This document is the preprint of the conference paper:
L. Lucas, C. Costa and P. M. Jorge, "Cross-Device Platform for Collaborative and Immersive Experiences in Mixed Reality," 2025 9th International Young Engineers Forum on Electrical and Computer Engineering (YEF-ECE), Caparica / Lisbon, Portugal, 2025, pp. 67-73,
doi: 10.1109/YEF-ECE66503.2025.11117510.

Cross-Device Platform for Collaborative and Immersive Experiences in Mixed Reality

Letícia Lucas
ISEL - Instituto Superior de Engenharia de Lisboa
Lisbon, Portugal
leticia.lucas@isel.pt

Carla Costa
ISEL - Instituto Superior de Engenharia de Lisboa
Lisbon, Portugal
carla.costa@isel.pt

Pedro Mendes Jorge
NOVA LINCS-ISEL
Instituto Superior de Engenharia de Lisboa
Lisbon, Portugal
pedro.mendes.jorge@isel.pt

***Abstract*—This work aims to develop a platform for managing Augmented Reality (AR) and Virtual Reality (VR) devices, enabling multiple users to engage in collaborative and immersive experiences occurring both in the physical and digital world.**

The proposed solution addresses challenges in synchronization, data transmission, and real-time rendering, so that the multiple participants - either co-located or geographically dispersed - can share a unified, high-fidelity experience.

This research intends to advance mixed reality technologies for education, entertainment, and business purposes.

To demonstrate this cross-device approach, an application using the MagicLeap2 and MetaQuest3 devices is implemented. This application allows various users to collaboratively interact within the same mixed reality environment.

Unity3D is the core framework of this system, and OpenXR is the layer that ensures hardware compatibility.

***Index Terms*—Mixed Reality, Cross-Device Development, Environment Synchronization, Immersive Experiences.**

I. Introduction

The exponential technological evolution of recent years mandates the redefinition of human-machine interaction paradigms. Virtual Reality (VR) and Augmented Reality (AR) technologies allow the creation of immersive experiences previously considered unattainable, opening up infinite possibilities across various sectors such as healthcare, industry, or entertainment [1].

Education exemplifies this potential transformation, as traditional lecture-based approaches often fail to meet modern demands for dynamic learning environments [2]. AR and VR technologies can bridge this gap by enabling immersive and embodied learning experiences in which students actively interact with both physically and virtual elements and collaborate in both co-located and remote settings. Recent studies have shown that these approaches enhance learners' motivation, deepen conceptual understanding, support long-term knowledge retention, and foster critical thinking and problem-solving skills. [3] [4] [5].

Still, despite clear promise, widespread adoption remains limited by systemic challenges, particularly a fragmented technological landscape characterized by isolated, proprietary solutions for specific platforms [6]. This forces developers to create multiple versions of the same content, while users struggle with compatibility issues.

This research aims to address these challenges through developing an open-source cross-platform application for collaborative Mixed Reality (MR) experiences, enabling both AR and VR device users to interact within the same real and/or digital environment regardless of hardware.

The implementation utilizes Unity 3D as the development framework and OpenXR as the compatibility layer, demonstrating interaction between Magic Leap 2 (AR device) and Meta Quest 3 (MR device) users within a shared virtual space. This modular approach facilitates adaptation for future devices, promoting interoperability and reducing platform-specific development needs.

The proof of work focuses on:

1) **Collaborative Interaction:** enabling users to manipulate virtual objects using marker-based alignment techniques;
2) **Environment Synchronization:** implementing shared rooms where users observe real-time scene changes;
3) **Modular Support:** facilitating seamless integration of new devices without system rework.

II. Related Work

The fragmentation of Mixed Reality technologies has led to the development of standards such as OpenXR (with XR standing for Extended Reality), an open Application Programming Interface (API) by the Khronos Group that enables cross-platform VR and AR application development [7]. Unity 3D [8], combined with OpenXR and toolkits like the Mixed Reality Toolkit (MRTK), has drastically facilitated MR application development across multiple platforms.

