

Real-Time Onboard Object Detection for Augmented Reality: Enhancing Head-Mounted Display with YOLOv8

Mikołaj Łysakowski*, Kamil Żywanowski*, Adam Banaszczyk*, Michał R. Nowicki*†, Piotr Skrzypczyński*†, Sławomir K. Tadeja‡

* Poznań University of Technology, Centre for Artificial Intelligence and Cybersecurity

† Poznań University of Technology, Institute of Robotics and Machine Intelligence

‡ University of Cambridge, Department of Engineering, Institute for Manufacturing

Abstract—This paper introduces a software architecture for real-time object detection using machine learning (ML) in an augmented reality (AR) environment. Our approach uses the recent state-of-the-art *YOLOv8* network that runs onboard on the *Microsoft HoloLens 2* head-mounted display (HMD). The primary motivation behind this research is to enable the application of advanced ML models for enhanced perception and situational awareness with a wearable, hands-free AR platform. We show the image processing pipeline for the *YOLOv8* model and the techniques used to make it real-time on the resource-limited edge computing platform of the headset. The experimental results demonstrate that our solution achieves real-time processing without needing offloading tasks to the cloud or any other external servers while retaining satisfactory accuracy regarding the usual mAP metric and measured qualitative performance.

Index Terms—augmented reality, machine learning, real-time object detection, edge computing

I. INTRODUCTION

Augmented Reality (AR) technology belonging to the class of *immersive technologies* offers an ability to blend digital artifacts and the physical environment by superimposing digital content in the user’s field of view (FoV) [1] (Fig. 1).

Presently, popular AR applications running on mobile devices, such as smartphones or tablets, can be further enhanced with machine learning (ML). Thanks to such an approach, we can include vision-based features for object detection and tracking on video and imagery data [2].

However, mobile AR solutions have significant limitations, such as a relatively small FoV confined by screen canvas or needing hand control [3]. The latter narrows down potential scenarios where we can successfully deploy AR, such as manual assembly [4], device repair task [5], or the use of AR enhancers by older adults [6]. In such cases, the user’s ability to not only freely move hands but promptly shift the unconstrained FoV or the body posture is crucial for safety concerns and task completion [5], [7].

These caveats are circumvented by the alternative technology of wearable smart head-mounted display (HMD) [4]. The AR headsets, such as widely-considered state-of-the-art

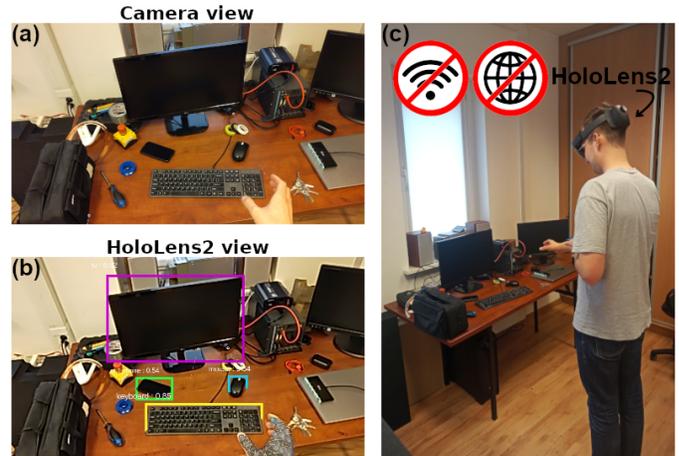


Fig. 1: The proposed onboard object detection with *YOLOv8* offers real-time onboard object detection enhancing *HoloLens 2* capabilities without a common requirement of WiFi or Internet access to perform server-based computations.

Microsoft HoloLens 2 (HL2) [8], offer a hands-free AR experience [4], [5]. Unfortunately, HL2 and other similar headsets do not offer a satisfactory level of support for ML-based processing that could enhance the user’s ability to interact with the environment [9], [10]. Thus, having onboard, real-time ML models running in the headset’s edge computing platform is crucial for developing new AR application areas.

We address the problem of real-time object detection on the HL2, including the most recent *You Only Look Once* [11] *YOLOv8* framework. We focus on defining the steps necessary to achieve a desired frame rate of image processing with the onboard ML model while identifying the constraints of the HL2 computing platform. Overcoming these limitations enables using widely-available ML algorithms on headsets. We also believe that AR developers can use our work on *YOLOv8* for HL2 to create new applications extending the current use cases of this headset.

Our contribution can be summarized as a unique, easy-to-replicate, real-time, onboard object detection pipeline on the

The study has been supported by funding provided through an unrestricted gift by Meta.

HL2 headset. Following the open-science principle, our code and complete guide on how to run the most recent YOLOv8 network architecture in the resource-limited hardware environment of this HMD is freely available as a GitHub¹ repository. Furthermore, as a byproduct of our work, we also developed a commented list of limitations of the HL2 as a computing platform for ML. These should be tackled first to broaden the development of modern ML-based applications on this widely used among AR community device [4], [5].

II. RELATED WORK

The usage of object detection on AR headsets is an item of past and current research explorations [13]–[16]. For example, in [13], the authors explore the possibility of using a no longer state-of-the-art object detection approach with a two-stage network to detect and track objects while offloading the processing to the high-end server. We argue that while on the server side, we are not constrained by the computational capabilities of HL2, we cannot use the headset to carry out object-detection tasks without local WiFi or a fast wireless broadband communication (e.g., LTE connection). Off-board processing limits the HL2 headset to only a frame-capturing camera and head-mounted display output than a standalone solution for object detection and tracking. Presently, the Microsoft Azure Custom Vision library [17] offers a complete high-level solution when considering off-board computations.

