Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Edge-XReal: Distributed Holistic Scene Understanding for Edge XR Devices

Gijs Fiten

gijs.fiten@kuleuven.be

KU Leuven

Sam Winant

KU Leuven

Jit Chatterjee

KU Leuven

Maria Torres Vega

KU Leuven

Research Article

Keywords: Holistic Scene Understanding, Distributed Systems, Extended Reality, Deep Learning, Head Mounted Displays

Posted Date: July 6th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9745238/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Edge-XReal: Distributed Holistic Scene Understanding for Edge XR Devices

Gijs Fiten, Sam Winant, Jit Chatterjee, Maria Torres Vega

FIIW, KU Leuven, Andreas Vesaliusstraat 13, Leuven, 3000, Belgium.

Corresponding author: gijs.fiten@kuleuven.be;

Abstract

The high levels of immersion and realism offered by Extended Reality (XR) rely on the quality and accuracy of the virtual scenes the user interacts with. While 3D Holistic Scene Understanding (HSU) facilitates this by automatically reconstructing 3D scene layouts, object meshes and object and camera poses, its implementation is currently hindered by the computational demands of the state-of-the-art deep learning models used. The hardware requirements for these algorithms exceed the on-board capabilities of any stand-alone Head Mounted Display (HMD) currently available, making the deployment of these HSU models on the edge infeasible. To address this gap, we present Edge-XReal, a distributed approach to bring intensive HSU algorithms to edge XR devices. A flexible software pipeline is proposed to provide fast processing and data transfer, minimising latency and power consumption on the edge without sacrificing accuracy and realism. Results show a 40% to 48% reduction in processing time compared to loading and running the algorithm whenever a computation is needed. Edge-XReal is agnostic to the underlying 2D detector and HSU pipeline, ensuring a future-proof solution that can support multiple concurrent users. All code is available on GitHub: https://github.com/GijsFiten/Edge-XReal/

Keywords: Holistic Scene Understanding, Distributed Systems, Extended Reality, Deep Learning, Head Mounted Displays

1 Introduction

Understanding and reconstructing the physical environment from image data is a fundamental challenge in Computer Vision (CV) and a crucial component of many emerging applications, such as Mixed Reality (MR) where an edge device needs to understand the environment it is in to provide the best, and safest, experience to the end user. Classical scene understanding techniques rely on images from different viewpoints to understand and reconstruct objects and the environment [1], or omit the reconstruction and only map and localise[2] in order to make this executable on edge devices. 3D Holistic Scene Understanding (HSU) on the other hand, uses just a single input image to fully reconstruct a 3D scene, recreating both the objects, the layout and the camera position [3–5]. This approach offers greater flexibility by eliminating the need for many images to be taken. However, since only one viewpoint is available, a significant amount of information must be inferred, which presents unique challenges in the process. Deep Learning (DL) excels at inferring this type of information, with recent advancements in DL enabling significant progress in the area of HSU. Solutions like Implicit3D [4] and 3D-Scene-Former [5] have shown significant progress in terms of accuracy.

However, the computational requirements of these state-of-the-art HSU models exceed the onboard capabilities of low-power standalone edge devices such as Head-Mounted Displays (HMDs). This limitation hinders the development of immersive and interactive eXtended Reality (XR) experiences that rely on near real-time HSU, mainly remote experiences where objects, such as pieces of furniture, are moved around dynamically.

To address this challenge, we propose Edge-XReal[1], a novel distributed architecture designed to offload computationally intensive DL tasks from the HMD to a more powerful server, while using the HMD for data capture and displaying the XR scenes. This approach allows low-power edge devices to access intensive DL techniques without compromising on performance or battery life. Edge-XReal enables near-real-time 3D HSU on XR devices by separating the data gathering and visualisation from the actual DL calculations, which are more suited to a high-power server environment.

This paper presents the design, implementation, and evaluation of Edge-XReal. The integration of various 2D object detection models is explored, including Faster-RCNN [6] with ResNet50 and MobileNet backbones and the transformer-based Detection Transformer (DETR) [7]. Their accuracy and computational requirements for the given task are assessed as current HSU methods do not take this step into account and assume the 2D object detection to be 100% accurate. The performance of the current state-of-the-art HSU models is investigated, Total3D [3], Implicit3D [4] and 3D-Scene-Former [5] and how Edge-XReal can drastically improve their performance in XR settings. Our evaluation focusses on fast execution times and low client-side energy consumption. The solution should be future-proof and flexible, as

[1] https://github.com/GijsFiten/Edge-XReal

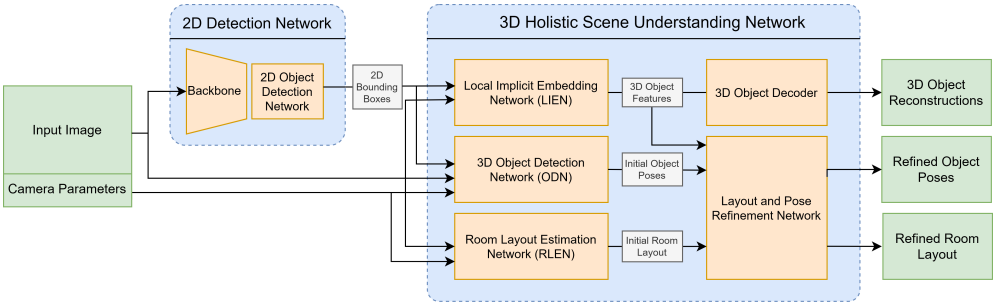


Fig. 1: Basic overview of a State of the Art 3D Holistic Scene Understanding pipeline, showing all substeps and intermediate data.

the pipeline maintains compatibility with any 2D detector-HSU combination, with server-side scalability for multiple clients as a secondary objective.

A Unity-based application was developed which visualises 3D scene reconstructions on a Meta Quest 3 headset to validate the effectiveness of the proposed approach. This Proof of Concept (POC) demonstrates the potential of Edge-XReal to expand XR experiences by enabling real-time, high-quality 3D scene understanding on low-power devices. Creating an architecture that meets these requirements will allow researchers and industry leaders to bring these complex DL techniques to the edge and make them more accessible and interactive. All of our code and data is available on GitHub.

