# IRIS: An Immersive Robot Interaction System

**Xinkai Jiang**[1], **Qihao Yuan**[2], **Enes Ulas Dincer**[1], **Hongyi Zhou**[1], **Ge Li**[1], **Xueyin Li**[1], **Xiaogang Jia**[1], **Timo Schnizer**[1], **Nicolas Schreiber**[1], **Weiran Liao**[1], **Julius Haag**[1], **Kailai Li**[2], **Gerhard Neumann**[1], **Rudolf Lioutikov**[1]

[1]Karlsruhe Institute of Technology, [2]University of Groningen

Project Page: https://intuitive-robots.github.io/iris-project-page/


Figure 1: We present **IRIS**, an **I**mmersive **R**obot **I**nteraction **S**ystem designed to support various simulators and real-world scenarios.

**Abstract:**

This paper introduces IRIS, an Immersive Robot Interaction System leveraging Extended Reality (XR). Existing XR-based systems enable efficient data collection but are often challenging to reproduce and reuse due to their specificity to particular robots, objects, simulators, and environments. IRIS addresses these issues by supporting immersive interaction and data collection across diverse simulators and real-world scenarios. It visualizes arbitrary rigid and deformable objects, robots from simulation, and integrates real-time sensor-generated point clouds for real-world applications. Additionally, IRIS enhances collaborative capabilities by enabling multiple users to simultaneously interact within the same virtual scene. Extensive experiments demonstrate that IRIS offers efficient and intuitive data collection in both simulated and real-world settings.

**Keywords:** Human-Robot Interaction, Extended Reality, Imitation Learning

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

# 1 Introduction

Robot learning relies on diverse and high-quality data to acquire complex behaviors [1, 2]. Recent studies indicate that models trained on more varied and complex datasets generalize more effectively across diverse scenarios [3, 4, 5]. By providing immersive perspectives and interactions, Extended Reality [1] (XR) has emerged as a promising tool for efficient and intuitive large-scale data collection in both simulation [7, 8, 9] and real-world environments [10, 11]. However, existing XR approaches face significant challenges when reused or reproduced in new scenarios, primarily due to three limitations: *asset diversity*, *platform dependency*, and *XR device compatibility*.

Current approaches [12, 7, 8, 13, 14] rely heavily on predefined sets of objects and robot models, thereby exhibiting limited *asset diversity*. Furthermore, most methods [15, 16, 9, 8] are specifically tailored to particular simulators or real-world conditions, resulting in substantial *platform dependency*. This limitation significantly reduces reusability and complicates adaptation to different simulation platforms. Additionally, existing XR frameworks [16, 17, 10, 18] are typically optimized for specific XR headset versions, leading to poor *device compatibility*. Together, these limitations severely constrain reproducibility and broader adoption of XR-based data collection and robot interaction methodologies within the research community.

To address these challenges, we propose **IRIS**—an **I**mmersive **R**obot **I**nteraction **S**ystem, demonstrated in Figure1. IRIS is a general and extensible framework that supports various simulators and real-world environments, with compatibility across different XR headsets. It is designed to generalize robot applications with XR across six key features: *Cross-Scene*, *Cross-Embodiment*, *Cross-Simulator*, *Cross-Reality*, *Cross-Platform*, and *Cross-User*.

**Cross-Scene** enables XR systems to handle arbitrary simulated objects, removing constraints from predefined models. IRIS introduces a unified scene specification representing all objects as data structures with meshes, materials, and textures. This specification is transmitted to XR headsets for consistent scene rendering, with dynamic updates during simulation. Through its flexible and dynamic architecture, IRIS is also the first XR-based system that supports deformable objects manipulation. **Cross-Embodiment** is achieved by modeling robots as compositions of standard objects, enabling seamless compatibility with diverse robot embodiments without requiring specialized configurations. **Cross-Simulator** ensures compatibility with a range of simulation engines. Since the unified scene specification is simulator-agnostic, new simulators can be supported by implementing a parser to translate their scenes into this format. This flexibility is demonstrated by IRIS's support for MuJoCo [19], IsaacSim [20], CoppeliaSim [21], and Genesis [22]. **Cross-Reality** allows IRIS to operate across both simulated and real-world environments. For real-world applications, IRIS incorporates point cloud visualization using camera data, facilitating immersive data collection. **Cross-Platform** ensures compatibility across XR devices. IRIS implements its XR application using the Unity framework [23], with modular design separating visualization and interaction logic. This allows developers to deploy the system on new XR headsets by reusing visualization modules and implementing device-specific input handling. IRIS has been successfully deployed on the Meta Quest 3 and HoloLens 2. **Cross-User** supports collaborative multi-user interaction within a shared scene via a communication protocol that synchronizes XR headsets. This enables coordinated tasks and collective data collection in both virtual and real environments. Table 1 highlights the advantages of IRIS over existing XR-based systems across these features.

The contributions of IRIS are summarized as follows: (1) A unified scene specification that integrates seamlessly with multiple robot simulators, enabling consistent visualization and interaction across diverse XR headsets, while promoting reproducibility and reusability. (2) The first XR-based system to support deformable object manipulation in robot simulators, allowing realistic interaction and data collection for soft-body tasks. (3) A collaborative, multi-user framework for XR applications that enhances robot data collection through synchronized interactions in shared virtual or physical environments.

[1] Extended Reality (XR) is an umbrella term encompassing Augmented Reality, Mixed Reality, and Virtual Reality [6].

| | Cross-Scene | Cross-Embodiment | Cross-Simulator | Cross-Reality | Cross-Platform | Cross-User | Control Space |
|---|---|---|---|---|---|---|---|
| Fan et al. [24] | Limited | Single Robot | Unity | Real | Meta Quest 2 | N/A | Cartesian |
| ARC-LfD [12] | N/A | Single Robot | N/A | Real | HoloLens | N/A | Cartesian |
| Zhu et al. [25] | Limited | Single Robot | N/A | Real | HTC Vive Pro | N/A | Cartesian |
| Jiang et al. [7] | Limited | Single Robot | N/A | Real | HoloLens 2 | N/A | Joint & Cartesian |
| Mosbach et al. [15] | Available | Single Robot | IsaacGym | Sim | Vive | N/A | Joint & Cartesian |
| Holo-Dex [26] | N/A | Single Robot | N/A | Real | Meta Quest 2 | N/A | Joint |
| ARCADE [8] | N/A | Single Robot | N/A | Real | HoloLens 2 | N/A | Cartesian |
| DART [9] | Limited | Limited | Mujoco | Sim | Vision Pro | N/A | Cartesian |
| ARMADA [17] | N/A | Limited | N/A | Real | Vision Pro | N/A | Cartesian |
| Meng et al. [18] | Limited | Single Robot | PhysX | Sim & Real | HoloLens 2 | N/A | Cartesian |
| Bunny-VisionPro [27] | N/A | Single Robot | N/A | Real | Vision Pro | N/A | Cartesian |
| IMMERTWIN [28] | N/A | Limited | N/A | Real | HTC Vive | N/A | Cartesian |
| Open-TeleVision [11] | N/A | Limited | N/A | Real | Meta Quest, Vision Pro | N/A | Cartesian |
| Szczurek et al. [29] | N/A | Limited | N/A | Real | HoloLens 2 | Available | Joint & Cartesian |
| OPEN TEACH [10] | N/A | Available | N/A | Real | Meta Quest 3 | N/A | Joint & Cartesian |
| **Ours** | **Available** | **Available** | **Mujoco, CoppeliaSim, IsaacSim** | **Sim & Real** | **Meta Quest 3, HoloLens 2** | **Available** | **Joint & Cartesian** |

Table 1: Comparison of XR-based system. IRIS is compared with related works in seven aspects.

## 2 Related Work

**Teleoperation-Based Data Collection on Real Robots.** Collecting data using tele-operation on real robots has been explored by many previous works. Aloha [30] introduced a low-cost teleoperation system that collects real-world demonstrations for imitation learning. A bimanual workspace is set up, where leader robots are used to control the follower robots. Followup work [1] improved the performance, ergonomics, and robustness compared to the original design. In addition, a mobile version of Aloha [31] improved data collection outside of lab settings. GELLO [32] supports a variety of robot arms through a 3D-printed low-cost leader robots with off-the-shelf motors. In order to tele-operate dexterous end effectors, prior work has retrieved hand motion data through visual hand tracking [33] or customized gloves [2]. In contrast to IRIS, none of these approaches leverages the immersive advantages of XR.

**XR-Based Data Collection in Real World.** Common XR systems show virtual robots to help users understand how their movements control real robots [33]. For instance, recent works developed mobile apps to allow data collection in augmented reality without the need for XR headsets [34, 35], which allow more intuitive robot manipulation [8, 17, 7]. Instead of displaying the virtual robot in a third-person view, Cheng et al. [11], Iyer et al. [10] directly provide the first-person camera feed of the real robot to the user. Other systems [14, 36, 37, 28, 25, 24] visualize the real-world scene in the headset and control robot arms with controllers [25] or hand tracking [37]. XR-based data collection for dexterous hands has also been explored. For example, Arunachalam et al. [38] tracks hand motion using camera and retargets it on the real robot hand. Chen et al. [39] controls robot hand and robot arm at the same time. While these approaches do use XR, the robot data collection and interaction is limited to the real world, with no simulators used in the process.