An example application is tomato picking, where farmers rely on inexperienced, part-time workers to harvest the fruits with desired ripeness and blemishes [14]. It negatively impacts harvest quality and work efficiency as people need to learn the task. The presented work shows a complete AR and ML-based solution to this problem that requires powerful servers needing a steady, reliable LTE connection, which might not be available in the field. Another example concerns a system for supporting visually-impaired people, utilizing object detection in images to provide information about objects in the surroundings using an audio interface to the user [15]. The application directs users to the desired objects in the scene based on audio communication. This system uses an old YOLOv2 model that is offloaded to the server, thus making it vulnerable to LTE connection stability outdoors, hampering its ability to efficiently determine desired objects’ locations. Similarly, in [16], we are introduced to the concept of using HL2 as a tool to increase drivers’ road condition awareness. The authors explore the idea of a system that can focus users’ attention on incoming vehicles and support lane detection while extending the view with vehicle speed. In such a context, on-time, local processing capabilities are crucial to ensure the proper operation of the complete solution. These examples are only a subset of the existing works concerned with object detection and tracking in headset-based AR that rely on server-side computation [18], [19], summarized in a recent review paper [20].

¹https://github.com/kolaszko/hl2_detection

The alternative to server-side processing includes simpler algorithms that run in real-time on AR headsets. Such an approach is proposed in [21], where a custom-made processing pipeline is used to detect cracks in the constructions using only the onboard computation capabilities of HL2. Designing custom ML pipelines offers the desired object detection accuracy but requires expert knowledge and is time-consuming. Hence, another approach is to apply a readily-available software framework, like *Vuforia* [22] or *easyAR* [23]. These solutions, however, require a 3D CAD model of the considered objects, which in practice limits detection capabilities to an object instance from a class of rigid objects.

As a middle ground, it is possible to combine server-side processing with on-device edge processing. Still, the resulting solution has to properly synchronize the processing on both ends, raising even more issues [24].

Consequently, we propose a pipeline that allows anybody to achieve real-time object detection and tracking capabilities using the state-of-the-art YOLOv8 network onboard HL2 without needing to be connected to any server. Furthermore, to achieve our real-time performance, we put a hard limit of 100 ms on end-to-end processing from an image capturing to data visualization as greater latency reduces an immersive experience to users [25], [26]. To that end, our approach offers new advantages to the ones already provided by a standalone AR headset such as HL2 [7].

III. HARDWARE AND SOFTWARE FRAMEWORKS

The general, high-level processing idea is presented in Fig. 2. We start by preparing the YOLOv8 neural network models for HL2. These models can be optionally retrained (fine-tuned) to include different detection classes. The next step involves exporting the model to the *Open Neural Network Exchange* (ONNX) [27] format. The model is then used by the *Barracuda* [28] library in the *Unity* [29] engine to perform object detection on HL2 and to provide visualization of the detected objects. We decided to use the Unity platform as it is among the most widely used software framework in AR [4], [5] and VR (virtual reality) research [30]. We will introduce the used frameworks in more detail in the following sections.

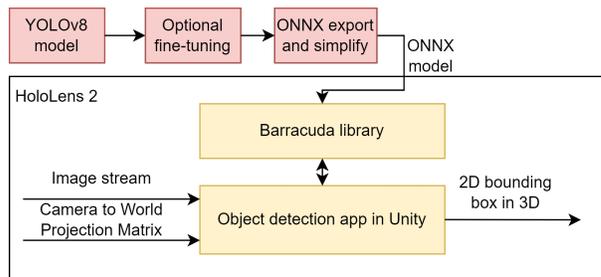


Fig. 2: The relation between the used hardware and software frameworks. Notice the clear distinction between the offline phase performed outside the AR device (red) and the online operation on HL2 (yellow).

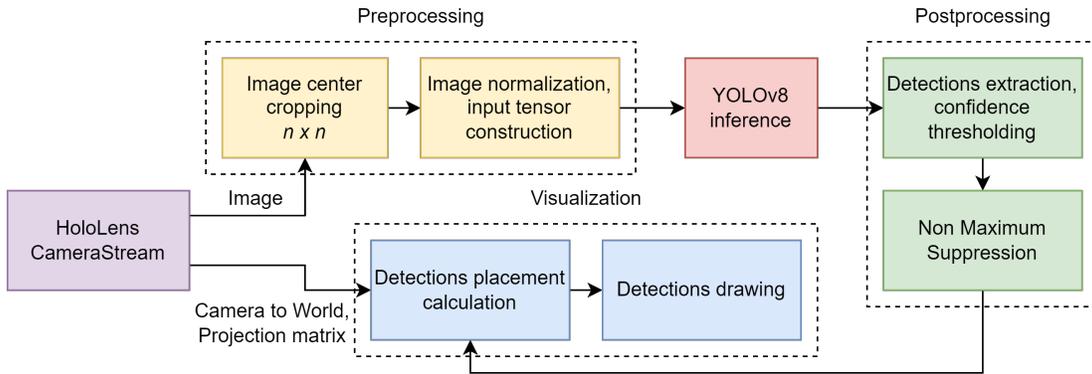


Fig. 3: The image processing steps performed on the HL2 to achieve the object detection within the user’s FoV.