The remainder of this paper is structured as follows: An in-depth analysis of the problem and the state-of-the-art is given in section 2. We will then outline our proposed method and its implementation in section 3.The results of this implementation can be found in section 4. The final conclusions can be found in section 5.

2 Related Work

HSU is a complex task that requires multiple CV and scene understanding steps to be executed. A general HSU pipeline, as seen in fig. 1, is implemented in most state-of-the-art solutions, such as Implicit3D [4], Total3D [3], and 3D-Scene-Former [5]. These solutions typically operate in the following order: first, all objects in the image are detected in 2D. Then, these 2D bounding boxes are converted to 3D bounding boxes by estimating a general 3D position, orientation, and shape for each object. The placement of each object is then fine-tuned in relation to the others using the estimated room layout, 3D poses, and implicit 3D shape representations. Finally, when all positions and orientations are finalized, the implicit 3D representation is converted into a mesh for each object. The remainder of this section details the state-of-the-art for each of these processes, namely: 2D object detection, the specific HSU pipeline substeps (layout estimation and mesh generation), and the datasets used for evaluation.

2.1 Object Detection

The first step in HSU is to detect the objects in the scene. This is a challenge which has been widely studied over the past decade, with DL CV models arising based on the Convolutional Neural Network (CNN) architecture [6, 8, 9]. The most popular models are You Only Look Once (YOLO) [8] (currently at version 11 [10]), Faster R-CNN [6], and MobileNetv3 [11] for low-power devices. These models have been instrumental in the development of HSU as they provide a fast and accurate way to detect the 2D bounding boxes of objects in an image. This part is usually left out of the HSU pipeline, and assumed to be 100% accurate [3–5]. These detectors and their feature extractors are discussed below.

ResNet-50 [9] is a deep convolutional neural network designed to efficiently extract hierarchical visual features from images. Its architecture employs residual blocks with skip connections, enabling stable training of very deep layers by mitigating vanishing gradient issues. As a feature extractor, it processes input images through successive stages that progressively downsample spatial resolution while expanding channel depth, capturing low-level edges, mid-level textures, and high-level semantic patterns. This approach has been widely adopted in recent years and has shown great detection results. [12]

MobileNetV3 [11] is a CNN architecture designed for efficient performance on mobile and embedded devices. It combines advancements from MobileNetV2 [13] and Neural Architecture Search (NAS) [14] to achieve a balance between low computational complexity and high accuracy. This ensures efficient processing on devices with limited resources, making it ideal for real-time applications.

Faster R-CNN [6] is a two-stage object detection framework that first identifies regions of interest and then classifies them. It begins by extracting hierarchical features from the input image using a convolutional backbone network. ResNet-50 and MobilenetV3 are two commonly used backbones. These features are fed into a Region Proposal Network (RPN), which generates candidate object regions. The proposed regions are pooled into fixed-size feature maps for consistent processing. Finally, a detection head classifies each region into specific object categories and refines their bounding box coordinates for precise localization.

DETR [7] is an object detection architecture which replaces the traditional region proposal and anchor-based mechanisms with a transformer architecture [15]. DETR, unlike Faster-RCNN, directly predicts bounding boxes and labels using the features extracted by a backbone. The ability of attention mechanisms to capture long range dependencies should allow it to make more contextually aware predictions.

Current state-of-the-art HSU pipelines are evaluated using the ground truth bounding boxes of the objects in the scene, which are given in the datasets these models are trained on [16]. To avoid any reliance on ground truth data, and make our

pipeline as flexible as possible, we add the 2D object detection task into our proposed method.

2.2 HSU Pipeline: Layout Estimation, Object Detection, and Mesh Generation

Once all objects in the input image have been identified by the 2D detection algorithm, the pipeline proceeds to the HSU stage.

Room Layout Estimation: Previous works [3–5, 17] first estimate the room bounds and camera position directly from the input image. The image is fed through a Room Layout Estimation Network (RLEN) consisting of a Residual Networks (ResNet) based encoder [9], which extracts features from the image. These features are then passed through a Multi-Layer Perceptron (MLP) to extract the estimated camera position and orientation, and the room 3D bounding box in one pass. This information is stored in a feature vector to provide global context for later use.

3D Object Detection: While the room layout bounds the scene, individual objects must also be localized in 3D. The input image is split into several patches, each corresponding to an object detected by the 2D Object detector. These patches are then passed through a ResNet based encoder [9] and a MLP to obtain the 3D bounding box of each object. This 3D Object Detection Network (ODN) will generate a good first guess as to where the object will be placed in the 3D space, this vector containing the location and pose estimation is again stored for later use.

Implicit Shape Representation: Beyond just bounding boxes, the actual geometric shape of these objects must be inferred. 3D objects can be represented in many ways, commonly they are represented as a point cloud, a mesh, or a voxel grid. None of these representations are ideal for DL as they are indefinite in size and not easily implemented in Machine Learning (ML) pipelines. Recently, the Signed Distance Function (SDF) [18] has gained traction as a way to encode the 3D object as the zero level set of a deep implicit function. This allows for the object to be represented by a latent vector, which can be passed through a MLP, along with 3D coordinates to obtain a grid of values whose sign depends on if they are located inside or outside the object. This grid of points can be converted into a mesh by using marching cubes [19] or Deep Marching Tetrahedra (DMTet) [20]. This method has been extended to Local Deep Implicit Functions (LDIF) [21] to generalize to novel image shapes and keep their complexity intact. The Local Implicit Embedding Network (LIEN) will extract these features, which will later be used to generate the object meshes.