Networking solutions such as Photon PUN and Mirror are also often employed to enable real-time multi-user interactions, although they lack robust spatial alignment across heterogeneous devices. Existing spatial alignment techniques, including visual markers and spatial anchors, provide partial solutions but fail to ensure precise synchronization between

AR and VR users [9].

Several platforms that also aim to integrate interactions between AR and VR users can also be found:

- **ENGAGE XR:** An XR platform widely used in education and enterprise sectors for immersive collaboration, offering secure virtual meeting spaces [10].
- **vTime XR:** A social XR network allowing users to interact in virtual or augmented spaces, supporting multiple platforms including mobile devices [11].
- **VirtualNexus:** A system that synchronizes VR and AR users' perspectives through 360-degree video environments, enhancing remote collaboration [12].
- **Meta Metaverse:** A suite of XR applications designed for cross-device compatibility and various levels of immersion [13].

While these platforms demonstrate progress in AR-VR interoperability, they often remain limited to specific ecosystems or lack seamless cross-device spatial alignment [14]. Additionally, most educational MR applications are confined to proprietary hardware [15].

This research's purpose is to address these limitations by developing a cross-platform collaborative system leveraging Unity 3D and OpenXR. This study contributes to a more inclusive and extensible Mixed Reality ecosystem for educational and collaborative applications.

III. Development

A. System Architecture

The platform presented in this work proposes a client-server architecture. This solution addresses key challenges in mixed-reality collaborative environments, namely real-time data synchronization, multitasking, object ownership, low-latency performance, and cross-device interoperability. Fig. 1 illustrates the implemented solution.

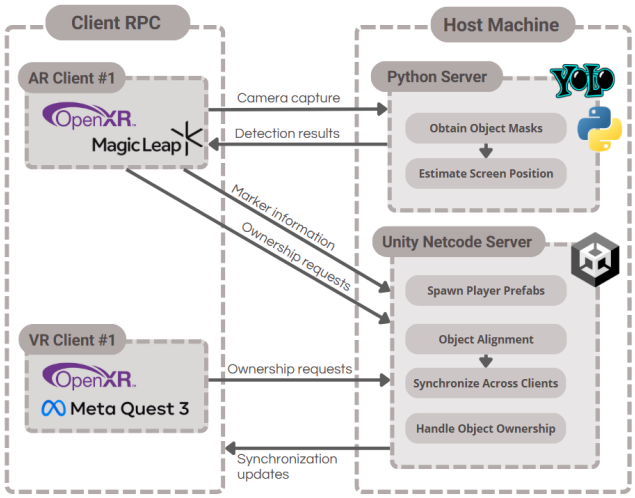

Fig. 1. Diagram representing the implemented architecture.

The proposed architecture has the following general ideas::

- **Unity Netcode Server:** The platform employs a **server-authoritative model** built on Unity's Netcode for GameObjects (NGO) framework [16]. As such, the server acts as the central authority for maintaining game state, processing client inputs, and synchronizing object transformations across connected devices.
- **XR Clients:** Heterogeneous devices that connect to the central server while maintaining device-specific interaction capabilities.
- **Processing Server:** A Python-based server based on You Only Look Once (YOLO) detection models for marker detection, it communicates with the Unity server via WebSockets.

The system implements a hybrid communication model that aims to maximize the separation of time-critical synchronization logic - which occurs via Unity's Netcode transport layer - from computation-intensive computer vision tasks which are offloaded to a specific processing server. Messages between clients and the server are sent using Remote Procedure Calls (RPC).