A. Augmented Reality Equipment: Microsoft HoloLens 2

To showcase the possibility of optimizing an ML model performance on an HMD treated as an edge computing platform, we decided to use the *HoloLens 2* [8]. As a computing platform, this headset is equipped with Snapdragon 850, a high-performance 64-bit ARM LTE system on a chip designed by Qualcomm, and an Adreno 630 graphics processing unit (GPU). The headset also contains a second-generation custom-built holographic processing unit (HPU) for computations related to sensor information, core algorithm accelerators, and compute nodes enabling onboard image processing without using Snapdragon’s resources.

HL2 perception system consists of four grayscale cameras used in simultaneous localization and mapping (SLAM), two infrared (IR) cameras for the built-in eye tracking, a time-of-flight depth sensor used in hand tracking and spatial mapping, an inertial measurement unit (IMU) sensors, and frontal RGB camera. Our work uses a world-facing RGB camera mounted above the user’s eyes in the center of the front headset panel.

All these features and their widespread application in AR-related research [4], [5], made the HL2 a best-suited candidate for deploying and testing our real-time, onboard object detection with YOLOv8 network architecture.

B. Object detection: You Only Look Once (YOLO) network

Object detection is an active research topic with multiple scientific and real-world applications, as previously summarized [31]–[33]. The YOLO family [11] is commonly used when we need robust object detection capabilities [32]. YOLO model is based on single-stage object detection with different backbone sizes that can be chosen based on the available processing power [11]. Real-time object detection on constrained devices can be powered by YOLO Tiny or YOLO Nano, the smallest models in the YOLO family, which however, limits the performance [12].

Despite offering unrivaled results, this approach is still actively explored [33], with research focusing on achieving the best possible score (i.e., mean average precision, mAP) on the available datasets while doing it with the least amount of network parameters [31]. Currently, from the YOLO family

[11], the YOLOv8 network architecture gives the best results. Hence, we will focus on this version in our demo application.

C. Barracuda library for ML inference

The neural network part of the detection pipeline is based on the Barracuda [28] library. It is an open-source library developed by Unity [29] for utilizing neural networks in the game engine. It supports the most common deep learning layers and provides GPU and CPU inference engines. Cross-framework support for different machine learning libraries is ensured by using an ONNX format to load pretrained neural networks. It enables interoperability between different ML frameworks, providing a standard set of operations used in deep learning.

D. Model preparation

Each model used in the online operation can be prepared using the same pipeline. We export each model from *PyTorch* [34] serialized `.pt` file to the ONNX format. Since the current Barracuda version supports ONNX deep learning operations (opset) up to version 9, exporting models with the proper opset flag is crucial. Apart from the export, it is also possible to reduce the model with the ONNX simplification tool. The operation merges redundant operators using constant folding, consequently speeding up inference.

We successfully tested exporting and deploying the publicly available original YOLOv8 object detection models. Moreover, we can train the YOLOv8 for any custom class with sufficient data while following the guidelines for model fine-tuning to custom datasets.

IV. OBJECT DETECTION PIPELINE ON HL2

We present the universal pipeline for onboard, real-time YOLO-based object detection for HL2. The processing pipeline used in our evaluation is presented in Fig. 3.

The processing starts by an image acquisition done with `HoloLensCameraStream` [35] package. This plugin enables users to collect RGB camera images in all HL2-supported resolutions and frame rates, along with the current camera position in the world coordinate system. The package

provides the functionality essential to calculate a projection from pixel coordinates into 3D world space using extrinsic and projection matrices. The image, current camera-to-world matrix, and projection matrix are constantly updated in a separate thread whenever new data is available.

Next, we perform the initial image preprocessing step, which consists of cropping an $n \times n$ image out of the center of the acquired camera image, where n is the size of the neural network’s input, and all pixel values are normalized to $[0; 1]$ range. Since YOLOv8 accepts a square input, we have omitted using a rectangular image, simplifying the pre and postprocessing. Finally, the input tensor of size $(1, n, n, 3)$ is created. The image is then passed to the module for image inference with a YOLOv8-based model. The neural network structure and weights are loaded to `Unity.Barracuda.Model` using an ONNX file distributed as an asset inside the application. The procedure of model preparation is described in Sect. III-D. Once inference is completed, we have to parse the raw YOLO output detection into final detection, consisting of bounding box, object class and class score. At first, we determine a class of an object by choosing the one with the highest score for every detection in raw output. Next, the detections are filtered by a class score. The threshold can be selected using a precision-recall curve and depends on the requirements of the target application. Elements with scores lower than the given threshold are rejected. The next step is to perform *Non-Maximum Suppression* (NMS) [36] to discard overlapping boxes and select the best one.

The inference step of neural network inference produces a 2D bounding box on the image. We use the time of the original image acquisition from the camera-to-world matrix to project this 2D bounding box into the 3D scene observed by the user in AR. Based on this implementation, we can adequately annotate the 3D position of the object even if the user is looking in a different direction than when we captured the image for object detection.

V. EVALUATION

The proposed processing pipeline with YOLOv8 is evaluated under two key criteria for the final application: (1) processing time and (2) object detection performance. All experiments were performed using the *val2017* subset of Microsoft COCO dataset [37]. This image data collection is a large-scale object detection, segmentation, and captioning dataset containing 91 categories of ordinary objects. It is a common choice in object detection tasks regarding benchmarking methods and using COCO pretrained models to perform transfer learning and fine-tuning to adapt models to different detection tasks [38]. Weights of YOLOv8 models pretrained on COCO are available online.