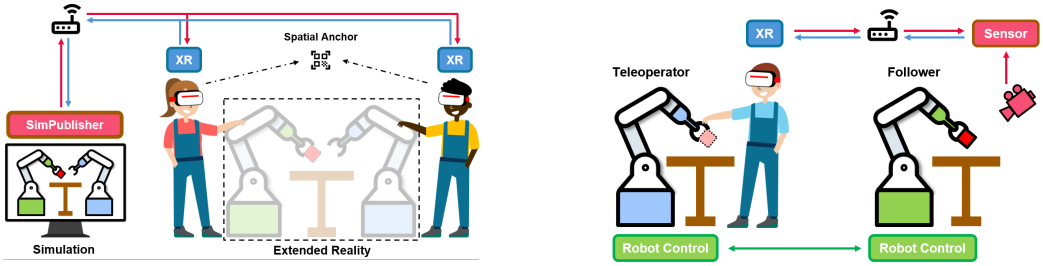
**XR-Based Data Collection in Simulation.** Real robot data collection is limited by available environments and objects. Virtual data collection offers a more efficient way to gather demonstrations while providing access to extensive 3D asset libraries. For instance, DART [9] runs a cloud-based simulation, and users collect demonstrations in any virtualized environment from any location. Mosbach et al. [15] collects dexterous hand manipulation data with a special glove device in physics simulations. Although Meng et al. [18] also leverages simulators, their virtual scene is a replica of the real scene, thus the flexibility of simulation is not fully exploited.

## 3 System Overview

This section presents the hardware and software architecture of IRIS, along with several applications which have been explored in this paper. An overview of its paradigm is shown in Figure 2.

### 3.1 System Architecture

**Node Communication Protocol.** The IRIS system operates across simulation or sensor-processing computers and XR headsets. It provides a dedicated Python library, SimPublisher, for use on the




(a) Interact with Robots in Simulation

(b) Interact with Robots in Real World

Figure 2: Paradigms of the system architecture in both simulation (left) and real world (right). All the devices are connected through a Wi-Fi router. In the left image, the simulation updates the scene to all headsets using the SimPublisher. A spatial anchor is used to align the virtual scenes across different headsets. In the right image, a sensor generates a point cloud transmitted to the XR headset, allowing the operator to clearly observe the manipulated object in front of the follower robot.

computers, in combination with a Unity application deployed on the headsets. Through this library, users are able to render objects remotely on the headsets and manage groups of headsets without the need to write C# code. This approach eliminates the requirement for developing separate Unity projects, thereby substantially reducing the cost and complexity associated with integrating robotic systems into VR applications. This architecture requires a robust network connection between all components. While Robot Operating System (ROS) [40] offers a general communication framework and it is possible to connect to Unity and XR development, it is quite heavy to deploy. IRIS was intentionally designed to be decoupled from the ROS framework in order to support a broader range of simulators and avoid introducing unnecessary dependencies. It built an lightweight communication protocol based on ZeroMQ (ZMQ) [41], and extended it by auto-node discovery features. This design choice enhances modularity and simplifies integration in diverse environments. Importantly, IRIS remains compatible with ROS through a lightweight and flexible communication interface, allowing seamless integration when needed. To ensure node discovery, the master node broadcasts UDP messages to the broadcast port on the network at a specific frequency. The network is built via Wi-Fi or cable. When a new XR node launches, it listens to the broadcast port and receives messages from the master node. Then it extracts connection details from these messages and establishes a ZMQ connection. This protocol achieves **Cross-User** ability, ensures reliable communication, automatic reconnection, and smooth recovery from disconnections, making it ideal for dynamic multi-device XR systems.

**Unified Scene Representation.** To visualize simulation scenes and virtual objects in XR headsets, existing solutions [7, 9, 8, 12] rely on predefined models, which limits flexibility for incorporating new objects and robots. IRIS overcomes this limitation with a Unified Scene Representation (USR) parsed directly from simulations. Unlike URDF, USR can be serialized with meshes and texture information and is adapted to Unity's GameObject structure, allowing scenes to be reconstructed natively in Unity. Serving as a middleware layer between simulators and Unity, USR enables objects to be generated in Python, serialized, and seamlessly loaded into Unity without additional conversion. The USR includes all objects with their geometry, meshes, materials, and textures. All the objects is loaded in this specification using a kinematic tree structure and serialized into byte format. IRIS application rebuild an identical scene upon received this specification from simulation node. IRIS provides a custom Python library named *SimPublisher* that automatically generates specifications from simulation data, then it continuously collects simulation states and transmits them to headsets at a fixed frequency. This scene specification enables IRIS to process all kinds of robots and objects in simulation, facilitating both **Cross-Scene** and **Cross-Embodiment** capabilities. The USR is a general definition that does not rely on any specific simulator. Hence, IRIS can be easily adapted to various simulators by implementing a new simulation parser to generate a scene specification from the simulator and a new publisher to update the states of the scene. Currently, IRIS supports scene parsers for **MuJoCo**, **IsaacSim**, **CoppeliaSim**, and **Genesis**, with the potential to be extended to other simulation engines as desired. This demonstrates that IRIS can be easily adapted to various benchmarks and simulators, highlighting its **Cross-Simulator** capability.

**Multiple Headsets Compatibility.** IRIS implements an XR application using Unity. This application can be directly deployed to other headset platforms using the Unity deployment pipeline, showcasing IRIS's **Cross-Platform** capability. Currently, IRIS has been tested on HoloLens 2 [42] and Meta Quest 3 [43]. Due to Meta Quest 3 visualization resolution is better than HoloLens 2, this paper conducted experiments and displayed XR scenes using this headset.

**Intuitive Robot Control Interface.** In data collection tasks or robot interaction, robot control interfaces are used to operate the robot in both simulated and real-world environments. Based on prior work [7, 10], IRIS implemented Kinesthetic Teaching (KT) and Motion Controller (MC) as its default robot controllers. The details of these two interfaces are in the Appendix. C. These two methods were used and evaluated with other interfaces in Sec. 4.1. IRIS's flexible framework allows users to easily customize and implement additional control interfaces, including hand tracking, gloves, smartphones, and motion tracking systems. Fig. 14 shows how these two interfaces work.

### 3.2 System Application

IRIS is a versatile platform with significant potential for robot learning and robotics research community. This section outlines several applications that have been explored in this work.

**General Manipulation Data Collection.** Through its flexible framework design, IRIS supports four simulators and various robot manipulation benchmarks. IRIS has been tested in some MuJoCo-based benchmarks including **Meta World** [44], **LIBERO** [45], **RoboCasa** [46], **robosuite** [47], **Fancy Gym** [48], and CoppeliaSim-based benchmark like **PyRep** [49], **Colosseum** [50]. Robots can be operated using either our default controllers (Sec. 3.1) or user-customized controllers.

**Deformable Object Manipulation.** IRIS supports deformable object by dynamically updating the mesh in real time, making it possible to train and test robotic algorithms for tasks that involve soft objects. As far as we know, no existing work has explored the manipulation of deformable objects using by rendering them in XR headsets. This paper conducted an experiment (Sec. 4.2) to validate the data collected by IRIS based on IsaacSim.

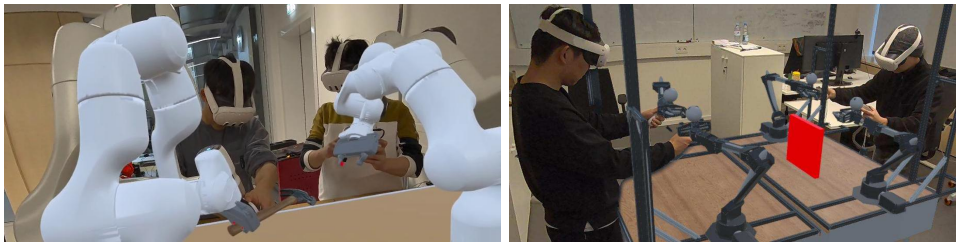


Figure 3: Collaborative manipulation in simulation via IRIS. The left image shows the collaborative manipulation for handing over a hammer between two Franka Panda robots by KT, and the right image shows that collaborative manipulation for handing over a red board between two Aloha 2 Arms by MC.

**Collaborative Manipulation.** Collaborative manipulation, where multiple users provide demonstrations simultaneously, is vital for human-robot systems [51]. Previous approaches [33] often use multiple screens, lacking XR's immersion, and typically limit control to one person while others merely observe [29]. IRIS's communication protocol enables seamless integration of devices for controlling multiple robots in shared scenes, supporting additional XR headsets with minimal setup for dynamic collaborative environments. Fig. 3 shows collaborative manipulation for handover task.

**High-Dynamic Task Data Collection and Interaction** Prior work used keyboards, 3D mice, or smartphones [54, 55, 45, 56], which are limited for complex motions. With the support for motion controllers, IRIS can capture precise user movements. This paper demonstrates the usage of motion controllers in a simulated table tennis task against an RL agent trained with BBRL [52]. Fig. 4 presents this application and an experiment of using collected data to train policies used in Sec. 4.2.

