon CPU
GPU	-	NVIDIA Tegra X1 CUDA v10.2	Tesla T4 CUDA v11.2
RAM	8.00 GB	4.00 GB	13.00 GB
OS	Windows 10	Ubuntu 18.04 LTS	-

From the numerous frameworks accessible for machine learning deployment, we opted to deploy our model with PyTorch, ONNX, TensorRT, and TensorFlow Lite. Each of these frameworks possesses distinct characteristics that set it apart from the others. The table displayed below offers an overview of the unique attributes associated with each framework. (Table III)

TABLE III. COMPARISON OF DEPLOYMENT FRAMEWORKS UTILIZED.

	PyTorch (.pt)	ONNX (.onnx)	TensorRT (.trt)	TensorFlow Lite (.tflite)
Version	v 1.12.0	v 1.12.0	v 11.2	TensorFlow 2.x v
Advantages	Adaptable for various deployment scenarios. Easy to use. Compatible with graphics cards (GPU)[18].	Easy to access hardware acceleration. Supports popular DL frameworks[19].	High-speed inference. Flexibility in deploying models [20].	Optimizing existing models to be less memory and cost-consuming [21].
disadvantages	Large Model Sizes Limited production optimizations[18].	Not designed for model training [19].	Designed to work with NVIDIA GPUs [20].	Performance depends on the hardware it's deployed on [21].

IV. RESULTS

Here, we provide the findings from our investigation on machine learning model deployment and how well they operate in different deployment scenarios, platforms, and frameworks. Our work is motivated by the urgent need to close the knowledge gap between the creation of machine learning models and their successful implementation in practical applications.

Before discussing the details of our findings, it is critical to highlight that our results are significant not just for machine learning practitioners but also for companies and

organizations looking to integrate AI and machine learning into their daily operations.

A. Training Results

This part of our research was designed to analyze the behavior of training models, optimization methods, and the efficacy of the YOLOv7 algorithm. The training phase forms the fundamental step of machine learning, where models learn to extract patterns, make predictions, and drive innovations.

After three hours of training the model, the obtained results are received from the confusion matrix, which offers detailed information about predictions for each class, facilitating a deeper understanding of the model's performance across the five classes.

Here, it is clear that the model has high accuracy, precision, and average precision with 0.967 for all classes. The 'recommendation' class shows the best results compared with other classes. It stands out with the highest recall score of 1 and an impressive mAP@0.5 of 0.991.

The 'prohibition' class exhibits relatively lower results in comparison to other classes, with notable figures of 0.807 in precision and 0.897 in mAP@0.5. This indicates that more development of this class is necessary to improve its accuracy and performance.

B. Deployment results:

Regarding the results of ML deployment, here, we provide the results of running the model on several platforms, providing a thorough rundown of the results for every resource and framework.

Beginning with the results from PyTorch, the Table IV presents the inference times (IT), including the highest (HV), lowest (LV), and average (AV) values for model deployment on various resources: local PC, Google Colab, and Jetson Nano. Notably, the GPU demonstrates a significant average inference time of 15.4 ms when compared to CPUs. On the other hand, the CPU of Google Colab delivers favorable results when compared to the CPUs of other resources. The inability to utilize a GPU on Jetson Nano is primarily due to the absence of GPU support on Jetson Nano, and mainly because of its lack of CUDA compatibility. The

Table V highlights the ONNX results, it shows that the CPU of Google Colab performs remarkably well compared to the CPUs of other resources. In this experiment, the pytorch wights are converted to ONNX and then the inference is performed, giving us the following results.

TABLE IV. COLLECTED RESULTS FOR PYTORCH FRAMEWORK ON DIFFERENT RESOURCES

		Local		Colab		Jetson Nano	
		CPU [s]	GPU [ms]	CPU [s]	GPU [ms]	CPU [s]	CPU [s]
IT	HV	2.56	-	2.19	15.7	3.01	-
	LV	1.37	-	1.32	14.3	2.95	-
	AV	1.96	-	1.76	15.4	2.98	-

TABLE V. COLLECTED RESULTS FOR ONNX FRAMEWORK ON DIFFERENT RESOURCES

		Local		Colab		Jetson Nano	
		CPU [s]	GPU [ms]	CPU [s]	GPU [ms]	CPU [s]	GPU [ms]
IT	HV	4.13	-	3.78	19.6	4.62	-
	LV	2.45	-	1.59	18.1	3.16	-
	AV	3.29	-	2.67	18.5	3.89	-

Recording its limited compatibility with other hardware, TensorRT is highly designed for NVIDIA GPUS. It is an NVIDIA library and runtime for optimizing deep learning models for deployment on GPUs, with a focus on enhancing inference performance. Hence, in the table below, we observe that there are no results available for the CPU, with results exclusively presented for the GPU (Table VI).