We used `System.Diagnostics.Stopwatch` for processing time measurement with high-resolution performance counter mode on capturing either the whole processing or a selected part of the pipeline. Each time measurement was repeated 100 times after the model warm-up, i.e., several initial inferences that are necessary for each GPU application

to stabilize the processing times and further ensure fair and robust comparisons between different configurations. The HL2 battery charge level was over 50% for all trials, and the headset was not connected to any other device or power source.

A. Measuring the impact of YOLOv8 model size

The first design choice we have to make when using the YOLOv8 detector is selecting the network’s size among the available family of architectures. The YOLOv8 authors publicly share five pretrained networks that can be used out-of-the-box in the desired application, starting from the smallest network to the more extensive networks measured as a number of parameters: (i) nano (YOLOv8n), (ii) small (YOLOv8s), (iii) medium (YOLOv8m), (iv) large (YOLOv8l), and (v) extra large (YOLOv8x). Selecting a smaller network hinders the final performance while simultaneously taking less memory on the device and providing faster inference. Simultaneously, a larger network offers a better object detection performance.

Our experiments measured the processing time from the image capture moment to the bounding boxes projected in the 3D view. The comparison was performed with a usual image size of 224×224 pixels, and the results are presented in Fig. 4. These results suggest that for the best user experience in dynamic scenes, only the smallest YOLOv8n can meet the real-time requirements. Other models can still work in the resource-constrained environment of HL2. However, they might only be suited for scenarios where real-time performance is not vital for user experience, e.g., when used to classify the object held in hand or when the scene is not dynamic. In these cases, the object detection will still be able to properly place the object position in the user’s surroundings but will require more time to get these results.

The usage of network models with a lower number of parameters results in lower performance. The usual metric to quantify performance is mean average precision (mAP),

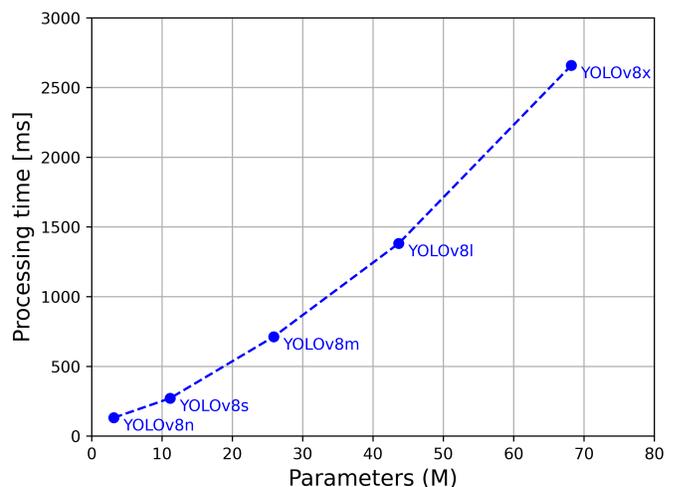


Fig. 4: The single-image, onboard HL2 processing times depending on the chosen YOLOv8 model size for a fixed image size 224×224 pixels.

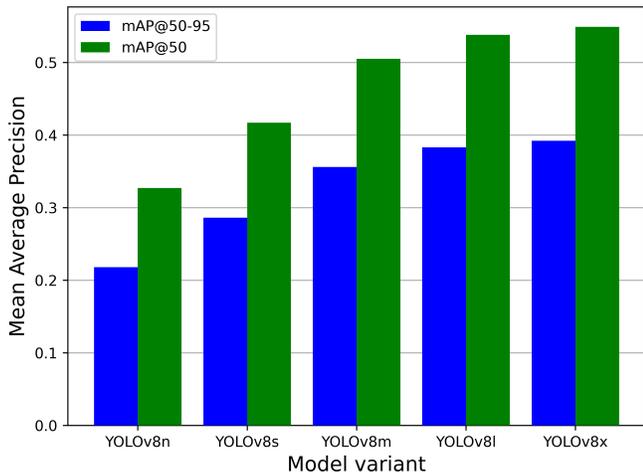


Fig. 5: The measured performance of object detection as mAP values depending on the network size for YOLOv8 starting from the smallest network (YOLOv8n) to the largest network (YOLOv8x) for a fixed image size 224×224 pixels.

which is the average precision for all object classes measured at a selected threshold A . $mAP@A$ indicates the performance when at least a $A\%$ overlap between the bounding box from detection and ground truth bounding box (Intersection over Union – IoU) is required to assume that the object was correctly recognized. The performance of different detection model configurations is presented in Fig. 5 for $mAP@50$ and $mAP@50-95$ averaging the performance over a range of IoU thresholds. The obtained results suggest that a significant drop in performance should be expected when using smaller models.

B. Object detection depending on the input image size

Unfortunately, even with the smallest model, i.e., YOLOv8n, we ought to seek further improvements to achieve real-time performance dictated by the best immersive experience for AR headset users.

Apart from the size of the network, the other possibility is to reduce the input image size as it directly impacts the inference times. The results we obtained for varying image input sizes are presented in Fig. 6.

The obtained relation between processing times and an input image size shows that processing times scale almost quadratically with the side length of an image (i.e. linearly with the number of pixels). Based on this observation, we can see that it is possible to obtain object detection results in less than 100 [ms] when using an image size of 160×160 pixels. Using smaller input image sizes might impact the achieved performance of the algorithm. We show the influence of the input image size on the mAP of the algorithms in Fig. 7.

C. Choosing the best model based on processing time budget

We might also have a greater processing time budget depending on the application. In these scenarios, we wanted to quantify if using a larger network for inference is more

beneficial or, rather, increasing the backbone size to improve the network’s detection performance makes sense.