When a user launches the application on their device, the following process ensues:

- The app allows the user to search for and **join a session** (hosted in the Unity Netcode server);
- Within that session, based on the device that joined, the server **initializes a player prefab on the client**, with that device's specific implementation;
- Clients **register their devices and capabilities**, which includes initialization of their interaction and camera subsystems, as will be further detailed;

Due to its importance in the application, further details regarding **marker detection and object alignment implementation** will be provided in Section III-C, but to provide a first overview of the overall process, the following flow is implemented:

- Any connected AR devices will **capture and transmit camera frames** overtime to the Python processing server via WebSockets ("Camera capture" in Fig. 1);
- The Python server employs optimized detection algorithms to **identify markers in the physical environment**;
- Detections are return to the device, which applies all necessary screen-to-world transformations to get the **real-world position of the markers** ("Detection results" in Fig. 1);
- The client forwards these results to the server, which handles **object placement and synchronization across all clients** ("Marker information" and "Object Synchronization" in Fig. 1);

This approach **enables VR users to interact with digital representations of real-world objects detected by AR**

users, even if they can not spawn those objects themselves.

Regarding performance, this solution stems from several design tradeoffs: since AR devices tend to have limited computational resources, marker detection was offloaded to a dedicated Python server instead of being processed on-device. This reduces the computational burden on the XR hardware, even though it requires transferring more data between components. However, the ability to free up resources for other tasks on the device outweighs the drawbacks of this approach.

Besides, transferred image data is **processed at adjustable resolution based on available bandwidth and using compression techniques**.

Moreover, the Unity server **always maintains authority over shared objects unless a client is directly manipulating it**, which also removes latency on the XR device.

To sum up, this implementation achieves the following features:

- **Modularity:** With clear separation of synchronization concerns from computer vision/detection tasks;
- **Computational offloading:** By removing all intensive tasks to the server, performance on resource-constrained XR devices is preserved.
- **Server-authoritative model:** A paradigm that enables consistency across all clients and reduces exploitation potential in educational contexts.
- **Real-time functionality:** The overall architecture enables low-latency when it comes to virtual object alignment (registration) and synchronization.
- **Adaptability:** The distributed architecture easily accommodate the addition of new devices and processing nodes as the application expands. It can also incorporate various detection algorithms without requiring client-side changes.

B. Device Functionalities

The system leverages **OpenXR** to provide a unified development approach while maintaining device-specific capabilities for interaction and camera management.

As stated earlier, **when a client joins a session the server instantiates a device-specific player prefab**, configured for that device's capabilities. This prefab encapsulates two subsystems: the **interaction subsystem** and the **camera management subsystem** (in AR devices). Both systems operate concurrently, using asynchronous tasks to ensure seamless interaction and real-time updates across devices.

1) User Interaction Subsystem: Despite OpenXR providing a standard interface, applications must extend it with device-specific implementations to fully leverage each platform's unique capabilities.

Using a series of tools provided by the OpenXR package, users of VR and AR devices alike are able to manipulate objects in the scene. The following means are supported:

- **Controllers:** Used by both types of devices. Interaction events work based on button usage and raycast collision detection;
- **Hand Gesture:** Used exclusively by AR devices. OpenXR already provides a straight forward way to quickly create gesture-to-event configurations (e.g. fist closed over an object triggers a grab event), which facilitates implementation.

While OpenXR simplifies interaction across devices, synchronizing spatial and state changes across all clients—especially between AR and VR systems—poses a more significant challenge. The interaction system uses the following approach:

- **Device-specific input is captured**, OpenXR configuration asserts if an event is triggered;
- If so, the **client asks the server for ownership over the object**;
- If authorized, changes processed by the new owner's interaction subsystem are **forwarded to all other clients**;
- Once let go of, **server regains object ownership**;

2) Camera Management Subsystem: Camera management is essential for AR devices, as they have to periodically capture frames and process them for marker detection and spatial alignment.