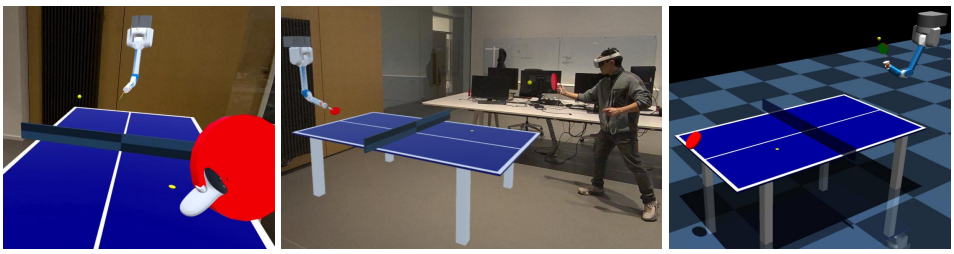


Figure 4: Playing table tennis with RL agent in Fancy Gym environment. The RL agent policy is trained with *Deep Black-Box Reinforcement Learning* (BBRL) [52, 53]

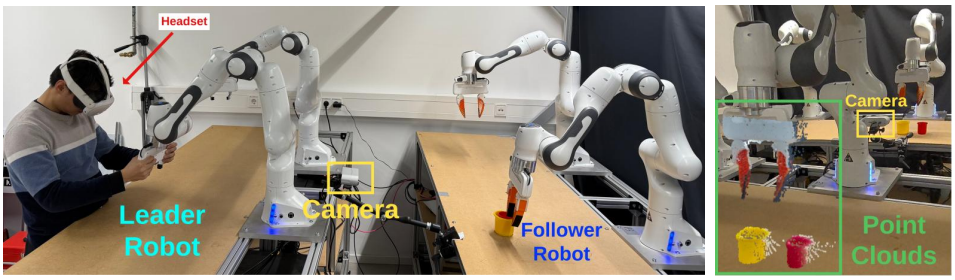



Figure 5: IRIS Real-world Application. This setup features two Franka robots: a leader robot controlled by a user wearing a Meta Quest 3 headset and a follower robot that mirrors its movements. A depth camera captures the environment for real-time point cloud visualization in XR.

**Real World Teleoperation and Data Collection.** IRIS not only supports simulation but can also be used in the real-world setting. To minimize physical obstruction from humans, tele-operation using XR is commonly employed for collecting such data. Current approaches [10, 11] that utilize video streaming to XR headsets are restricted to fixed viewpoints. IRIS overcomes this limitation by projecting point cloud from depth cameras to XR headsets, ensuring both immersion and interactivity. Fig. 5 shows the real-world application, and the details are in the Appendix. B.5.2

## 4 Experiment

In this section, we focus on three questions to demonstrate the effectiveness and scalability of IRIS: (1) How effective and intuitive is the IRIS system for data collection in terms of user experience? (2) Can data collected by IRIS be effectively used for policy training in simulation? (3) Is IRIS suitable for data collection in real-world scenarios? To address these questions, we evaluate the performance of IRIS across three groups of tasks, with detailed experimental settings presented in Appendix D.

### 4.1 User Experience Evaluation

To assess the efficiency and intuitiveness of IRIS data collection application, a pilot study was conducted by collecting demonstrations for LIBERO benchmark tasks [45]. Four tasks (Fig. 17 in the Appendix) were selected in the dimension of translation, rotation, and compound movement, including *close the microwave*, *turn off the stove*, *pick up the book in the middle and place it on the cabinet shelf*, and *turn on the stove and put the frying pan on it*. The control interface baselines for this study are two control interfaces from LIBERO: the Keyboard (KB) and the 3D Mouse (3M).

**Study Design** This study involved eight participants with no prior experience using IRIS or XR headsets. They evaluated each interface through both objective and subjective metrics. Objective measurements included success rate and average time per task, while subjective assessments were gathered via a questionnaire based on the UMUX framework [57]. The questionnaire evaluates each interface by a 7-point Likert scale in four dimensions including *Experience*, *Usefulness*, *Intuitiveness*, and *Efficiency*. This paper employed the Kruskal-Wallis test [58] for a better and more robust statistical analysis for the study.

| Interface | Task 1 | Task 2 | Task 3 | Task 4 |
|---|---|---|---|---|
| KB (LIBERO) | 0.90 | 0.725 | 0.750 | 0.500 |
| 3M (LIBERO) | **1.00** | 0.950 | 0.375 | 0.900 |
| KT (Ours) | **1.00** | **1.00** | **0.975** | 0.950 |
| MC (Ours) | **1.00** | 0.900 | **0.975** | **1.00** |

Table 2: Success rate of four interfaces. KT and MC lead to higher success rate across four tasks.

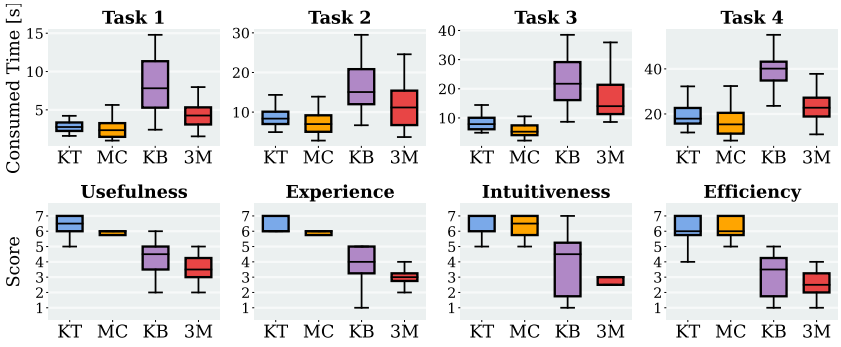



Figure 6: **The first row**: average task completion time for each interface across tasks. **The second row**: subjective evaluation scores of four metrics. Our interfaces, KT and MC in XR setting, are faster in data collection with better user experience.

**Study Result** In the result of objective metrics, Tab. 2 and Fig. 6 present the success rates and the average time consumed for each interface across the tasks. The result shows a success rate of over $90\%$ across all four tasks when using the KT and MC interfaces from IRIS, and these XR-based interface significantly outperformed the non-XR interface ($p < 0.05$ [2] ) in all conditions except MC in Task 2. The KT and MC interfaces consistently demonstrate lower task completion times than KB and 3M ($p < 0.05$) particularly for Task 3 and Task 4, indicating higher efficiency. The result of subjective result is shown in Fig. 6. The KT and MC interfaces consistently receive significant ($p < 0.05$) higher scores than KB and 3M across all criteria, indicating positive user perception and ease of use. This study demonstrates that IRIS outperformed baseline interfaces in both objective and subjective metrics, indicating it provides a more intuitive and efficient approach for data collection.

### 4.2 Policy Evaluation in Simulation

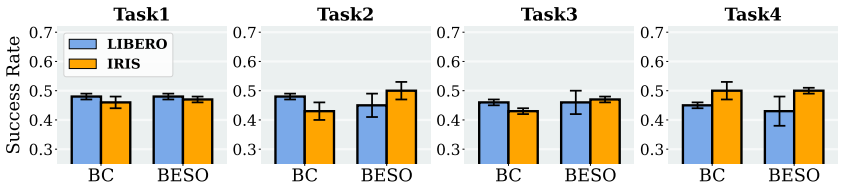



Figure 7: Performance comparison of policies trained on different datasets across LIBERO tasks

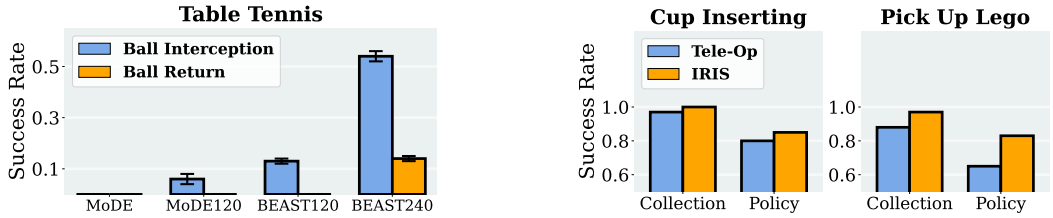
**General Manipulation** To evaluate the quality of data collected using IRIS, we employ two standard imitation learning algorithms: BC-Transformer [59] and BESO [60]. These models are trained separately on datasets collected with IRIS and on the original LIBERO dataset, with results shown in Figure 7. To ensure a fair comparison, we collect the same number of trajectories using IRIS as in LIBERO, using only the MC interface (instead of KT), since LIBERO operates in Cartesian space rather than joint space. Each model is trained for 50 epochs with three random seeds to capture performance variance, and all experiments use identical training parameters. The results demonstrate that the quality of data collected by IRIS is on par with the original LIBERO dataset.

**Deformable Objects** We also evaluate the data collected by IRIS from deformable object manipulation. Three tasks were designed to evaluate the data including *Fold Cloth*, *Lift Teddy*, and *Stow*

[2]level of statistical significance by Kruskal-Wallis test

*Teddy*. The policy used in this experiment is the U-Net diffusion model [61]. Observations are robot EEF pose, depth, and image data. The success rate of each task are *Fold Cloth*: $0.97 \pm 0.018$, *Lift Teddy*: $0.90 \pm 0.035$, and *Stow Teddy*: $0.85 \pm 0.053$.