Layout and Pose Refinement: With the room layout, object locations, and shape embeddings established, the pipeline must ensure the objects physically make sense together in the scene. The first wave of HSU pipelines placed the objects they detected directly where the initial ODN placed them in the scene [3, 22]. More recent implementations have added a refinement stage to fine-tune the placement of the objects in the scene. Implicit3D [4] introduced a Scene-Graph Neural Network (SGNN) which

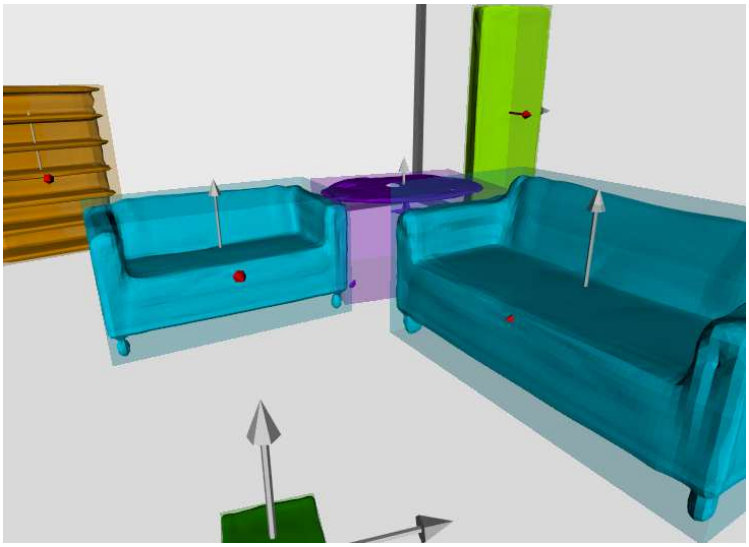


Fig. 2: Demo of the Implicit3D [4] software, displaying the generated object meshes in the scene from the perspective of the camera as detected by Implicit3D.

was fed the outputs of the RLEN, ODN and LIEN to have a full context aware refinement stage to allow the network to create more plausible object placements. More recently, 3D-Scene-Former [5] replaced this SGNN with a transformer [15] based encoder-decoder to improve this refinement stage even further.

Mesh Generation: Finally, once the objects are accurately placed within the scene, their visual models must be generated. The local implicit embeddings generated by the LIEN are fed through LDIF [21] to generate high quality 3D meshes, which can be rendered with modern graphics pipelines such as Unity or Unreal Engine. Multiple techniques exist to convert this implicit SDF representation of a 3D shape to a discretized mesh representation. Most notably, the Marching Cubes algorithm [19] has been used for many years to generate object meshes, such as the ones visualized in fig. 2, but recent improvements such as DMTet [20] promise to be more efficient and accurate when representing complex shapes with sharp features.

2.3 SUN-RGBD

The current state-of-the-art HSU algorithms have been trained on the SUN-RGBD dataset. SUN-RGBD is a real-life dataset created by Song et al. [16] to facilitate high level HSU using RGBD sensors. The dataset contains 10,335 Red Green Blue Depth (RGBD) photos of indoor scenes with a ground truth for objects 3D bounding boxes, 2D bounding boxes, centroid and scale. A reworked version of this dataset was used, containing only the Red Green Blue (RGB) images, the object labels and the 2D object bounding boxes to train our 2D Object Detector.

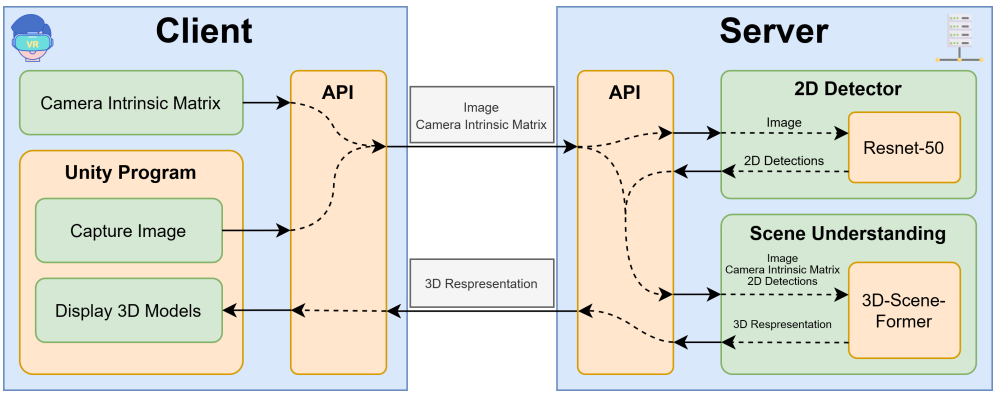


Fig. 3: Our Distributed architecture, showing the flow of data between the sub-tasks and the data transmitted between the client and the server

2.4 Limitations in the State-of-the-art and the proposed solution

Combining all the previous DL techniques into a single pipeline is a challenging task when it comes to implementing the network for inference on an edge device. Implementing these networks on a hardware-limited platform can be very challenging due to the large nature of them, ranging from 75M parameters in Total3DUnderstanding [3] to 92M parameters in 3D-Scene-Former [5], and from 19M parameters in small 2D object detectors like MobilenetV3 [11] to 41M parameters in larger detectors like ResNet50 [9] and DETR [7]. The meshes generated by HSU pipelines [3–5, 17, 22] lacked texture generation and focused primarily on mesh generation and object placement. This combined with the absence of readily available APIs for DL frameworks such as PyTorch [23] and TensorFlow [24] on commercially available Virtual Reality (VR) headsets and XR platforms and very limited Video RAM (VRAM) access on these devices makes it nearly impossible for these models to be natively accessible on an edge XR device. Currently, no framework exists to use HSU technology on edge devices.

To address these limitations, we propose a novel client-server framework designed to enable photorealistic HSU on resource-constrained edge devices. Unlike previous standalone implementations, our approach offloads the heavy computational load of the DL inference to a remote server, utilizing a lightweight client application solely for data transmission and visualization. This also allows us to connect multiple edge devices to the same server, allowing for a more efficient use of computational resources.

3 Edge-XReal System Design and Implementation

We propose an easy-to-use, distributed method to bring these complex DL based HSU models to edge XR devices. As shown in fig. 3, the intensive HSU tasks are offloaded to a separate server-like device, which can provide these predictions to multiple clients.