With the specific implementation dependent on the used device - as each device might require different initialization parameters and permissions - this subsystem ensures:

- **Camera Initialization:** By requesting all needed permissions and starts up connection to the physical camera;
- **Frame Capture:** Using asynchronous threading, frames are periodically captured and compressed;
- **Frame Processing:** These frames are asynchronously sent to the aforementioned Python server, which handles marker detection logic;
- **Detection Handling:** When detections are received, screen-to-world coordinates transformations are applied based on the specific camera parameters - these coordinates refer to a coordinate system with the device's position at its origin;
- **Transmission:** The client informs the server of these detections via a RPC, which handles alignment logic, as will be described in Section III-C;

Maintaining a shared spatial reference between all devices is critical. While using a device's intrinsic and extrinsic camera parameters allows the acquisition of coordinates within that device's own coordinate system, **it does not necessarily align across other devices**, with their own system. This

issue is easily solved since the server takes responsibility for spawning objects - these are placed in the server's own environment coordinates, inside of which each device is positioned at spawn.

Since VR devices lack physical cameras, they rely solely on the usage of a virtual cameras to simulate vision and to set their coordinates system origin. To make the environment around them more natural, virtual spaces such as classrooms or labs can be used.

In short, this system ensures real-time interaction and camera management. By using asynchronous tasks, it allows efficient object manipulation across devices, with the server managing shared spatial references for consistent alignment. This modular approach enhances overall interoperability.

C. Marker Detection and Virtual Object Alignment

In the XR environment, virtual objects must be accurately positioned and synchronized based on real-world markers/reference points. This project defines two key object types — "Spawnables" and "Interactables" — to structure this behavior efficiently.

A Spawnable is a virtual conceptualization of the marker itself. It can be thought of as a tray that holds other virtual items - the Interactables. When these items are sitting on the tray, they move along with it, keeping their same positions relative to the tray. However, the items (Interactables) may be picked up or even fully removed from the tray and handled independently.

In the system's context, the Spawnable ('tray') is an invisible virtual object that's always attached to a real-world marker, while the Interactables ('items') are virtual items that users can interact with. This setup ensures that virtual objects stay appropriately aligned with the real markers when needed, while still allowing user manipulation.

The core responsibilities of each component follow:

- **Spawnable:**
  - **Docking Behaviour:** Allows Interactables to attached or detach - in other words, enables or disables hierarchical parenthood over those objects;
  - **Network Ownership:** Controls network ownership, ensuring only one client has authority over its Interactables at any given time;
  - **Transformation Synchronization:** Synchronizes transformations (position, rotation, scale) across all clients;
- **Interactable:**
  - **User Input Processing:** Processes user input for grabbing and releasing;
  - **Ownership Requests:** Triggers ownership requests when interacted with;
  - **Docking Tracking:** Tracks docking state to determine when to reattach to the Spawnable (i.e. when moved past a certain distance);

Regarding the implementation, Fig. 2 presents a UML class diagram depicting the relationships between the core components: `Spawnable`, `Interactable`, and `MarkerInfo` - a data structure that stores a marker's ID, position, rotation, and size.

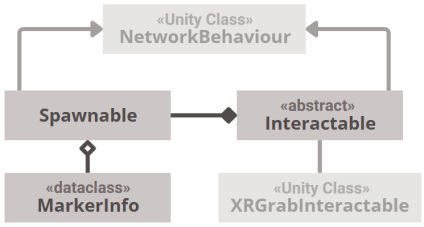

Fig. 2. Class diagram illustrating the implementation of "spawnable" and "interactable" concepts.

As illustrated, both `Spawnable` and `Interactable` inherit from Unity's `NetworkBehaviour` class, providing synchronization capabilities. `Spawnable` objects maintain references to `MarkerInfo` structures which define the alignment parameters. The abstract `Interactable` class works as a front for XR interaction handling, maintaining a reference to their parent `Spawnable` for all necessary network and spatial management.

By separating networking concerns from interaction logic in this way, the design follows a single responsibility principle, allowing each class to focus on its core functionality.