TABLE VI. COLLECTED RESULTS FOR TENSORRT FRAMEWORK ON DIFFERENT RESOURCES

		Local		Colab		Jetson Nano	
		CPU [s]	GPU [ms]	CPU [s]	GPU [ms]	CPU [s]	GPU [ms]
IT	HV	-	-	-	10.2	-	-
	LV	-	-	-	8.9	-	-
	AV	-	-	-	9.6	-	-

Relating to TensorFlow Lite, which is comparatively more resource-intensive than other frameworks, the table below illustrates that the inference time on Colab consistently remains shorter than on other resources (Table VII).

TABLE VII. COLLECTED RESULTS FOR TENSORFLOW LITE FRAMEWORK ON DIFFERENT RESOURCES

		Local		Colab		Jetson Nano	
		CPU [s]	GPU [ms]	CPU [s]	GPU [ms]	CPU [s]	GPU [ms]
IT	HV	6.42	-	5.62	212.3	6.93	-
	LV	6.12	-	5.48	182.6	6.87	-
	AV	6.27	-	5.56	194.2	6.87	-

While analyzing the various results, we noticed a contradiction in the outcomes: one set of results pertained to GPU performance, while the other set focused on CPU performance (Fig. 7 and Fig. 8).

Regarding the GPU, it is important to highlight that TensorRT exhibits strong performance, delivering lower inference times. On the other hand, PyTorch delivers strong performance and provides favorable results on the CPU, leading to shorter inference times.

Fig. 7. CPU-based Deployment Results

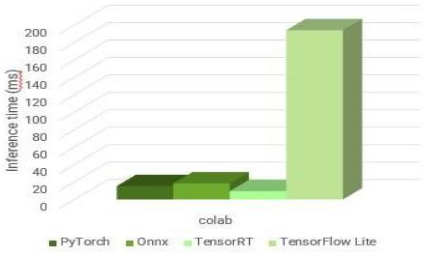

Fig. 8. GPU-based Deployment Resutls

V. Conclusion and Limitations

In conclusion, our comprehensive study on machine learning deployment has illuminated the complexities, challenges, and opportunities in efficiently utilizing the power of AI models for real-world applications. Throughout this research, we explored the various stages of model deployment, from data preparation to selecting deployment frameworks, and even extending to evaluating their performance on different resources.

The research followed two principal objectives. The study's primary objective was to develop a traffic sign detection and recognition system. Utilizing the YOLOv7 algorithm, beginning with data collection and preparation, model training, and the eventual model selection. The second objective involved evaluating the algorithm's performance across diverse platforms and selecting a robust production deployment strategy for the model.

The training phase produced a high-performance model with an impressive mAP@0.5 of 0.967, which was subsequently chosen for the deployment stage. This model was deployed throughout various resources, including PCs, Google Colab, and Jetson Nano, utilizing both GPU and CPU. The inference time data indicates that TensorRT excels in GPU performance, while PyTorch delivers robust performance on the CPU.

The constraints identified in our study, including the inability to use GPU on Jetson Nano, the complexity of the YOLOv7 model, its size, and the number of classes, will be addressed in our future work. We plan to evaluate these limitations and explore alternative deployment strategies using other boards equipped with GPUs.

References

[1] U. Bhatt *et al.*, "Explainable machine learning in deployment," *FAT* 2020 - Proc. 2020 Conf. Fairness, Accountability, Transpar.*, pp. 648–657, 2020, doi: 10.1145/3351095.3375624.

[2] A. Paleyes, R. G. Urma, and N. D. Lawrence, "Challenges in Deploying Machine Learning: A Survey of Case Studies," *ACM Comput. Surv.*, vol. 55, no. 6, pp. 1–29, 2022, doi: 10.1145/3533378.