**Dynamic Task Data Collection** To evaulate the data quality of highly dynamic task collected by IRIS, this paper uses the table tennis from Fancy_Gym [48] by motion controllers. The observation includes bat proprioceptive state and dual camera images, and the action is the desired bat position and orientation in task space. Fig. 8a shows the performance of imitation learning models [62, 63] trained on the collected data, using ball interception rate and successful return rate as evaluation metrics. These experiments validate the data from three data collection scenarios in Section 3.2. The results demonstrate that IRIS collects data of comparable quality to traditional methods, while offering significantly greater efficiency.



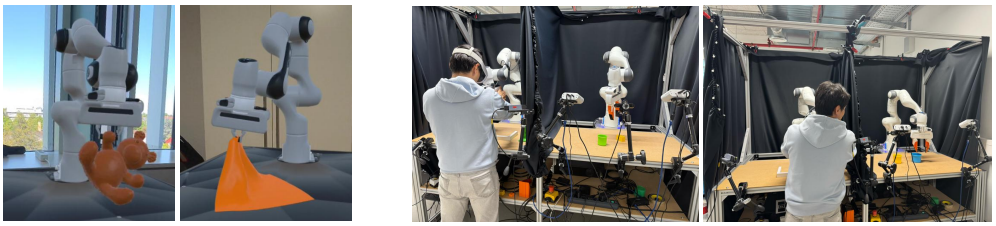

(a) Policy Performance of Table Tennis Task

(b) Policy Performance of Real World Experiment

Figure 8: Performance evaluation of policies trained on IRIS-collected data across diverse scenarios

### 4.3 Real World Evaluation

This experiment evaluates the effectiveness of IRIS for real-world data collection through two tasks: Cup Inserting and Picking Up Lego. IRIS was compared against Tele-Op, a widely used method for real robot data collection. In Tele-Op [59, 11, 10], users observe the scene via a monitor or from a distance, which is less comfortable and narrows the operator's field of view. For each task, 30 successful demonstrations were collected from both methods. These two datasets were used to train two BC-Transformer policies with identical hyperparameters. The metrics include data collection success rate (the percentage of successful attempts during data collection) and policy success rate after training. The results (Fig. 8 b) show that IRIS achieves a higher data collection success rate, and that policies trained with IRIS exhibit better quality than those trained with Tele-Op.



(a) Soft object manipulation in IsaacSim

(b) Real-world data collection by IRIS(left) and Tele-Op(right)

Figure 9: The experiment of deformable manipulation and real-world data collection

## 5 Conclusion

In this work, we introduced IRIS, an innovative framework that seamlessly integrates Extended Reality (XR) technologies with robotics data collection. IRIS addresses key challenges in reproducibility and reusability that are common in current XR-based systems. Its flexible, extendable design supports multiple simulators, benchmarks, real-world applications, and multi-user use cases. User studies demonstrate that IRIS outperforms previous data collection methods, positioning it as a promising solution for future data collection pipelines. As an open-source project, IRIS codebase promotes further research and adaptation across diverse use cases and hardware platforms.

## Limitations

Based on our development and usage experience, IRIS has three main limitations. First, the visualization in XR headsets is limited to basic RGB textures for materials. IRIS currently supports deformable objects exclusively in IsaacSim, as other simulators either lack robust deformable object capabilities or haven't implemented them. This paper selected IsaacSim for initial testing due to its superior deformable object simulation. Finally, IRIS has only been tested on Meta Quest 3 and HoloLens 2, and further evaluation on additional devices would be beneficial.


### Acknowledgments

The presented work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 448648559. Xinkai Jiang and Xiaogang Jia acknowledge the support from the China Scholarship Council (CSC).


## References


[1] J. Aldaco, T. Armstrong, R. Baruch, J. Bingham, S. Chan, K. Draper, D. Dwibedi, C. Finn, P. Florence, S. Goodrich, et al. Aloha 2: An enhanced low-cost hardware for bimanual teleoperation. *arXiv preprint arXiv:2405.02292*, 2024.

[2] C. Wang, H. Shi, W. Wang, R. Zhang, L. Fei-Fei, and C. K. Liu. Dexcap: Scalable and portable mocap data collection system for dexterous manipulation. *arXiv preprint arXiv:2403.07788*, 2024.

[3] B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 1, 2020.

[4] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[5] J. Gao, A. Xie, T. Xiao, C. Finn, and D. Sadigh. Efficient data collection for robotic manipulation via compositional generalization. *arXiv preprint arXiv:2403.05110*, 2024.

[6] Extended reality - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Extended_reality. [Accessed 11-01-2025].

[7] X. Jiang, P. Mattes, X. Jia, N. Schreiber, G. Neumann, and R. Lioutikov. A comprehensive user study on augmented reality-based data collection interfaces for robot learning. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, pages 333–342, 2024.

[8] Y. Yang, B. Ikeda, G. Bertasius, and D. Szafir. Arcade: Scalable demonstration collection and generation via augmented reality for imitation learning. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2855–2861. IEEE, 2024.

[9] Y. Park, J. S. Bhatia, L. Ankile, and P. Agrawal. Dexhub and dart: Towards internet scale robot data collection. *arXiv preprint arXiv:2411.02214*, 2024.

[10] A. Iyer, Z. Peng, Y. Dai, I. Guzey, S. Haldar, S. Chintala, and L. Pinto. Open teach: A versatile teleoperation system for robotic manipulation. *arXiv preprint arXiv:2403.07870*, 2024.

[11] X. Cheng, J. Li, S. Yang, G. Yang, and X. Wang. Open-television: Teleoperation with immersive active visual feedback. In *8th Annual Conference on Robot Learning*.

[12] M. B. Luebbers, C. Brooks, C. L. Mueller, D. Szafir, and B. Hayes. Arc-lfd: Using augmented reality for interactive long-term robot skill maintenance via constrained learning from demonstration. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3794–3800. IEEE, 2021.

[13] A. George, A. Bartsch, and A. B. Farimani. Openvr: Teleoperation for manipulation. *SoftwareX*, 29:102054, 2025.

[14] A. Naceri, D. Mazzanti, J. Bimbo, Y. T. Tefera, D. Prattichizzo, D. G. Caldwell, L. S. Mattos, and N. Deshpande. The vicarios virtual reality interface for remote robotic teleoperation: Teleporting for intuitive tele-manipulation. *Journal of Intelligent & Robotic Systems*, 101: 1–16, 2021.

[15] M. Mosbach, K. Moraw, and S. Behnke. Accelerating interactive human-like manipulation learning with gpu-based simulation and high-quality demonstrations. In *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pages 435–441. IEEE, 2022.

[16] J. I. Lipton, A. J. Fay, and D. Rus. Baxter's homunculus: Virtual reality spaces for teleoperation in manufacturing. *IEEE Robotics and Automation Letters*, 3(1):179–186, 2017.

[17] N. Nechyporenko, R. Hoque, C. Webb, M. Sivapurapu, and J. Zhang. Armada: Augmented reality for robot manipulation and robot-free data acquisition. *arXiv preprint arXiv:2412.10631*, 2024.

[18] L. Meng, J. Liu, W. Chai, J. Wang, and M. Q.-H. Meng. Virtual reality based robot teleoperation via human-scene interaction. *Procedia Computer Science*, 226:141–148, 2023.

[19] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[20] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi:10.1109/LRA.2023.3270034.

[21] E. Rohmer, S. P. N. Singh, and M. Freese. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013. www.coppeliarobotics.com.

[22] G. Authors. Genesis: A universal and generative physics engine for robotics and beyond, December 2024. URL https://github.com/Genesis-Embodied-AI/Genesis.

[23] U. Technologies. Unity - Manual: Unity 6 User Manual — docs.unity3d.com. https://docs.unity3d.com/6000.0/Documentation/Manual/UnityManual.html. [Accessed 31-01-2025].

[24] W. Fan, X. Guo, E. Feng, J. Lin, Y. Wang, J. Liang, M. Garrad, J. Rossiter, Z. Zhang, N. Lepora, et al. Digital twin-driven mixed reality framework for immersive teleoperation with haptic rendering. *IEEE Robotics and Automation Letters*, 2023.

[25] Y. Zhu, B. Jiang, Q. Chen, T. Aoyama, and Y. Hasegawa. A shared control framework for enhanced grasping performance in teleoperation. *IEEE Access*, 2023.

[26] S. P. Arunachalam, I. Güzey, S. Chintala, and L. Pinto. Holo-dex: Teaching dexterity with immersive mixed reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5962–5969. IEEE, 2023.

[27] R. Ding, Y. Qin, J. Zhu, C. Jia, S. Yang, R. Yang, X. Qi, and X. Wang. Bunny-visionpro: Real-time bimanual dexterous teleoperation for imitation learning. *arXiv preprint arXiv:2407.03162*, 2024.

[28] F. P. Audonnet, I. G. Ramirez-Alpizar, and G. Aragon-Camarasa. Immertwin: A mixed reality framework for enhanced robotic arm teleoperation. *arXiv preprint arXiv:2409.08964*, 2024.

[29] K. A. Szczurek, R. M. Prades, E. Matheson, J. Rodriguez-Nogueira, and M. Di Castro. Multimodal multi-user mixed reality human–robot interface for remote operations in hazardous environments. *IEEE Access*, 11:17305–17333, 2023.