1) Detection and Alignment Logic: The implemented approach to object spawning and handling follows a marker-based detection system integrated with Unity's networking framework and is supported, primarily, by two components:

- **`DetectionConfiguration`:** Markers might not always spawn the same object, as it depends on the application context. This component provides a fast way to adapt the system by allowing the creation of simple marker-to-prefab mappings. This abstraction defines **what markers are detected, and what `Spawnable` prefab each one is assigned to**.
- **`NetworkObjectManager`:** A centralized manager spawned onto the server at launch. It processes marker detection events (according to an assigned a `DetectionConfiguration`) and handles object life-cycle management across the network. The overall process will be further detailed throughout this section.

Fig. 3 details the various steps that the manager follows when a marker is detected in the physical environment.

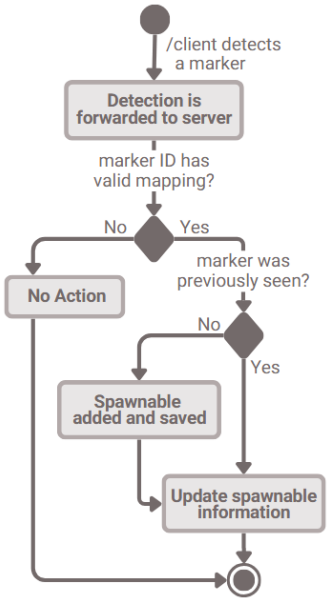

Fig. 3. Diagram that illustrates the spawning process, handled by the manager on the server.

To comment on this approach, the chosen architecture enhances modularity and efficiency by clearly separating detection configuration from runtime logic, centralizing network operations, and ensuring each object handles only its specific interactions.

2) Marker Detection Server: To achieve adaptable object detection in immersive experiences, several approaches can be considered. Standardized fiducial markers, such as ArUco codes [17], posed as the first implementation option. These black and white markers are designed to be easily recognizable by computer vision systems and provide reliable positioning information when placed in an environment. They offer a simple implementation path with minimal computational demands, and support for their detection is already built-in in the OpenXR framework.

However, a more realistic experience would require the use of more complex shapes, with markers that resemble other specific objects. To better fit into the context of the environment, a YOLO model will be employed. YOLO is a state-of-the-art deep learning-based object detection framework capable of processing images in real-time and with high accuracy [18] - both highly important in XR contexts that rely on seamless interaction [19].

Regarding the detection process, as stated before, an asynchronous WebSocket server is implemented using Python. It manages to seamlessly process image data and return detected objects in real-time, based on a given YOLO model. The server is designed to handle multiple client requests with minimal latency.

As referenced in Section III-B, AR clients will send captured frame data to the server, which in turn **returns the following information regarding each detected marker:**

- **Marker Identification:** Which will later be used to determine wether or not it should spawn an object;
- **Mask Corner Positions:** Pixel coordinates of the quadrilateral mask that best fits the detected marker;

With this information, each device is able to quickly calculate marker pose (position and rotation) in its coordinate system and send an alignment request over to the Unity server.

IV. Example of Use

To validate the proposed platform, an interactive educational application on renewable energies was implemented. A solar panel image serves as a marker, and a virtual sun is assigned as its corresponding Spawnable object.

A YOLOv11 segmentation model was trained using a custom dataset composed of roughly 30 annotated images of small printed solar panels in various positions and lighting conditions. The dataset was limited but sufficient for the application environment. Segmentation was used instead of standard detection to precisely capture the solar panel's shape, enabling more accurate pose estimation (especially rotation calculation) for AR alignment.

When a marker is detected, the server spawns the virtual sun at the calculated position, aligning it with the marker's pose using the Spawnable and Interactable concepts. Clients can then drag the sun around the scene, changing the incidence angle displayed in the User Interface (UI). Through the UI, users can also adjust the cloud cover percentage. Based on both these parameters, the energy generated by the panel is estimated.

This test was run with two clients: one used an AR MagicLeap 2 device, while the other ran on a local computer to simulate a generic VR device. Fig. 4 shows an example of the application at work.