[3] S. Saleh, C. Rellan, S. P. Surana, and J. Nine, "Collision Warning Based on Multi-Object Detection and Distance Estimation Network Simulation View project Cross-Correlation with Up-Sampling View project," no. January 2021, 2020, [Online]. Available: https://www.researchgate.net/publication/348155370

[4] A. Møgelmose, M. M. Trivedi, and T. B. Moeslund, "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 4, pp. 1484–1497, 2012, doi: 10.1109/TITS.2012.2209421.

[5] S. Saleh, S. A. Khwandah, A. Mumtaz, A. Heller, and W. Hardt, "Traffic signs recognition and distance estimation using a monocular camera," *CEUR Workshop Proc.*, vol. 2514, no. November, pp. 407–418, 2019.

[6] S. Soo, "Object detection using Haar-cascade Classifier Object detection using Haar-cascade Classifier," *Academia*, pp. 1–12, 2014, [Online]. Available: https://www.academia.edu/38877608/Object_detection_using_Haar_cascade_Classifier

[7] M. Gokarna and R. Singh, "DevOps: A Historical Review and Future Works," *Proc. - IEEE 2021 Int. Conf. Comput. Commun. Intell. Syst. ICCCIS 2021*, pp. 366–371, 2021, doi: 10.1109/ICCCIS51004.2021.9397235.

[8] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Softw.*, vol. 33, no. 3, pp. 94–100, 2016, doi: 10.1109/MS.2016.68.

[9] "devops-lifecycle @ www.browserstack.com." [Online]. Available: https://www.browserstack.com/guide/devops-lifecycle

[10] A. B. Kolltveit and J. Li, "Operationalizing Machine Learning Models - A Systematic Literature Review," *Proc. - Work. Softw. Eng. Responsible AI, SE4RAI 2022*, pp. 1–8, 2022, doi: 10.1145/3526073.3527584.

[11] "mlops-principles @ ml-ops.org." [Online]. Available: https://ml-ops.org/content/mlops-principles

[12] "652a482cfdce57aac13f06ccda16905e5345cff2 @ learnopencv.com." [Online]. Available: https://learnopencv.com/yolov7-object-detection-paper-explanation-and-inference/

[13] "understanding-yolov7-neural-network-343889e32e4e @ medium.com." [Online]. Available: https://medium.com/@nahidalam/understanding-yolov7-neural-network-343889e32e4e

[14] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors," pp. 7464–7475, 2023, doi: 10.1109/cvpr52729.2023.00721.

[15] P. Joshi, M. Hasanuzzaman, C. Thapa, H. Afli, and T. Scully, "Enabling All In-Edge Deep Learning: A Literature Review," *IEEE Access*, vol. 11, no. December 2022, pp. 3431–3460, 2023, doi: 10.1109/ACCESS.2023.3234761.

[16] "33fe758121d8c2765c578d4bdd3e7bfca40e8696 @ blogs.nvidia.com." [Online]. Available: https://blogs.nvidia.com/blog/2022/01/05/difference-between-cloud-and-edge-computing/

[17] L. Y. Deng, M. Garzon, and N. Kumar, "What is dimensionality reduction (dr)?," *Dimens. Reduct. Data Sci.*, pp. 67–77, 2022, doi: 10.1007/978-3-031-05371-9_3.

[18] "pytorch-tout-savoir @ datascientest.com." [Online]. Available: https://datascientest.com/pytorch-tout-savoir

[19] "two-benefits-of-the-onnx-library-for-ml-models-4b3e417df52e @ medium.com." [Online]. Available: https://medium.com/trueface-ai/two-benefits-of-the-onnx-library-for-ml-models-4b3e417df52e

[20] "c4a60abc7bc3349f66acb3de35fb219511cfb0cc @ itsocial.fr." [Online]. Available: https://itsocial.fr/actualites/nvidia-annonce-tensorrt-llm-sa-bibliotheque-opensource-pour-accelerer-le-developpement-de-lia/

[21] "cd54b39442713720ef91d9daa105de4492a07d28 @ viso.ai." [Online]. Available: https://viso.ai/edge-ai/tensorflow-lite/#:~:text=TF Lite can optimize existing,use cases of TensorFlow Lite.