[30] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[31] Z. Fu, T. Z. Zhao, and C. Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In *Conference on Robot Learning (CoRL)*, 2024.

[32] P. Wu, Y. Shentu, Z. Yi, X. Lin, and P. Abbeel. Gello: A general, low-cost, and intuitive teleoperation framework for robot manipulators. *arXiv preprint arXiv:2309.13037*, 2023.

[33] Y. Qin, W. Yang, B. Huang, K. V. Wyk, H. Su, X. Wang, Y.-W. Chao, and D. Fox. Anyteleop: A general vision-based dexterous robot arm-hand teleoperation system. *ArXiv*, abs/2307.04577, 2023. URL https://api.semanticscholar.org/CorpusID:259367735.

[34] J. Duan, Y. R. Wang, M. Shridhar, D. Fox, and R. Krishna. Ar2-d2: Training a robot without a robot. In J. Tan, M. Toussaint, and K. Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2838–2848. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/duan23a.html.

[35] J. Wang, C.-C. Chang, J. Duan, D. Fox, and R. Krishna. Eve: Enabling anyone to train robots using augmented reality. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, UIST '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706288. doi:10.1145/3654777.3676413. URL https://doi.org/10.1145/3654777.3676413.

[36] S. Arevalo Arboleda, F. Rücker, T. Dierks, and J. Gerken. Assisting manipulation and grasping in robot teleoperation with augmented reality visual cues. In *Proceedings of the 2021 CHI conference on human factors in computing systems*, pages 1–14, 2021.

[37] X. Wang, S. Guo, Z. Xu, Z. Zhang, Z. Sun, and Y. Xu. A robotic teleoperation system enhanced by augmented reality for natural human–robot interaction. *Cyborg and Bionic Systems*, 5:0098, 2024.

[38] S. P. Arunachalam, I. Güzey, S. Chintala, and L. Pinto. Holo-dex: Teaching dexterity with immersive mixed reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5962–5969. IEEE, 2023.

[39] S. Chen, C. Wang, K. Nguyen, L. Fei-Fei, and C. K. Liu. Arcap: Collecting high-quality human demonstrations for robot learning with augmented reality feedback. *arXiv preprint arXiv:2410.08464*, 2024.

[40] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[41] ZeroMQ — zeromq.org. https://zeromq.org/. [Accessed 01-05-2025].

[42] lolambean. Hololens 2 hardware, Mar. 2023. URL https://learn.microsoft.com/en-us/hololens/hololens2-hardware.

[43] Meta Quest 3 - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Meta_Quest_3. [Accessed 01-05-2025].

[44] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020.

[45] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[46] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024.

[47] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

[48] F. Otto, O. Celik, D. Roth, and H. Zhou. Fancy gym. URL https://github.com/ALRhub/fancy_gym.

[49] S. James, M. Freese, and A. J. Davison. Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176*, 2019.

[50] W. Pumacay, I. Singh, J. Duan, R. Krishna, J. Thomason, and D. Fox. The colosseum: A benchmark for evaluating generalization for robotic manipulation. *arXiv preprint arXiv:2402.08191*, 2024.

[51] A. Tung, J. Wong, A. Mandlekar, R. Martín-Martín, Y. Zhu, L. Fei-Fei, and S. Savarese. Learning multi-arm manipulation through collaborative teleoperation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9212–9219. IEEE, 2021.

[52] F. Otto, O. Celik, H. Zhou, H. Ziesche, V. A. Ngo, and G. Neumann. Deep black-box reinforcement learning with movement primitives. In *Conference on Robot Learning*, pages 1244–1265. PMLR, 2023.

[53] F. Otto, H. Zhou, O. Celik, G. Li, R. Lioutikov, and G. Neumann. Mp3: Movement primitive-based (re-) planning policy. *arXiv preprint arXiv:2306.12729*, 2023.

[54] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, et al. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, pages 879–893. PMLR, 2018.

[55] J. Luo, C. Xu, J. Wu, and S. Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *arXiv preprint arXiv:2410.21845*, 2024.

[56] A. Mandlekar, C. R. Garrett, D. Xu, and D. Fox. Human-in-the-loop task and motion planning for imitation learning. In *Conference on Robot Learning*, pages 3030–3060. PMLR, 2023.

[57] K. Finstad. The usability metric for user experience. *Interacting with computers*, 22(5):323–327, 2010.

[58] Kruskal–Wallis test - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Kruskal%E2%80%93Wallis_test. [Accessed 27-04-2025].

[59] X. Jia, D. Blessing, X. Jiang, M. Reuss, A. Donat, R. Lioutikov, and G. Neumann. Towards diverse behaviors: A benchmark for imitation learning with human demonstrations. *arXiv preprint arXiv:2402.14606*, 2024.

[60] M. Reuss, M. Li, X. Jia, and R. Lioutikov. Goal-conditioned imitation learning using score-based diffusion policies. *arXiv preprint arXiv:2304.02532*, 2023.

[61] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2024.

[62] M. Reuss, Ö. E. Yağmurlu, F. Wenzel, and R. Lioutikov. Multimodal diffusion transformer: Learning versatile behavior from multimodal goals. *arXiv preprint arXiv:2407.05996*, 2024.

[63] H. Zhou, W. Liao, X. Huang, Y. Tang, F. Otto, X. Jia, X. Jiang, S. Hilber, G. Li, Q. Wang, et al. Beast: Efficient tokenization of b-splines encoded action sequences for imitation learning. *arXiv preprint arXiv:2506.06072*, 2025.

[64] N. Bell and J. Hoberock. Thrust: A productivity-oriented library for cuda. In *GPU computing gems Jade edition*, pages 359–371. Elsevier, 2012.

[65] NVIDIA Corporation. NVIDIA Isaac Sim, 2024. URL https://developer.nvidia.com/isaac-sim.

[66] NVIDIA Corporation. NVIDIA Omniverse, 2024. URL https://www.nvidia.com/en-us/omniverse/.

[67] R. Gong, J. Huang, Y. Zhao, H. Geng, X. Gao, Q. Wu, W. Ai, Z. Zhou, D. Terzopoulos, S.-C. Zhu, et al. Arnold: A benchmark for language-grounded task learning with continuous states in realistic 3d scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[68] P. A. Studios. Universal scene description (usd). https://github.com/PixarAnimationStudios/USD, 2016. Accessed: 2025-01-29.

[69] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

[70] trimesh - trimesh 4.6.1 documentation — trimesh.org. https://trimesh.org/trimesh.html. [Accessed 01-02-2025].

[71] S. Wrede, C. Emmerich, R. Grünberg, A. Nordmann, A. Swadzba, and J. Steil. A user study on kinesthetic teaching of redundant robots in task and configuration space. *Journal of Human-Robot Interaction*, 2(1):56–81, 2013.

[72] F. Sukkar, V. H. Moreno, T. Vidal-Calleja, and J. Deuse. Guided learning from demonstration for robust transferability. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5048–5054. IEEE, 2023.

[73] A. Pettinger, C. Elliott, P. Fan, and M. Pryor. Reducing the teleoperator's cognitive burden for complex contact tasks using affordance primitives. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11513–11518. IEEE, 2020.

[74] T.-C. Lin, A. U. Krishnan, and Z. Li. Comparison of haptic and augmented reality visual cues for assisting tele-manipulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 9309–9316. IEEE, 2022.

[75] SteamVR — store.steampowered.com. https://store.steampowered.com/steamvr. [Accessed 31-01-2025].

[76] three.js docs — threejs.org. https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene. [Accessed 30-01-2025].

# A IRIS Framework

## A.1 Node Communication Protocol

The IRIS system operates across simulation and/or sensor-processing computers, multiple XR headsets, and other monitoring and control programs, requiring a robust and reliable network connection between them. All devices are part of the same subnet, with all the devices connected via Wi-Fi or cable. Different from Robot Operating System (ROS [40]), IRIS leverages a local network across multiple hosts instead of using a single host, while using both Request-Response and Publish-Subscribe patterns for data transmission. This protocol follows a master-node architecture, where the simulation PC serves as the master node, and the XR headsets and other devices act as XR nodes. Communication is achieved through a combination of UDP sockets and ZeroMQ (ZMQ).

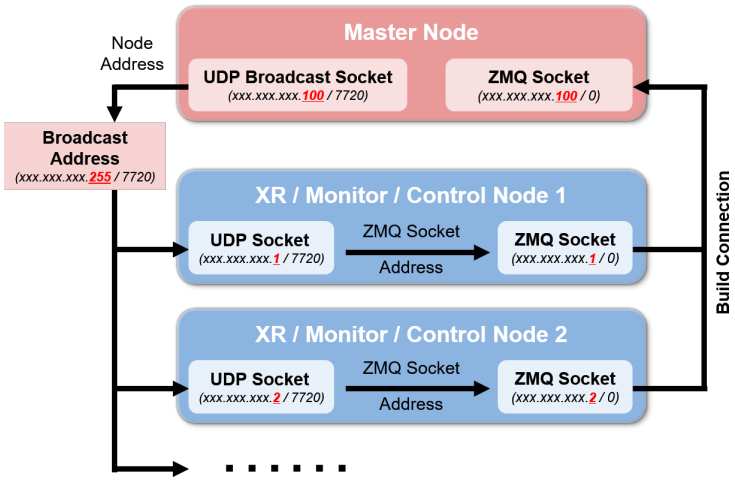



Figure 10: The master node broadcasts UDP messages containing its details to the broadcast address (e.g., *192.168.0.255*) at port 7720. Each IP address in the diagram (e.g., *192.168.0.100/0*) represents a device's unique address on the network, where '0' indicates a dynamically assigned port provided by the operating system. XR nodes, upon startup, listen on the broadcast port (7720) to receive these messages, extract the master node's IP and ZMQ socket address, and build a stable connection. This architecture supports both request-response and publish-subscribe communication patterns, ensuring robust, multi-device connectivity with automatic reconnection capabilities.