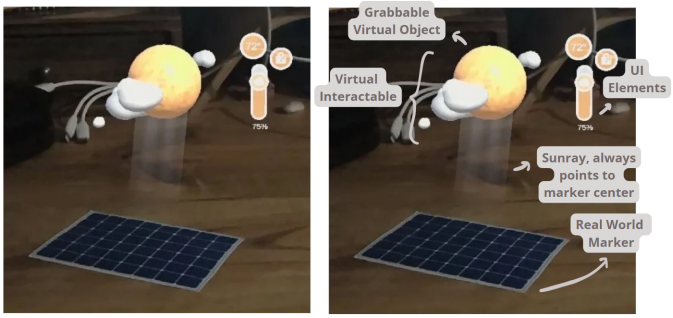

Fig. 4. Application example: (a) Capture taken from the AR device showing a sun Interactable object. (b) Annotations explaining key components.

This simple test confirmed the following:

- Marker detection operates asynchronously, ensuring no bottlenecks.
- Clients successfully transmit detection data to the server, which, in turn, correctly tracks detected markers and assigns corresponding objects based on the configured mapping.
- Spawned objects remain synchronized across all clients.
- Object alignment is maintained dynamically, with virtual objects adapting to marker pose updates.
- User interaction with the AR device is fully functional, with modifications propagating across the network in real time.
- Undocking behavior is correctly implemented — if an Interactable is moved beyond a predefined threshold, it detaches from the Spawnable and behaves independently (it stops following the marker).

These findings confirm that the system achieves stable real-time tracking, interaction, and synchronization, meeting the intended performance and usability requirements.

V. Conclusions

This research successfully set the groundwork for a cross-device platform that enables collaborative mixed reality experiences across heterogeneous devices. It focused on addressing critical technical challenges such as spatial alignment and real-time synchronization.

The proposed architecture presents the following key advantages:

- **Architectural Optimization:** A client-server architecture efficiently distributes processing tasks, offloading computer vision computations to a dedicated server while preserving responsive interactions on resource-constrained XR devices.
- **Real-time Communication:** Using Unity's Netcode framework with custom synchronization protocols enables seamless interactions with minimal latency across different devices.
- **Modularity and Extensibility:** The separation of concerns into distinct subsystems ensures that the platform remains easily adaptable. New devices and capabilities can be integrated without system-wide modifications.

Preliminary testing with solar panel detection also confirms that the system achieves stable marker recognition, with both accurate and synchronized object alignment and interaction across devices.

In conclusion, this research takes a step toward overcoming the challenges of fragmentation. By providing an open-source, cross-device platform for collaborative experiences, it contributes to making immersive technologies more accessible and practical for a varying range of applications. The modular architecture ensures adaptability alongside evolving XR hardware, laying the groundwork for future innovations.

While a strong foundation has already been established, this work will continue to incorporate further enhancements:

- Address the need for validation in more complex scenarios by incorporating additional heterogeneous devices, such as the Meta Quest 3, and multiple simulated VR clients on distinct machines to evaluate cross-device synchronization under higher user loads, aiming to assess the platform's scalability and stability in diverse collaborative environments.
- Collect objective performance metrics on system responsiveness and data throughput, including evaluation of the bit rate generated by each device to estimate total bandwidth requirements for real-time interactions as the number of connected users increases. Based on the collected data, any necessary refinement of the used synchronization protocols and compression techniques will be explored. Special attention may also be given to optimize data transfer for resource-constrained devices.
- Collaborate with Educational Sciences researchers to implement the platform in solar energy education, incorporating Inquiry-Based Learning and collaborative learning models, and systematically evaluation of learning outcomes across various educational levels.

Acknowledgment

The authors gratefully acknowledge support from the ICSE Science Factory project (2023–2026), funded by the European Union's Horizon Europe Research and Innovation Programme under the Call HORIZON-WIDERA-2022-ERA-01 (Grant Agreement No. 101093387). DOI: 10.3030/101093387.