To demonstrate the capabilities of Edge-XReal and address the limited computational resources of current edge VR devices, a POC Unity MR program was developed. This section details the system architecture alongside the specific hardware and software implementations used to build this POC.

3.1 Server Architecture and Hardware

As the server is only used to perform HSU tasks, the models are loaded into VRAM when the server starts, and keep them there. This allows to significantly reduce the turnover time between HSU calculations. The Application Program Interface (API) will receive the client's image and camera intrinsic matrix, it will then perform the 2D detection to obtain 2D bounding and pass that result to the HSU pipeline (e.g. 3D-Scene-Former [5]). The results of the HSU pipeline that will be returned to the client can be big if there are a lot of detections or when the generated models are complex. These files (object meshes, object poses, room layout and camera position) are combined into a JavaScript Object Notation (JSON) file and zip it to minimize the network delay introduced by transferring all of these files. This can be especially needed if the client is a VR headset with limited connectivity, such as a Wi-Fi connection in a crowded area.

The server we used consists of a relatively old, but still capable system with an Intel I7–3770 Central Processing Unit (CPU), an NVIDIA GTX 1070 Graphics Processing Unit (GPU) with 8 GB of VRAM, 16 GB of DDR4 Random Access Memory (RAM), an NVMe SSD of 2 TB and 400 down / 50 up Mbps networking. Different stages in Edge-XReal use different parts of the server, so an upgrade to any component should lead to an increase in speed, with the GPU being the most critical component. Using a more recent GPU will speed up the pipeline considerably, but this implementation gives a solid baseline.

3.2 Client Application and Hardware

The interface is general enough, so any programmable device with an internet connection can send and receive the data from Edge-XReal. The client only needs to provide a way to take a picture with an on-board camera and send it together with its camera intrinsic matrix to the server. After which it will receive a JSON containing all object details and their meshes. The client then needs to show these to the user in an XR environment of their choosing. This approach allows this pipeline to be *language-agnostic* and flexible enough to use any 2D detector and HSU pipeline.

3.2.1 Camera Intrinsic Matrix

$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 855 & 0 & 710 \\ 0 & 852 & 730 \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

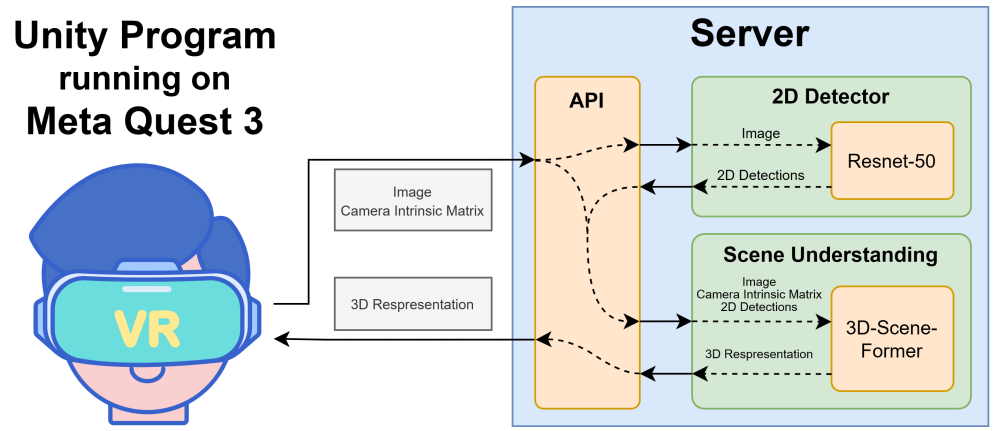


Fig. 4: Proof of Concept setup, showing the Meta Quest 3 as the Client device, communicating with the server over WiFi.

A camera can be modelled as a projection of the 3D physical space onto an image plane. This projection can be characterized by a matrix containing the focal length on both axes f_x and f_y, image skew s and sensor offsets x_0 and y_0, structured as in eq. (1) [25]. The HSU models need the camera's intrinsic value matrix, along with a single RGB image, to invert the image deformation caused by the camera. This matrix can usually be calculated from the metadata which is bundled with most images taken by a digital camera. This could however not be done in our specific implementation, so a short description is provided of a method to calculate it. The camera intrinsic matrix can be estimated by taking multiple images of a chequerboard pattern, which is a known size and ratio, and estimating the deformation from these images.

3.2.2 Unity program

We use Unity to provide an interactive demo that fully demonstrates the capabilities of Edge-XReal. Unity comes with a MR template project, on which our POC is built. The Unity framework uses C# scripts, so a set of scripts was created which collect and refactor all the data necessary for Edge-XReal and then pass that to our server API. All the functionality has been created to make an API call to our remote server, so these data are sent over the internet and receive a JSON array with all the information that was generated by the HSU pipeline of our choosing. The received data contains a variety of formats that are converted to data types which are compatible with Unity's C# dialect.

3.2.3 Meta Quest 3

For our POC, the Meta Quest 3 was used as shown in fig. 4. This headset comes with a relatively powerful onboard computer, capable of playing fully functional VR games. A device which has a camera that can take pictures and has a camera intrinsic matrix was needed. Meta, the company behind the Meta Quest 3, has recently given

Unity programs access to the on-board camera [2], but they do not provide the intrinsic parameters of the camera. We calculated and provide the intrinsic parameters for the meta quest 3 and list them in eq. (1), the Unity POC also provides an on-the-fly calibration method for convenience. From these values, more specifically f_x and f_y, the focal length of the front facing cameras of the Meta Quest 3 was calculated to be the same on the x and y-axis, concluding there is no unintentional or uneven distortion. The principal point offsets x_0 and y_0 tell us that the camera sensor is not centred on the principal point axis, and as such experiences an off-centre distortion pattern. The s value is 0, showing us that the x and y-axis of the image are perpendicular, leading to no shear distortion.

3.3 Experimental Setup

To evaluate the usability of the architecture when multiple clients connect to a single server, we tested two scenarios that may induce high server load. The first scenario simulates a single user issuing requests in rapid succession, while the second simulates multiple users sending requests concurrently. To explore the current limitations of Edge-XReal, the response time and error rate were analysedunder two test conditions: varying the interval between consecutive requests, and varying the number of simultaneous requests.