The obtained results suggest that for any application with an inference time budget below 400 [ms], it is beneficial to use YOLOv8n while tuning the image size to fit the budget requirements. Compared to YOLOv8n at the same processing time, larger networks perform worse as they need to use smaller images. For processing times thresholds greater than 400 [ms], we should choose YOLOv8s as it offers better performance than YOLOv8n despite smaller input image sizes than YOLOv8n while outperforming all larger backbones in the analyzed processing time interval. The presented conclusions are drawn based on the obtained performance on all objects in the COCO dataset, which might not hold equally for particular object classes.

D. Model performance analysis for AR applications

Flustered by the reduced performance of the YOLOv8n with a small input image size of 160×160 , we conducted a series of real-world test experiments. We focused on an example object, i.e. a smartphone, detected from 1 [m] up to 4 [m] with an object observed from 20 different viewing angles at each distance as presented in Fig. 9. We selected these distances, arguing that reaching up to 2 [m] is crucial for AR interaction due to the maximum extent of human arms and hand-held tools. We frequently encounter such situations when dealing with shop floor tasks [4] or device repair [5], which require close vicinity, i.e., arms-stretch distance from non-digital asset users are interacting with.

We measured the performance at each distance as a recall, i.e., a ratio of the number of cases when the smartphone was properly detected to the number of images. The results obtained for different confidence thresholds are presented

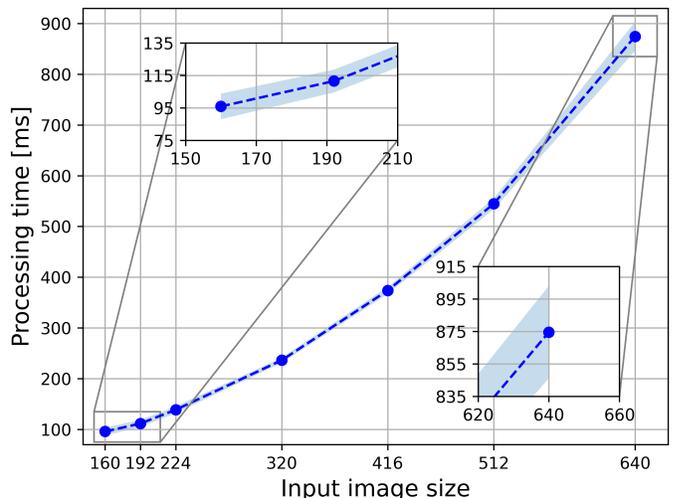


Fig. 6: The total processing time for object detection for YOLOv8n model with different input image sizes. The light blue interval shows the standard deviation of the performed measurements.

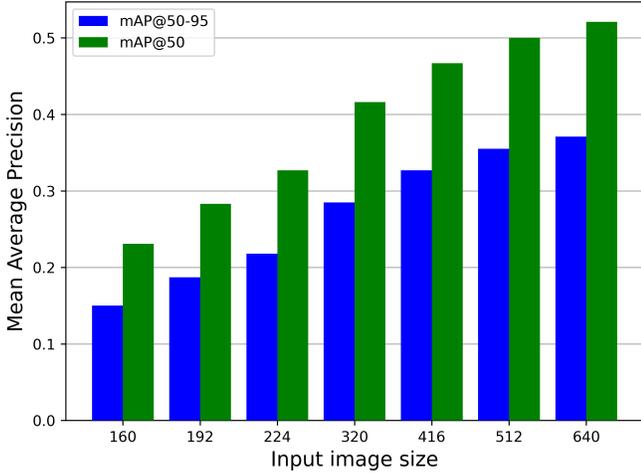


Fig. 7: The measured performance of object detection using YOLOv8n model with different input image sizes.

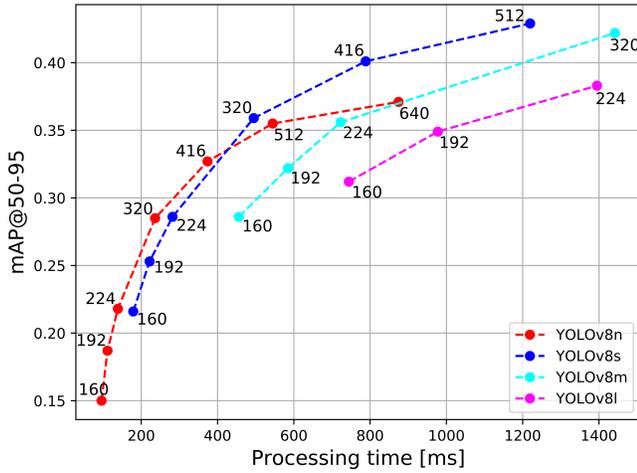


Fig. 8: Comparison of mAP and inference time for different sizes of YOLOv8 models

in Fig. 10. These outcomes suggest that the proposed configuration offering inference results in less than 100 [ms] can still detect all object (i.e. smartphone) instances if we focus on distances closer than 2.5 [m]. We believe that such performance fulfills the requirements for most AR use case scenarios [4], [5].

VI. ABLATION STUDY

The goal of the following section is to understand further the limitations of the YOLOv8 with 160×160 input image size and propose a solution that could further speed up the processing capabilities depending on the application.