References

[1] Grand View Research, "Virtual Reality (VR) in Healthcare Market Size, Share & Trends Analysis Report," 2024. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/virtual-reality-vr-healthcare-market-report

[2] A. A. Mazhar, M. M. Al Rifaee, "A Systematic Review of the use of Virtual Reality in Education," 2023 International Conference on Information Technology (ICIT), Amman, Jordan, 2023, pp. 422-427, doi: 10.1109/ICIT58056.2023.10225794

[3] Akçayır, M., Akçayır, G. (2017). Advantages and challenges associated with augmented reality for education: A systematic review of the literature. Educational Research Review, 20, 1–11. https://doi.org/10.1016/J.EDUREV.2016.11.002

[4] Banjar, A., Xu, X., Iqbal, M. Z., Campbell, A. (2023). A systematic review of the experimental studies on the effectiveness of mixed reality in higher education between 2017 and 2021. Computers & Education: X Reality, 3, 100034. https://doi.org/10.1016/J.CEXR.2023.100034

[5] Johnson-Glenberg, M. C. et al., Front. Virtual Reality 4, 1047833 (2023). Available: https://doi.org/10.3389/FRVIR.2023.1047833

[6] T. Bezmalinovic, "Virtual reality is facing an old threat again: fragmentation," Mar. 2024. [Online]. Available: https://mixed-news.com/en/vr-industry-fragmentation-threat/

[7] Khronos Group, "OpenXR Overview," 2024. [Online]. Available: https://www.khronos.org/openxr/

[8] Unity Technologies, "Unity Manual," Unity Documentation. [Online]. Available: https://docs.unity3d.com/Manual/index.html

[9] J. Fu, A. Rota, S. Li, J. Zhao, Q. Liu, E. Iovene, G. Ferrigno, and E. De Momi, "Recent advancements in augmented reality for robotic applications: A survey," Actuators, vol. 12, no. 8, p. 323, 2023, doi: 10.3390/act12080323

[10] J. Stephen, "University of Miami Builds an Educational Metaverse with ENGAGE," May 2024. [Online]. Available: https://www.xrtoday.com/virtual-reality/university-of-miami-builds-an-educational-metaverse-with-engage/

[11] V. Roberts, "vTime Becomes vTime XR - The World's First Cross-Reality (XR) Social Experience," Feb 2019. [Online]. Available: https://vtime.net/press/2019-02-19T12:00:00/vtime-becomes-vtime-xr-the-worlds-first-cross-reality-xr-social-experience/

[12] D. Herrick, "Redefining immersive experiences through XR and AI," May 2024. [Online]. Available: https://magazine.ucsb.edu/fall-winter-2024/redefining-immersive-experiences-through-xr-and-ai

[13] E. Ravenscraft, "What Is the Metaverse, Exactly?," Jun 2023. [Online]. Available: https://www.wired.com/story/what-is-the-metaverse/

[14] L. Yang, S. Ni, Y. Wang, A. Yu, J. Lee, P. Hui "Interoperability of the Metaverse: A Digital Ecosystem Perspective Review," 2024, doi: 10.48550/arXiv.2403.05205

[15] N. Sala, "Virtual reality, augmented reality, and mixed reality in education: A brief overview," in Virtual and Augmented Reality in Education, Art, and Museums, Hershey, PA: IGI Global, 2021, ch. 3, doi: 10.4018/978-1-7998-4960-5.ch003

[16] Unity Technologies, "NetworkObject," Unity Multiplayer Documentation, 2024. [Online]. Available: https://docs-multiplayer.unity3d.com/netcode/current/basics/networkobject/

[17] OpenCV, "Detection of ArUco Markers," OpenCV Documentation. [Online]. Available: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html

[18] Ultralytics, "Ultralytics YOLO Docs," Ultralytics Documentation. [Online]. Available:https://docs.ultralytics.com/pt

[19] N. Bispo, "Using YOLO for Real-Time Object Detection with Koyeb GPUs," Koyeb Tutorials, 2024. [Online]. Available: https://www.koyeb.com/tutorials/using-yolo-for-real-time-ai-image-detection-with-koyeb-gpus