4 Results

The results of our proposed pipeline encompass four aspects, specifically, the pipeline should offer fast execution time, minimize energy use on the client side, remain compatible with any combination of 2D detector and HSU model and allow for multiple simultaneous connections to the server.

This section discusses each of these requirements where relevant. Firstly, section 4.1 discusses the performance of the different 2D Object Detectors we trained on the SUN-RGBD dataset [16]. Secondly, section 4.2 assess the performance of our Server-Client implementation, most notably the end-to-end response time. Finally, section 4.4 discusses the usability of our POC Unity application.

4.1 Performance of the 2D detectors

The performance was assessed of Faster-RCNN with a ResNet50 backbone [6, 9], Faster-RCNN with a MobileNetV3 backbone [11] and DETR [7] with a ResNet50 backbone using 2 metrics which are most indicative of their final performance in this pipeline. The Mean Average Precision (mAP) and the Mean Average Recall (mAR) were evaluated to assess the performance. These metrics represent the ratio of the correct predictions over the total number of predictions and total number of predictions over the expected number of predictions for each object class, respectively. These recall and precision values are usually calculated over a variety of scenarios, these include small, medium, and large bounding boxes, a varying number of detections on

[2] Meta Camera API Documentation

Table 1: Relevant parameters of the 2D object detection networks

	ResNet50	MobilenetV3	DETR
Model params	41276636	19071208	41280518
Train params	210125	210125	10794
Batch Size	6	12	12
Learning rate	5e-3	5e-3	1e-3
Weight decay	5e-4	5e-4	1e-4
Momentum	0.9	0.9	0.9
LR γ	0.1	0.5	0.5
LR step size	3	5	5

Table 2: Performance metrics (mean average precision and mean average recall) of 2D Object Detectors on the test split of SUN-RGBD. The best result is bold, the second best is underlined.

Backbone	Object Detector	mAP (%) ↑	mAR (%) ↑
ResNet-50	Faster-RCNN	**26.4**	<u>39.2</u>
MobileNetV3	Faster-RCNN	<u>15.8</u>	22.9
ResNet-50	DETR	7.8	**45.4**

an image and the required Intersection over Union (IoU), which is the overlap between the detected bounding box and the ground truth divided by the total area of the two bounding boxes. The mean of these values was calculated to get a more holistic view on the performance on each detector. The results can be found in table 2. We can see that Faster-RCNN with a ResNet backbone outperforms Faster-RCNN with a MobilenetV3 backbone in both precision and recall, but DETR seems to accurately detect most objects, as signified by the high mAR, but also detect many objects that are not present in the image, as shown by the low mAP. When investigating, many overlapping bounding boxes were found on objects that were detected. These bounding boxes are filtered by proximity to each other and relative overlap, and only keep the average bounding box coordinates if many closely related bounding boxes are found. This does negatively affect our mAR, but it prevents the 2D detector from detecting a single real life object as many instances of that same object. Based on the results, Faster-RCNN model with the Resnet50 backbone was selected for our use case, as it performs best in the most scenarios, including scenarios with bad lighting or occlusions.

When the bounding boxes which the 2D Object Detectors detect on the images in the test set are visualised, as seen in fig. 5, we can clearly see that there are still a few issues. The ground truth does not always capture all objects, Faster-RCNN with a MobilenetV3 backbone does a good job at detecting most objects as long as the scene is well lit, and the objects aren't occluded too much. DETR has a mixed performance, sometimes accurately finding all objects in a scene, sometimes not detecting a single

Ground Truth	Faster-RCNN ResNet-50	Faster-RCNN MobileNetV3	DETR ResNet-50

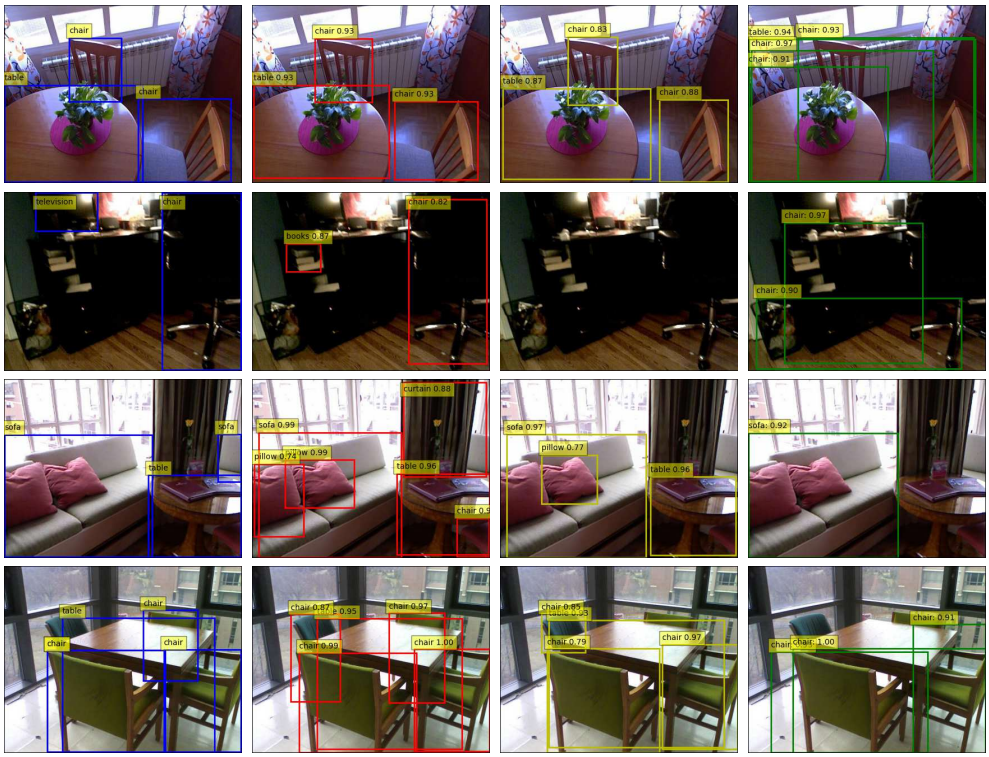


Fig. 5: Comparison of backbone architecture and 2D detector combinations under varying image conditions. Rows represent: (1) Well-lit scene with correct labels, (2) Poorly lit scene, (3) Scene with missing labels, and (4) Scene with occlusions and missing ground truth label.

object and sometimes completely misplacing a bounding box. Faster-RCNN with a ResNet50 backbone consistently gives the most reliable and accurate results and will be used in the final POC.