A. Processing time analysis

As a first step, we measured the time spent to prepare the data (i.e., preprocessing), make the actual bounding box prediction (i.e., inference), and the time necessary to analyze

the obtained predictions (i.e., postprocessing). The results are summarized in Tab. I and indicate the most time spent performing the inference. It shows that postprocessing, even though the NMS step is not performed on the GPU, is not a limiting factor.

operation	processing time	
	mean [ms]	std [ms]
preprocessing	1.97	1.49
inference	89.84	7.45
postprocessing	4.06	1.17

TABLE I: The analysis of the total processing time when using the YOLOv8n model and 160×160 images size input revealing that the most time is spent doing the core neural model inference

Further analysis of the inference time using the Unity profiler tool revealed that the time spent to copy the data for inference is negligible, taking less than 1% of the overall inference time. Therefore, further improvements should be sought in the inference itself.

B. Testing different model processing backends

The processing time analysis indicates that we should improve the inference time. One possibility is to explore the inference backends available in the Barracuda package [39]. The backend choice determines whether the neural network will be run on GPU or CPU and what kind of implementation will be used. The results received for different backends are presented in Tab. II.

device	backend	inference time	
		mean [ms]	std [ms]
GPU	Compute	103.64	6.68
	ComputeRef	174.65	6.65
	Compute Precompiled	89.84	7.45
	CSharp	344.83	23.31
CPU	CSharpBurst	216.11	25.62

TABLE II: Inference times for different backend selections on HL2 using YOLOv8n and image input size of 160×160

The fastest inference times were obtained for ComputePrecompiled backend, followed by the Compute backend, which did not improve further results. We were unable to execute an inference with remaining CSharpRef and PixelShader backends.

Another way to improve the neural network inference performance is the usage of quantization, either with reduced float precision (FP16) or integer (INT8). Unfortunately, the Barracuda library does not support the FP16 nor INT8 quantization in its current implementation. FP16 usually offers a significant speed-up compared to the full float implementations as already proven on other computing platforms [40], [41].

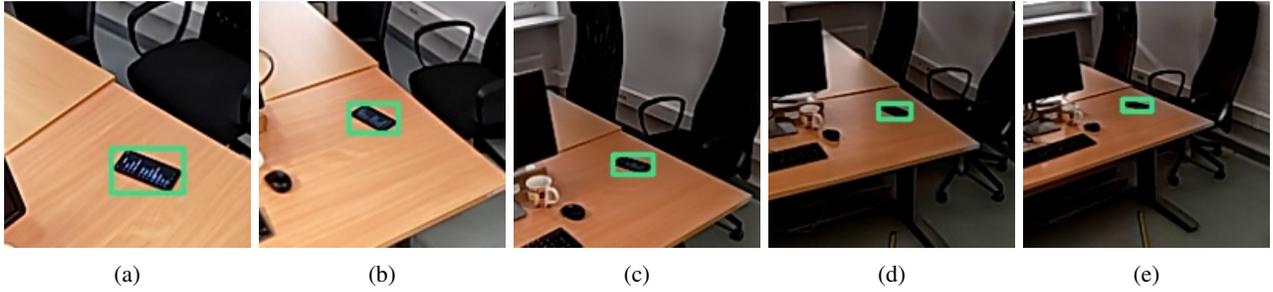


Fig. 9: The performance of the proposed real-time YOLOv8n when using 160×160 input image size. The network detects objects reliably from (a) 1 [m], (b) 1.5 [m], (c) 2.0 [m], (d) and 2.5 [m] distance, with decaying results up to (e) 3.0 [m].

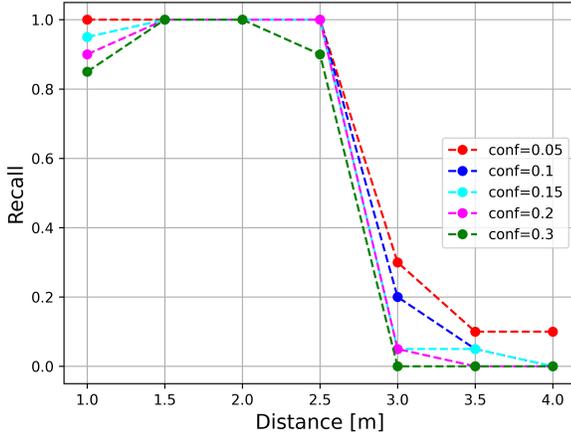


Fig. 10: Recall depending on the distance to the smartphone for different confidence thresholds based on the real-time YOLOv8n run on 160×160 input image sizes. The obtained results suggest that the proposed real-time configuration of YOLOv8n on HL2 can be sufficient for most AR applications.

C. Dealing with non-square image input

One commonly used approach to dealing with larger images and smaller objects assumes sliding the object detector over the whole image [42]. Consequently, we wanted to verify if it is beneficial for HL2 to divide the input image into smaller sub-images while increasing the batch size of the passed data input. Here, another gain is that the view from HL2 is not square, as we have a greater horizontal field of view than the vertical one. Our experiment compared two approaches: the network with 320×320 input image size and a network input of 320×160 pixels divided into two sub-images, each of size 160×160 pixels, passed as a batch size two for network inference. The results are summarized in Tab. III.

The total processing time for two side-stacked images is almost half the time required to process the square-size single image. These results confirm that the inference time linearly depends on the number of image pixels and that the proposed batch approach allows the process of non-square image inputs to preserve the natural aspect sizes of objects. Retraining the model for non-square inputs would require more effort than the proposed sliding window approach.

image size	detection time	
	mean [ms]	std [ms]
2 x 160 x 160	133.64	3.94
1 x 320 x 320	236.31	4.98

TABLE III: Inference time comparison between the YOLOv8 processing the 320×320 input image size compared to the two side-stacked images of 160×160 input images passed as batch size two.