To ensure node discovery, the master node broadcasts UDP messages at 5 Hz to a fixed broadcast port on the network. When a new XR node is launched, it listens on the broadcast port to receive a broadcast message from the master node. Upon receiving the broadcast message, the XR node extracts the master node's details, including its ZMQ socket address and port, to establish a reliable ZMQ connection. If the master node goes offline, XR nodes continue listening on the discovery port, allowing automatic reconnection when the master node relaunches. This protocol (Fig. 10) achieves **Cross-User** ability of IRIS, ensures reliable communication, automatic reconnection, and smooth recovery from disconnections, making it ideal for dynamic multi-device XR systems.

## A.2 Unified Scene Specification

To visualize a scene with arbitrary objects, the XR headsets need to receive the scene model from the simulation and reconstruct it. However, the XR application and the simulation run on different devices and use different software architectures (the XR application is developed with C# and Unity, while simulators might be built in Python or C++). This makes it impractical to directly transfer the scene from the simulation to the XR environment. To address this issue, existing solutions rely on predefined models in the XR application for specific robots and assets, requiring a static set of

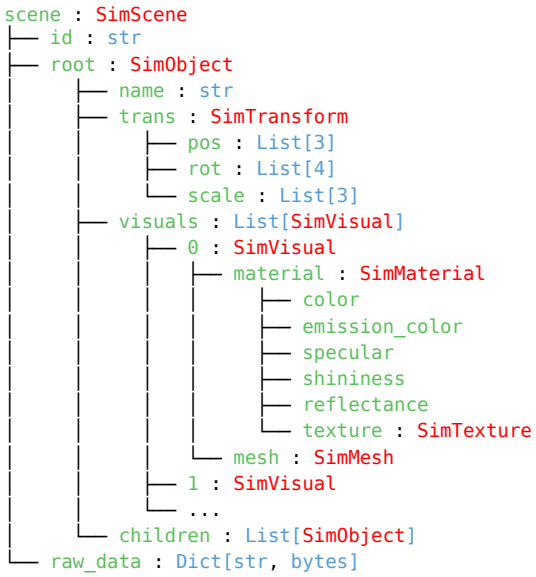



Figure 11: The hierarchical structure of the Scene Specification begins with a root SimObject, which contains all objects in the scene. Each SimObject has a name, a list of child SimObjects, and a list of visuals. Each visual represents a geometric element attached to the object. Within each geometric element, materials define properties such as color and texture, while meshes determine the shape. The scene's raw data includes the byte streams of meshes and textures. Since these streams are extensive, they are sent to XR headsets sequentially after the initial scene specification is transmitted.

models to be maintained within the application. This approach restricts flexibility and reusability, preventing the support of robots or objects not included in the model set. To address this issue, IRIS introduces a novel unified scene specification, which is generated by parsing the scene directly from the simulation. This specification is subsequently transmitted to the headsets using the node communication protocol, enabling the XR application to accurately and dynamically recreate the scene in real time.

The unified scene specification (shown in Fig. 11) includes all objects with their geometry, meshes, materials, and textures. IRIS provides a Python library called *ScenePublisher* to parse a simulation scene into this scene specification. In the specification, geometry defines the object's shape (e.g., cube, sphere, capsule, cylinder, or mesh). The mesh contains vertex lists, face lists, normals, and texture coordinates. Materials define surface properties, including color, emission color, reflectance, and attached textures, and the texture is an image representing the object's surface appearance. Objects are organized using a kinematic tree structure and are serialized into JSON by the *ScenePublisher*.

Since meshes and textures contain large amounts of data, geometry and materials store only their hash code to reduce the transmission load. The XR application rebuilds the scene upon receiving the kinematic tree and then requests the meshes and textures from the simulation server in byte format. In some cases, textures can be quite large (e.g., the textures for the RoboCasa [46] scene exceed 700 MB). To ensure the scene loads within an acceptable time for users, we compress the textures. This compression reduces the loading time to a few seconds. Afterwards, the *ScenePublisher* continually acquires simulation states and forwards them to the XR headset. This way the positions and rotations of all the objects are updated at a fixed frequency.

The scene specification enables IRIS to support a wide range of robots and objects in simulation, facilitating both **Cross-Scene** and **Cross-Embodiment** capabilities.

### A.3 Extendable and Flexible Framework Support

The unified scene specification is a general definition that does not rely on any specific simulator, providing an extensible mechanism for scene loading and updating. IRIS can be easily adapted to various simulation engines and frameworks by implementing a new simulation parser to generate the unified specification from the simulation scene and a new publisher to update the states of scene.

Currently, IRIS supports scene parsers for MuJoCo, IsaacSim, CoppeliaSim, and Genesis, with the potential to be extended to other simulation engines as desired. IRIS has been tested in some MuJoCo-based benchmarks including **Meta World** [44], **LIBERO** [45], **RoboCasa** [46], **robosuite** [47], **Fancy Gym** [48], and CoppeliaSim-based benchmark like **PyRep** [49], **Colosseum** [50]. This demonstrates that IRIS can be easily adapted to various benchmarks and simulators, highlighting its **Cross-Simulator** capability.

IRIS provides a user-friendly API. For each environment or framework, a single line of code suffices to visualize and update the simulation in the XR headset. Here is a short example of how to use it in the MuJoCo Simulation:

```
# import scenepub
from scenepub.sim.mj_publisher import MujocoPublisher
# define the mujoco environment
model = mujoco.MjModel.from_xml_path(xml_path)
data = mujoco.MjData(model)

# define the ScenePublisher for mujoco; only this line needs to be added
publisher = MujocoPublisher(model, data, host)

# run simulation
while True:
    # run simulation step logic
    pass
```

The MujocoPublisher instance only needs to access the model and data from the MuJoCo simulation. It then creates a separate thread to run the communication protocol, automatically connecting and communicating with all available XR headsets. IRIS provides various Simulation Publishers for different environments, all with a consistent and easy-to-use interface. This mechanism ensures a seamless experience, making the system very user-friendly.

### A.4 Real Scene Loading

The loading and updating of real-world scenes in IRIS follow a process similar to that of simulation scenes, demonstrating its **Cross-Reality** capability. It processes point clouds from one or multiple RGB-D cameras, which are extrinsically calibrated to a fiducial marker in the scene. A point cloud processor applies the extrinsic transformation to each point cloud before merging them. The merged point cloud is then cropped and downsampled using a voxel-grid filter. This filter maps all 3D points to voxel-grid indices, blends the colors of points within the same voxel, and calculates the voxels centroid. The output is a reduced point cloud that retains both color and position data. To ensure low latency, this process runs on the GPU using Thrust [64]. Finally, the processed point cloud is transmitted to the headsets, where IRIS uses a particle system in Unity to visualize and dynamically update the scene in real-time.

## B Simulator Support and Real-world Data Collection

### B.1 Mujoco

MuJoCo [19] is a fast, accurate physics engine ideal for simulating robots with complex joint structures and contact dynamics. It supports advanced robot simulations with features like customizable actuators, collision detection, and friction modeling, enabling realistic testing of robotic control, manipulation, and learning algorithms in dynamic environments.

When starting a MuJoCo simulation, MuJoCo provides a model instance and a data instance. The scene specification of IRIS can be generated from the model, while the simulation states can be retrieved from the data instance. The model instance contains all the necessary assets for the scene specification, including meshes, textures, and materials. These assets are stored as NumPy arrays and need to be converted to byte streams with data types such as *float32* or *int8* for transmission.

MuJoCo uses a standard robotic coordinate system, where X is forward, Y is left, and Z is up. This differs from Unity's coordinate system, where X is right, Y is up, and Z is forward. Therefore, object transforms, mesh vertices, faces, and normal data must be adapted for Unity using the following code:

```
def mj2unity_pos(pos: List[float]) -> List[float]:
    return [-pos[1], pos[2], pos[0]]

def mj2unity_quat(quat: List[float]) -> List[float]:
    return [quat[2], -quat[3], -quat[1], quat[0]]
```

Since MuJoCo includes visual group settings that specify which object groups can be visualized, IRIS provides an API to support this feature through the *MujocoPublisher* definition, as shown below:

```
class MujocoPublisher(ScenePublisher):
    def __init__(
        self,
        mj_model,
        mj_data,
        host: str = "127.0.0.1",
        no_rendered_objects: Optional[List[str]] = None,
        no_tracked_objects: Optional[List[str]] = None,
        visible_geoms_groups: Optional[List[int]] = None,
    ) -> None:
```

### B.2 IsaacSim

IsaacSim [65] is a robotics simulation environment based on the NVIDIA Omniverse platform [66]. Several frameworks [20, 67] are built on top of it, sharing the same underlying data structures. IsaacSim supports ray-tracing for realistic rendering, and batched physics simulation on GPUs, significantly accelerating the training of models.