As we experimented with different training methods, the final training parameters are listed in table 1, so reproduction of our results is possible. An important observation is that many detections in the SUN-RGBD dataset are of category 'NULL', leading to detections labelled as incorrect, even when the 2D detectors give the correct results. This partially explains the relatively low mAP and mAR results seen in table 2.

It should be mentioned that the impact of an incorrect 2D detection is drastic on the HSU model, it mostly results in incoherent outputs. Resilience to incorrect or slightly

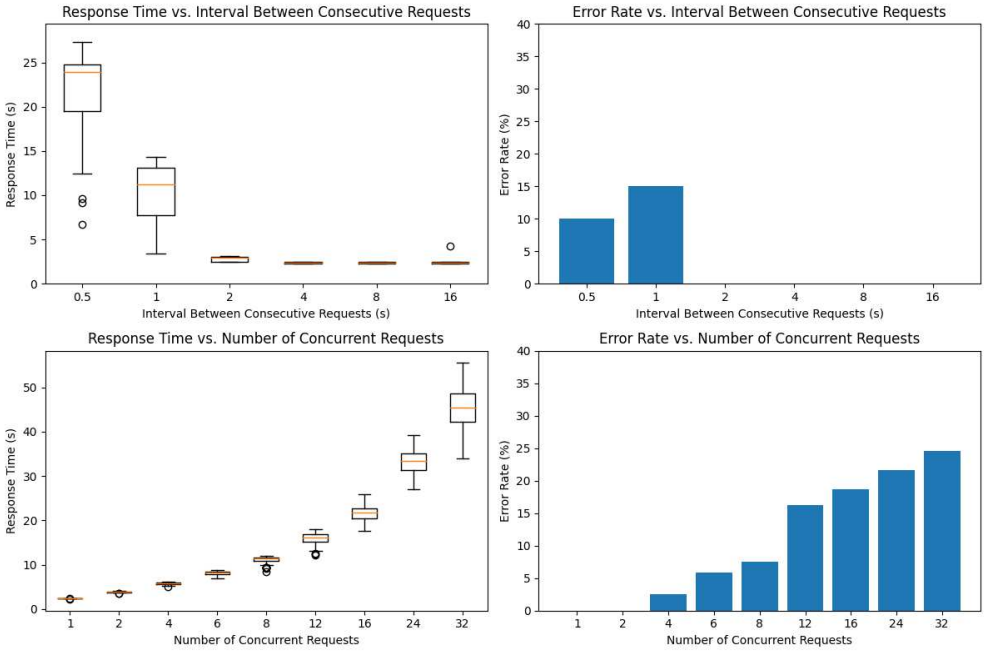


Fig. 6: Concurrent client evaluation results, showing box plots of the response times on the left and error rate on the right.

offset 2D detections is an important limitation of current HSU algorithms, and offers an interesting research opportunity.

4.2 Response Time Server

We are mostly interested in our end-to-end processing time to evaluate the performance of our Server-Client approach. To move towards a more real-time environment, this time should be minimised. To evaluate the timing, a benchmark test that evaluates 100 images and returns the average time for each step in Edge-XReal was ran. The final average time used can be seen in table 3. These test were run with the ResNet50 2D Object Detector and the Implicit3D HSU pipeline. To fully understand the advantages of our implementation, A benchmark on the Implicit3D pipeline without our server-client structure was also performed. The same test setup was used for this benchmark and the results can be found in table 3. It is clear that the HSU takes the most amount of time in this pipeline, followed by the networking aspect. This consists of receiving all the data from the Meta Quest 3 and sending the output back. This time has been greatly reduced by using GZIP to compress anything that should be sent to the Meta Quest 3, the compression takes around 400ms, which is well worth it as it saves 1 second in transfer time on our hardware, this tradeoff will depend on the network speed and the compression speed and should thus be evaluated on a case-by-case basis.

Table 3: Comparison of time spent on different sub-tasks in HSU using Total3D [3], Implicit3D [4] and 3D-Scene-Former [5], all ran locally and through Edge-XReal (abbreviated to EXR for conciseness). Averaged over 100 prediction passes

Description	Total3D		Implicit3D		3D-Scene-Former	
All times in ms	On-Demand	EXR	On-Demand	EXR	On-Demand	EXR
Preprocessing Data	55	42	48	61	51	62
Loading HSU Model	1295		1562		1589	
Loading 2D Object Detector	2652		2320		2556	
2D Bounding Box Detection	138	135	132	130	128	134
HSU Model Prediction	1295	1352	2325	2259	2653	2704
Saving Files	341		616		624	
Zipping Files		385		415		442
Networking		1101		1205		1182
Total	5776	3015	7003	4070	7601	4524
Improvement in %		47.8%		41.9%		40.5%

4.3 Concurrent client evaluation

The system's response time and error rate was evaluated under two conditions: consecutive requests issued at decreasing time intervals, and an increasing number of concurrent requests. The results indicate that the system maintains stable performance when the interval between requests is at least 2 seconds. In real world usage, where a client would recalculate the room layout approximately once per minute, this corresponds to supporting up to 30 concurrent users without significant degradation, assuming the requests are being received in a spread out manner.

However, the error rate increases noticeably when four or more requests are issued simultaneously, indicating a limit to the server's ability to handle bursts of concurrent activity. We recommend that better queueing strategies are used to avoid this problem. The detailed response time distributions and error rates for both experiments are shown in fig. 6.