VII. CONCLUSIONS

This paper presents the steps required for real-time object detection using the state-of-the-art YOLOv8 network model on the Microsoft HoloLens 2 edge computing platform.

We believe that the ability to run advanced ML such as YOLOv8 algorithms directly on an HMD will become necessary for the emerging edge-based virtual reality applications [30] and the *Metaverse* concept [43]. For a wide range of practical applications, particularly educational [44] and medical [45], the VR and AR technologies need to converge [46], providing the users with an ability to seamlessly interact with both real and virtual elements of the surroundings.

To that end, our experiments show the universal path that can be taken to ensure real-time operation by finding the best trade-off between the neural network model size and the input image size. In the presented case of an AR headset, we are forced to reduce the input image sizes to 160×160 pixels, still obtaining satisfactory results from the perspective of AR applications mainly within the 2.5 [m] range for an object of interest detection. Such distance span is well-aligned with the typical operational range of AR use case scenarios [4], [7].

The analysis of processing times (see Tab. I) reveals that further improvements should be sought in the inference itself, as preprocessing, postprocessing, or data copying take little time. Beyond that, to boost the number of possible use cases of the proposed solution, we also show that slicing a view into multiple images processed in a single batch can be a sensible approach when we are dealing with situations where wider FoV is used.

In our future work, we plan to tackle the missing FP16 support to reduce inference times further. In addition, we will also incorporate tracking into object detection to extend the possible applications of the presented framework.