The scenes in IsaacSim are organized in a Universal Scene Description (USD) format, and data can be accessed directly with the OpenUSD API [68]. The scene hierarchy consists of transformations (Xform), meshes, articulations (joints) among other elements. Fig. 12 illustrates a sample scene hierarchy.

Besides the low-level OpenUSD API, IsaacSim also provides some utility APIs for accessing scene data. In our system, we use a mix of both APIs. A (simplified) code snippet for accessing parsing the tree structure of the USD scene is given below. We use the OpenUSD API `GetChildren()` to retrieve child primitives here.

```
# compute local transforms of the current prim
trans, rot, scale = self.compute_local_trans(root)
# parse material and geometry
mat_info = self.parse_prim_material(prim=root)
self.parse_prim_geometries(
    prim=root,
    prim_path=prim_path,
    sim_obj=sim_object,
    mat_info=mat_info or inherited_material,
)
# parse children of the current prim
for child in root.GetChildren():
    if obj := self.parse_prim_tree(
        root=child,
        parent_path=prim_path,
        inherited_material=mat_info or inherited_material,
    ):
        sim_object.children.append(obj)
```

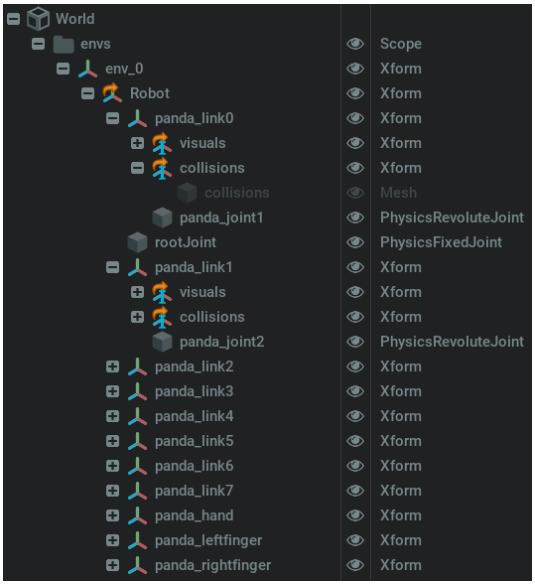



Figure 12: An example of USD scene hierarchy for Franka Panda robot arm in IsaacSim.

Here is another snippet for computing the world transform of a primitive in the hierarchy. Instead of computing the world transform with OpenUSD API, the class in IsaacSim API `XFormPrim` is used to simplify the code.

```
from pxr import Usd
from omni.isaac.core.prims import XFormPrim

def compute_world_trans(self, prim: Usd.Prim):
    prim = XFormPrim(str(prim.GetPath()))
    assert prim.is_valid()

    pos, quat = prim.get_world_pose()
    scale = prim.get_world_scale()

    return (
        pos.cpu().numpy(),
        quat_to_rot_matrix(quat.cpu().numpy()),
        scale.cpu().numpy(),
    )
```

### B.3 CoppeliaSim

CoppeliaSim, formerly known as V-REP, is a versatile and widely used robot simulation software that supports various physics engine backbones. Scenes in CoppeliaSim are constructed using a tree structure, which includes objects such as visual shapes, dynamic shapes, and joints, as illustrated in Fig. 13.

In our system, we interact with CoppeliaSim using Python APIs. Depending on the version of CoppeliaSim, we utilize two groups of Python API functions:

**ZeroMQ Remote API** [3]. This is an official API introduced in version 4.4. It facilitates fast and straightforward communication between the simulator and a Python script, enabling users to efficiently build scenarios. For example, to retrieve mesh information for all shape objects in a CoppeliaSim scene, one can use the following call:

```
from coppeliasim_zmqremoteapi_client import RemoteAPIClient
client = RemoteAPIClient()
sim = client.require('sim')

list_vertices, list_indices, list_normals = [], [], []
```

[3] https://manual.coppeliarobotics.com/en/apiFunctions.htm

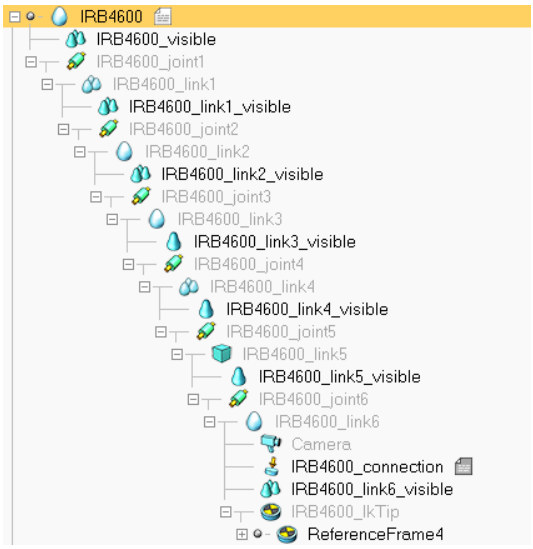



Figure 13: An example of a tree structure for building an ABB IRB4600 manipulator.

```
objects_id_list = sim.getObjectsInTree(sim.handle_scene,
                                       sim.handle_all, 0)

for idx in objects_id_list:
    if sim.getObjectType(idx) == sim.sceneobject_shape:
        # We assume all the shapes are primary shapes
        vertices, indices, normals = sim.getShapeMesh(idx)

        list_vertices.append(vertices)
        list_indices.append(indices)
        list_list_normals.append(normals)
```

Listing 1: Example ZeroMQ Remote API.

**PyRep API**[49] [4]. This is a widely-used third-party Python communication interface under the CoppeliaSim version 4.1. Several notable CoppeliaSim projects and benchmarks, such as RLBench [69], are built on this interface. Similar to the official interface, PyRep provides various API functions and properties. For instance, to retrieve all object handles from a CoppeliaSim scene, one can use the following call:

```
from pyrep.backend.sim import simGetObjectsInTree
from pyrep.backend.sim import simGetObjectType
from pyrep.backend.sim import simGetShapeMesh
from pyrep.backend.simConst import sim_handle_scene
from pyrep.backend.simConst import sim_handle_all
from pyrep.backend.simConst import sim_object_shape_type

list_vertices, list_indices, list_normals = [], [], []

objects_id_list = simGetObjectsInTree(sim_handle_scene,
                                      sim_handle_all, 0)

for idx in objects_id_list:
    if simGetObjectType(idx) == sim_object_shape_type:
        # We assume all the shapes are primary shapes
        vertices, indices, normals = simGetShapeMesh(idx)

        list_vertices.append(vertices)
        list_indices.append(indices)
        list_list_normals.append(normals)
```

Listing 2: Example PyRep API.

[4] https://github.com/stepjam/PyRep

### B.4 Genesis

Genesis [22] is a versatile physics platform designed for robotics, embodied AI, and physical AI applications. It combines a universal, re-engineered physics engine capable of simulating diverse materials and phenomena with a lightweight, ultra-fast, and user-friendly robotics simulation environment. It also features a powerful, photorealistic rendering system and a generative data engine that transforms natural language prompts into multi-modal data. By integrating various physics solvers within a unified framework, Genesis supports automated data generation through a generative agent framework, with its physics engine and simulation platform now open-source and further expansions planned.

IRIS supports Genesis by providing a Genesis parser designed to translate simulation data from the Genesis physics platform into the IRIS framework's internal representation. It manages this by reading rigid entities, links, and geometries from a *gs.Scene* object and converting them into equivalent IRIS structures such as *SimScene*, *SimObject*, *SimVisual*, and *SimMaterial*. Here's a detailed breakdown of the parsing process and its components:

The main function initializes a *SimScene* and iterates through all entities in the Genesis scene. It builds a hierarchical scene graph based on parent-child relationships between entities and links. The graph is stored in the hierarchy dictionary, which tracks each object's parent and child nodes.

Each rigid entity (*RigidEntity*) from Genesis is processed to create a *SimObject*. The position and rotation of the entity are retrieved and transformed is similart to Mujoco (B.1) to ensure compatibility with Unity's coordinate system. If the Genesis scene is already built, the function pulls actual position and rotation data; otherwise, it defaults to identity transforms.

Links (RigidLink) represent individual parts of a rigid entity's structure. Each link is converted into a *SimObject* and positioned within the scene based on its parent link's position and orientation. If the link has associated visual geometries (*vgeoms*), the parser processes each visual element to create *SimVisual* objects, which store geometry and material data. Links without visual elements are simply added to the scene hierarchy without rendering.

Visual geometries (*RigidGeom*) from Genesis are mapped to IRIS visual representations. The parser constructs a *SimVisual* object, including position, rotation, and mesh data. The mesh generation function extracts vertex and face data from the Genesis mesh and constructs a *SimMesh* object for the IRIS scene, defining both the physical and visual structure of the geometry. Genesis leverages the *Trimesh* [70] library to handle mesh processing, which simplifies the conversion to the IRIS mesh format.

After processing all entities, the parser organizes the parsed objects into a tree structure. Objects without parents are attached to the root of the scene, while others are linked to their respective parents based on the hierarchy dictionary.