4.4 VR Proof of Concept

The POC made in Unity to visualize our work cannot be easily evaluated objectively as the main goal is to prove that it is possible for a low-power VR headset to use the capabilities of an external server to gain access to large DL models which would be beyond the normal compute capabilities of the VR device. Our Unity program shows, after 4 seconds of waiting time on average, the results of Edge-XReal in a MR environment. It better shows the shortcomings in terms of object placement compared to the 2D renderings on a monitor due to the interactivity and full 3D view with depth perception. When the models have been placed in the MR environment, as shown in fig. 7, the user can walk around the model and visually inspect the accuracy of the placements, orientation of the objects and the quality of the mesh. This type of

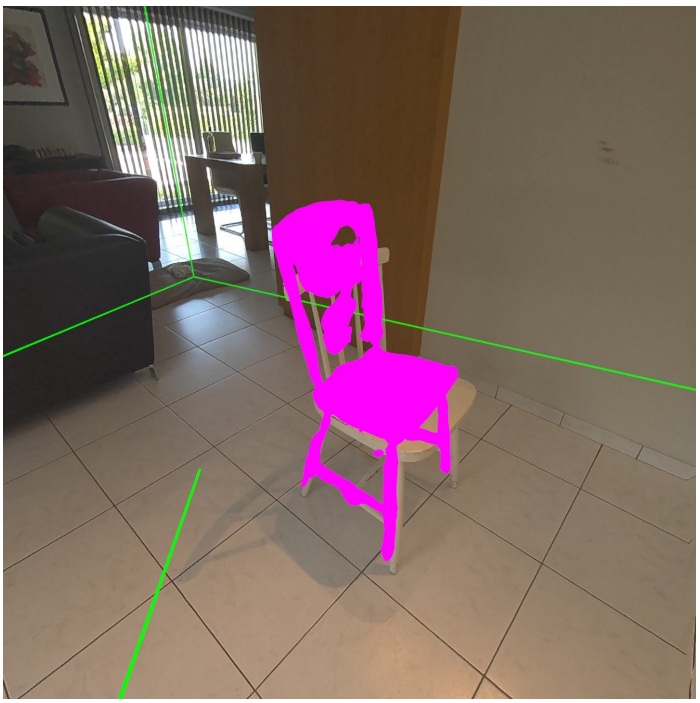


Fig. 7: Screenshot of the Unity Program showing a successful detection, overlaid on the camera view for an Augmented Reality (AR) experience.

demo can be a great tool for researchers in this area to better visually inspect their work.

5 Conclusion

In this work, Edge-XReal was presented, a novel distributed architecture for 3D Holistic Scene Understanding (HSU) specifically designed for use in low-power extended reality platforms. Different state of the art 2D object detectors were assessed on their speed and accuracy. Our findings suggest that Faster-RCNN with a ResNet50 backbone offers the best balance between accuracy and speed. The state of the art in HSU pipelines was evaluated and we conclude that newer models keep requiring more computation to complete, further establishing the need for Edge-XReal to bring these complex models to edge XR devices. We successfully implemented Edge-XReal in Unity and demonstrated its ability to visualize 3D scene reconstructions in a Mixed Reality (MR) environment on the Meta Quest 3 headset, which was impossible before.

Edge-XReal offers several key advantages: Firstly, it reduces computational strain on VR headsets as computationally expensive Deep Learning (DL) tasks are offloaded to a centralized server, allowing low-power edge devices to access powerful DL models.

Secondly, Edge-XReal adds flexibility to be easily adapted to use different 2D detectors and HSU models to allow developers to stay up to date in this quickly evolving field. Finally, visual inspection in our Unity application allows researchers to visually inspect their work to gain a better subjective understanding of the outputs of their models in virtual reality.

6 Declarations

6.1 Ethical Approval

Not applicable.

6.2 Consent to Participate

Not applicable.

6.3 Consent to Publish

Not applicable.

6.4 Data Availability Statement

The code supporting the findings of this study is available on a GitHub repository with a README file containing links to download all model checkpoints and the POC Unity project through Google Drive. All testing code and figure generation code is listed in the GitHub repository.

6.5 Authors Contributions

This work is primarily based on the Master's thesis submitted by Gijs Fiten and Sam Winant in partial fulfilment of the requirements for the degree of Master of Science in Engineering Technology at KU Leuven under supervision of Prof. Maria Torres Vega and drs. Jit Chatterjee. The manuscript was further refined by Gijs Fiten during the early stages of his Ph.D. candidacy under the supervision of Prof. Maria Torres Vega.

- Conceptualisation: Gijs Fiten & Sam Winant
- Methodology: Gijs Fiten & Sam Winant
- Formal analysis and investigation: Gijs Fiten & Sam Winant
- Writing - original draft preparation: Gijs Fiten
- Writing - review and editing: Gijs Fiten, Jit Chatterjee & Maria Torres Vega
- Supervision: Maria Torres Vega

6.6 Funding

This research is partially funded by the internally funded project of KU Leuven 'AI-driven generation of 6DoF Extended Reality' (C14/25/109).

6.7 Competing Interests

Not applicable.

References

[1] Mikhail, E.M., Bethel, J.S., McGlone, J.C.: Introduction to Modern Photogrammetry. John Wiley & Sons (2001). Google-Books-ID: D4h8EAAAQBAJ

[2] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., Leonard, J.J.: Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age. IEEE Transactions on Robotics **32**(6), 1309–1332 (2016) https://doi.org/10.1109/TRO.2016.2624754 . arXiv:1606.05830 [cs]. Accessed 2026-01-26

[3] Nie, Y., Han, X., Guo, S., Zheng, Y., Chang, J., Zhang, J.J.: Total3DUnderstanding: Joint Layout, Object Pose and Mesh Reconstruction for Indoor Scenes from a Single Image. arXiv. arXiv:2002.12212 [cs] (2020). https://doi.org/10.48550/arXiv.2002.12212 . http://arxiv.org/abs/2002.12212 Accessed 2023-09-05