REFERENCES

- [1] T. P. Caudell and D. W. Mizell, "Augmented reality: an application of heads-up display technology to manual manufacturing processes," 25th Hawaii Int. Conf. on System Sciences, Kauai, HI, USA, 1992, vol. 2, pp. 659–669.
- [2] J. Y. Mambu, E. Anderson, A. Wahyudi, G. Keyeh and B. Dajoh, "Blind Reader: An Object Identification Mobile-based Application for the Blind using Augmented Reality Detection," 2019 Int. Conf. on Cybernetics and Intelligent System, Denpasar, Indonesia, 2019, pp. 138-141.
- [3] S. K. Tadeja, D. Janik, P. Stachura, M. Tomecki, K. Książczak and K. Walas, "MARS: A Cross-Platform Mobile AR System for Remote Collaborative Instruction and Installation Support using Digital Twins," 2022 IEEE Conf. on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), Christchurch, New Zealand, 2022, pp. 373-380.
- [4] S. K. Tadeja, D. Janik, P. Stachura, M. Tomecki and K. Walas, "Design of ARQ: An Augmented Reality System for Assembly Training Enhanced with QR-Tagging and 3D Engineering Asset Model," 2022 IEEE Conf. on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), Christchurch, New Zealand, 2022, pp. 466-471.
- [5] L. O. Solari Bozzi, K. Samson, S. Tadeja, S. Pattinson, and T. Bohné, "Towards Augmented Reality Guiding Systems: An Engineering Design of an Immersive System for Complex 3D Printing Repair Process," 2023 IEEE Conf. on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), Shanghai, China, 2023, pp. 384-389.
- [6] A. Seifert and A. Schlomann, "The Use of Virtual and Augmented Reality by Older Adults: Potentials and Challenges", *Front. Virtual Real.* 2:639718. 2021.
- [7] A. Syberfeldt, M. Holm, O. Danielsson, L. Wang, and R. Brewster, "Support systems on the industrial shop-floors of the future – operators' perspective on augmented reality", *CIRP*, 2016, 44, pp. 108–113.
- [8] Microsoft, HoloLens 2, [online] <https://www.microsoft.com/en-us/HoloLens/hardware>
- [9] M. A. Latif Sarker and D. Seog Han, "Human-Centric Autonomous Driving Based on a Two-Stage Machine Learning Algorithm," 2022 27th Asia Pacific Conf. on Communications (APCC), Jeju Island, Korea, Republic of, 2022, pp. 334-3335.
- [10] A. Seeliger, R. Weibel, S. Feuerriegel, "Context-Adaptive Visual Cues for Safe Navigation in Augmented Reality Using Machine Learning", *Int. Journal of Human-Computer Interaction*, 1044-7318, 2022.
- [11] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788.
- [12] K. Roszyk, K.; Nowicki, M.R.; Skrzypczyński, P. "Adopting the YOLOv4 Architecture for Low-Latency Multispectral Pedestrian Detection in Autonomous Driving", *Sensors*, 22, 1082, 2022.
- [13] A. Farasin, F. Peciarolo, M. Grangetto, E. Gianaria, P. Garza, "Real-time object detection and tracking in mixed reality using microsoft HoloLens", 15th Int. Joint Conf. on Computer Vision, Imaging and Computer Graphics Theory and Applications, 2020, Vol. 4, 165–172.
- [14] R. Goka et al., "Development of Tomato Harvest Support System Using Mixed Reality Head Mounted Display," 2022 IEEE 4th Global Conf. on Life Sciences and Technologies, Osaka, Japan, 2022, pp. 167–169.
- [15] M. Eckert, M. Blex, C. M. Friedrich, "Object detection featuring 3D audio localization for Microsoft HoloLens", in Proc. 11th Int. Joint Conf. on Biomedical Eng. Sys. and Techn., 2018, Vol. 5, pp. 555–561.
- [16] R. Anderson, J. Toledo and H. ElAarag, "Feasibility Study on the Utilization of Microsoft HoloLens to Increase Driving Conditions Awareness," 2019 SoutheastCon, Huntsville, AL, USA, 2019, pp. 1-8.
- [17] Microsoft, Azure Custom Vision, [online] <https://learn.microsoft.com/en-us/azure/cognitive-services/custom-vision-service/>
- [18] A. Ibrahim, J. B. Lanier, B. Huynh, J. O'Donovan, T. Höllerer, "Real-time Object Recognition on the Microsoft HoloLens", technical report, UC Santa Barbara, 2017.
- [19] H. Bahri, D. Krčmařík, J. Kočí, "Accurate object detection system on HoloLens using yolo algorithm", in 2019 Int. Conf. on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO), 2018, pp. 219–224.
- [20] Y. Ghasemi, H. Jeong, S. H. Choi, K.-B. Park, J. Y. Lee, "Deep learning-based object detection in augmented reality: A systematic review", *Computers in Industry*, 139, 103661, 2022.
- [21] K. Malek, A. Mohammadkhorasani, F. Moreu, "Methodology to integrate augmented reality and pattern recognition for crack detection", *Computer-Aided Civil and Infrastructure Engineering* (2022).
- [22] PTC, Vuforia Engine Library, [online] <https://library.vuforia.com/>
- [23] EasyAR Library, [online] <https://www.easyar.com/>
- [24] Y. Guan, X. Hou, N. Wu, B. Han, and T. Han, "DeepMix: mobility-aware, lightweight, and hybrid 3D object detection for headsets", 20th Int. Conf. on Mobile Systems, Applications and Services, 2022.
- [25] K. Chen, T. Li, H.-S. Kim, D. E. Culler, and R. H. Katz. "MARVEL: Enabling Mobile Augmented Reality with Low Energy and Low Latency". In Proc. of ACM Conference on Embedded Networked Sensor Systems (SenSys), 2018.
- [26] L. Liu, H. Li, and M. Gruteser. "Edge Assisted Real-time Object Detection for Mobile Augmented Reality", 25th Annual Int. Conf. on Mobile Computing and Networking (MobiCom), 2019.
- [27] ONNX Runtime developers, ONNX Runtime, [online] <https://onnxruntime.ai/>
- [28] Unity, Barracuda Library, [online] <https://docs.unity3d.com/>
- [29] Unity, Unity Game Engine, [online] <https://unity.com/>
- [30] M. Xu et al., "A Full Dive Into Realizing the Edge-Enabled Metaverse: Visions, Enabling Technologies, and Challenges," in *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 656–700, 2023.
- [31] R. Padilla, S. L. Netto and E. A. B. da Silva, "A Survey on Performance Metrics for Object-Detection Algorithms," 2020 Int. Conf. on Systems, Signals and Image Processing, Niteroi, Brazil, 2020, pp. 237-242.
- [32] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, B. Lee, "A survey of modern deep learning based object detection models", *Digital Signal Processing*, pp. 103514, 2022.
- [33] Z. Zou, K. Chen, Z. Shi, Y. Guo and J. Ye, "Object Detection in 20 Years: A Survey," in Proc. of the IEEE, vol. 111, no. 3, pp. 257-276, March 2023.
- [34] PyTorch Foundation, PyTorch, [online] <https://pytorch.org/>
- [35] EnoxSoftware, HoloLensCameraStream, Unity plugin, [online] <https://github.com/EnoxSoftware/HoloLensCameraStream>
- [36] D. Lowe, (2004). Distinctive Image Features from Scale-Invariant Key-points. *Int. Journal of Computer Vision*. 60. 91.
- [37] L. Tsung-Yi, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, et al. (2014). "Microsoft COCO: Common Objects in Context", *CoRR*, abs/1405.0312.
- [38] I. Athanasiadis, P. Mousoulitis, and L. Petrou, Loukas, "A Framework of Transfer Learning in Object Detection for Embedded Systems", *arXiv:1811.04863*, 2018.
- [39] Unity, Unity Barracuda Backend, [online] <https://docs.unity3d.com/Packages/com.unity.barracuda@3.0/api/Unity.Barracuda.WorkerFactory.Type.html>
- [40] N. -M. Ho and W. -F. Wong, "Exploiting half precision arithmetic in Nvidia GPUs," 2017 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 2017, pp. 1-7.
- [41] A. Haidar, S. Tomov, J. Dongarra and N. J. Higham, "Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers," SC18: Int. Conf. for HPC, Networking, Storage and Analysis, Dallas, TX, USA, 2018, pp. 603-613.
- [42] J. Lee, J. Bang and S. -I. Yang, "Object detection with sliding window in images including multiple similar objects," 2017 Int. Conf. on Information and Communication Technology Convergence (ICTC), Jeju, Korea (South), 2017, pp. 803-806.
- [43] T. Ribeiro, et al., "Virtual Reality Solutions Employing Artificial Intelligence Methods: A Systematic Literature Review", *ACM Comput. Surv.* 55, 10, Article 214, 2023.
- [44] M. Igras-Cybulska, D. Hekiert, A. Cybulski, S. Tadeja, et al. (2023). "Towards Multimodal VR Trainer of Voice Emission and Public Speaking: Work-in-Progress," 2023 IEEE Conf. on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), Shanghai, China, 2023.
- [45] P. Zikas et al., "MAGES 4.0: Accelerating the World's Transition to VR Training and Democratizing the Authoring of the Medical Metaverse," in *IEEE CG&A*, vol. 43, no. 2, pp. 43-56, 2023.
- [46] K. Li, et al., "When Internet of Things Meets Metaverse: Convergence of Physical and Cyber Worlds," in *IEEE Internet of Things Journal*, vol. 10, no. 5, pp. 4148–4173, 2023.