[4] Zhang, C., Cui, Z., Zhang, Y., Zeng, B., Pollefeys, M., Liu, S.: Holistic 3D Scene Understanding from a Single Image with Implicit Representation. arXiv. arXiv:2103.06422 [cs] (2021). https://doi.org/10.48550/arXiv.2103.06422 . http://arxiv.org/abs/2103.06422 Accessed 2023-09-05

[5] Chatterjee, J., Torres Vega, M.: 3D-Scene-Former: 3D scene generation from a single RGB image using Transformers. The Visual Computer (2024) https://doi.org/10.1007/s00371-024-03573-2

[6] Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv. arXiv:1506.01497 [cs] (2016). https://doi.org/10.48550/arXiv.1506.01497 . http://arxiv.org/abs/1506.01497 Accessed 2024-04-16

[7] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-End Object Detection with Transformers. arXiv. arXiv:2005.12872 [cs] (2020). http://arxiv.org/abs/2005.12872 Accessed 2024-04-10

[8] Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You Only Look Once: Unified, Real-Time Object Detection. arXiv. arXiv:1506.02640 [cs] (2016). https://doi.org/10.48550/arXiv.1506.02640 . http://arxiv.org/abs/1506.02640 Accessed 2024-09-26

[9] He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. arXiv. arXiv:1512.03385 [cs] version: 1 (2015). https://doi.org/10.48550/arXiv.1512.03385 . http://arxiv.org/abs/1512.03385 Accessed 2024-05-14

[10] Khanam, R., Hussain, M.: YOLOv11: An Overview of the Key Architectural Enhancements. arXiv. arXiv:2410.17725 [cs] (2024). https://doi.org/10.48550/arXiv.2410.17725 . http://arxiv.org/abs/2410.17725 Accessed 2025-06-18

[11] Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q.V., Adam, H.: Searching for MobileNetV3. arXiv. arXiv:1905.02244 [cs] (2019). https://doi.org/10.48550/arXiv.1905.02244 . http://arxiv.org/abs/1905.02244 Accessed 2024-09-26

[12] Khan, S., Naseer, M., Hayat, M., Zamir, S.W., Khan, F.S., Shah, M.: Transformers in Vision: A Survey. ACM Comput. Surv. **54**(10s), 200–120041 (2022) https://doi.org/10.1145/3505244 . Accessed 2025-07-15

[13] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C.: MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv. arXiv:1801.04381 [cs] version: 4 (2019). https://doi.org/10.48550/arXiv.1801.04381 . http://arxiv.org/abs/1801.04381 Accessed 2024-04-29

[14] Zoph, B., Le, Q.V.: Neural Architecture Search with Reinforcement Learning. arXiv. arXiv:1611.01578 [cs] (2017). https://doi.org/10.48550/arXiv.1611.01578 . http://arxiv.org/abs/1611.01578 Accessed 2024-09-27

[15] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention Is All You Need. arXiv. arXiv:1706.03762 [cs] (2023). https://doi.org/10.48550/arXiv.1706.03762 . http://arxiv.org/abs/1706.03762 Accessed 2024-04-15

[16] Song, S., Lichtenberg, S.P., Xiao, J.: SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite, pp. 567–576 (2015). https://openaccess.thecvf.com/content_cvpr_2015/html/Song_SUN_RGB-D_A_2015_CVPR_paper.html Accessed 2024-09-30

[17] Liu, H., Zheng, Y., Chen, G., Cui, S., Han, X.: Towards High-Fidelity Single-view Holistic Reconstruction of Indoor Scenes. arXiv. arXiv:2207.08656 [cs] (2022). https://doi.org/10.48550/arXiv.2207.08656 . http://arxiv.org/abs/2207.08656 Accessed 2023-09-05

[18] Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. arXiv. arXiv:1901.05103 [cs] (2019). https://doi.org/10.48550/arXiv.1901.05103 . http://arxiv.org/abs/1901.05103 Accessed 2025-05-27

[19] Lorensen, W.E., Cline, H.E.: Marching cubes: a high resolution 3D surface construction algorithm. In: Seminal Graphics: Pioneering Efforts that Shaped the Field, Volume 1 vol. Volume 1, pp. 347–353. Association for Computing Machinery, New York, NY, USA (1998). https://doi.org/10.1145/280811.281026 Accessed 2024-09-24

[20] Shen, T., Gao, J., Yin, K., Liu, M.-Y., Fidler, S.: Deep Marching Tetrahedra: a Hybrid Representation for High-Resolution 3D Shape Synthesis. arXiv. arXiv:2111.04276 [cs] (2021). https://doi.org/10.48550/arXiv.2111.04276 . http://arxiv.org/abs/2111.04276 Accessed 2025-06-01

[21] Genova, K., Cole, F., Sud, A., Sarna, A., Funkhouser, T.: Local Deep Implicit Functions for 3D Shape. arXiv. arXiv:1912.06126 [cs] (2020). https://doi.org/10.48550/arXiv.1912.06126 . http://arxiv.org/abs/1912.06126 Accessed 2025-07-02

[22] Huang, S., Qi, S., Xiao, Y., Zhu, Y., Wu, Y.N., Zhu, S.-C.: Cooperative Holistic Scene Understanding: Unifying 3D Object, Layout, and Camera Pose Estimation. arXiv. arXiv:1810.13049 [cs] (2019). https://doi.org/10.48550/arXiv.1810.13049 . http://arxiv.org/abs/1810.13049 Accessed 2023-09-05

[23] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv. arXiv:1912.01703 [cs, stat] (2019). https://doi.org/10.48550/arXiv.1912.01703 . http://arxiv.org/abs/1912.01703 Accessed 2023-02-08

[24] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv. arXiv:1603.04467 [cs] (2016). https://doi.org/10.48550/arXiv.1603.04467 . http://arxiv.org/abs/1603.04467 Accessed 2024-09-26

[25] Sturm, P.F., Maybank, S.J.: On plane-based camera calibration: A general algorithm, singularities, applications. In: Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149), vol. 1, pp. 432–4371 (1999). https://doi.org/10.1109/CVPR.1999.786974 . ISSN: 1063-6919. https://ieeexplore.ieee.org/abstract/document/786974 Accessed 2025-05-15