### B.5 Real World

The point cloud-based XR teleoperation system enables immersive teleoperation and data collection by allowing users to interact with a real-world robotic setup through an Extended Reality (XR) interface. Unlike conventional video-streaming methods, this approach provides spatial awareness and depth perception by rendering real-time 3D reconstructions of the environment in XR. The system consists of an ORBBEC Femto Bolt depth camera positioned in front of the robot to capture RGB and depth data in real time. A QR code on the robot's end effector serves as a reference marker for pose estimation, aligning point clouds with the robot's frame. A C++-based point cloud processing pipeline converts raw data into XYZ-RGB point clouds at 30Hz, generating approximately 2 million points per frame. To optimize computational efficiency, GPU-accelerated voxel grid downsampling is applied before transmission. The processed point clouds are sent to a main process that handles transformation, cropping, and communication with the XR system. A ZeroMQ (ZMQ) messaging protocol is used to enable efficient, low-latency communication between system components, ensuring real-time transmission of point clouds and control commands. A Meta Quest 3 headset renders

the processed point clouds in real time, enabling users to visualize and interact with the scene in a fully immersive 3D environment.

### B.5.1 Camera-to-Robot Base Transformation

**Pose Estimation and Homogeneous Transformation** To ensure accurate spatial alignment, the system transforms the camera's coordinate frame into the robot's base coordinate frame using a homogeneous transformation matrix, computed as follows:

- The QR code on the robot's end effector provides a fixed reference point for tracking position and orientation in the camera's space.
- Using Forward Kinematics (FK), the end effector's pose relative to the robot base is determined.
- The camera-to-robot base transformation is derived as:

$$T_{\text{RobotBase}}^{\text{Camera}} = T_{\text{RobotBase}}^{\text{EndEffector}} \cdot T_{\text{EndEffector}}^{\text{Camera}} \tag{1}$$

  where:
  - $T_{\text{RobotBase}}^{\text{EndEffector}}$ is obtained from the robot's FK.
  - $T_{\text{EndEffector}}^{\text{Camera}}$ is estimated using the QR code tracking system.

This ensures that the captured point clouds align precisely with the robot's coordinate frame, eliminating drift and inconsistencies in XR visualization.

### B.5.2 Point Cloud Processing Pipeline

**Raw Data Acquisition** The ORBBEC Femto Bolt camera captures:

- Depth images (encoded as a depth map),
- RGB images (color information),
- Camera intrinsic parameters (for depth-to-3D conversion).

**Conversion to XYZ-RGB Point Cloud** Using the camera intrinsics, each depth pixel is converted into 3D world coordinates $(X, Y, Z)$ using:

$$X = (u - c_x)\frac{Z}{f_x}, \quad Y = (v - c_y)\frac{Z}{f_y}, \quad Z = D(u, v) \tag{2}$$

where:

- $(u, v)$ are pixel coordinates,
- $(c_x, c_y)$ are the camera's principal point offsets,
- $(f_x, f_y)$ are focal lengths,
- $D(u, v)$ is the depth value at pixel $(u, v)$.

Each $(X, Y, Z)$ point is assigned an $(R, G, B)$ value from the color image, forming the XYZ-RGB point cloud.

**Cropping and Filtering** To reduce noise and retain only relevant portions of the workspace, the system:

- Crops unnecessary regions using bounding box constraints,
- Applies statistical outlier removal to eliminate noise points.

**Voxel Grid Downsampling (GPU-Accelerated)** Since the raw point cloud consists of approximately 2 million points per frame, direct transmission is computationally expensive. To optimize efficiency, voxel grid downsampling is applied, which:

- Divides the workspace into 3D voxels,
- Averages all points within each voxel to generate a single representative point,
- Reduces the point cloud size while preserving structural details.

#### B.5.3 XR Visualization and Teleoperation

**Real-Time Communication with XR System** The processed point cloud is transmitted to a main process, which then relays the data to the Meta Quest 3 headset using ZeroMQ. ZeroMQ ensures efficient, asynchronous, and low-latency messaging between the processing node and the XR system, allowing:

- Real-time rendering of the scene in XR,
- Full 3D immersion with accurate depth perception,
- Reliable transmission of point cloud data and control commands.

**Leader-Follower Teleoperation Framework** The teleoperation system consists of:

- A leader robot controlled by a human wearing the Meta Quest 3 headset,
- A follower robot that replicates the leader's movements, including gripper actions.

A networked control architecture ensures:

- Low-latency synchronization between the leader and follower,
- Seamless remote manipulation, removing the need for physical human presence.

#### B.5.4 Calibration for XR-Robot Alignment

To ensure that XR visualization aligns precisely with real-world objects, the system undergoes a calibration process consisting of:

- Robot workspace alignment: The follower robot's workspace is adjusted to match the leader's XR-rendered environment,
- Point cloud adjustment: The camera-to-robot transformation is fine-tuned,
- User feedback correction: Users verify object positions in XR and real-world views.

#### B.5.5 Future Extensions: Multi-Camera and ICP-Based Fusion

Future implementations will:

- Incorporate multiple cameras for wider coverage,
- Perform point cloud fusion using Iterative Closest Point (ICP) for improved spatial accuracy,
- Enhance depth perception by reconstructing high-fidelity 3D representations of the workspace.

## C Intuitive Robot Control Interface

In data collection tasks, robot control interfaces are used to operate the robot in both simulated and real-world environments. Based on research in teleoperation and robot data collection [7],

Kinesthetic Teaching and Motion Controllers have been identified as the most intuitive and effective control interfaces. Hence, we ensured that IRIS supports these two methods. Thanks to IRIS's flexible framework, it is possible to easily customize and implement additional alternative control interfaces, such as hand tracking, gloves, smartphones, or motion tracking systems. This adaptability enables tailored solutions to meet specific requirements, enhancing both the usability and versatility of the system for various applications. Fig. 14 shows how these two interfaces work in IRIS. The implementation of these interfaces is outlined below.

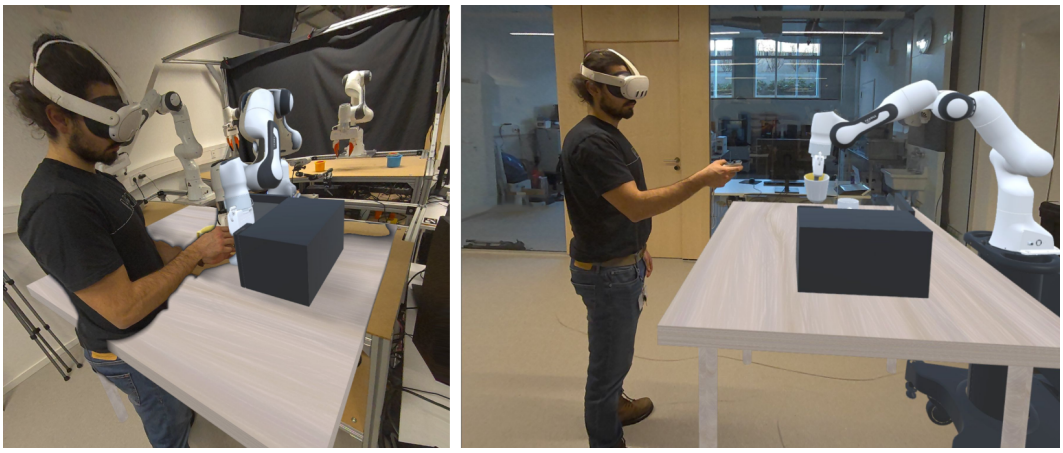


Figure 14: This image illustrates examples of using two interfaces to control robots in simulation: Kinesthetic Teaching (left) and Motion Controller (right), shown from a third-person perspective.

### C.1 Kinesthetic Teaching

Kinesthetic teaching is an intuitive interface that allows users to control robots by physically moving the real robot [71, 72, 7]. The real robot transmits joint positions and velocities in real time to the controlled robot, which can be either a virtual or another physical robot. In both real-world and simulation data collection, the key aspect of kinesthetic teaching is ensuring alignment between the real robot and its virtual counterpart in the XR headsets. By utilizing *Spatial Anchors* (C.4), the virtual robot in the XR headsets can be perfectly aligned with the real robot, which provide users with an intuitive and immersive experience.

### C.2 Motion Controller

Motion controller Interfaces are commonly used in robot data collection and teleoperation [73, 74], providing stable tracking and flexibility for various applications. Although some XR headsets, such as HoloLens 2, do not natively support motion controllers, this limitation can be resolved by integrating third-party controllers compatible with platforms such as SteamVR [75]. Motion controllers utilize inverse kinematics to control robots in Cartesian space, with the movement of the robot's end effector controlled by the controller's trigger. IRIS supports retrieval of motion controller data from devices like the Meta Quest 3. For more complex scenarios, users can design custom controllers using IRIS' flexible framework.

### C.3 Affiliated Monitor Tools

The extensibility of IRIS opens up numerous possibilities for creating new applications. IRIS includes a web-based monitoring tool for managing all XR headsets. This tool allows users to easily start and stop alignment processes, as well as rename devices. Additionally, the tool supports real-time scene visualization using *three.js* [76], using the unified scene specification. An example screenshot is shown in Fig. 15.

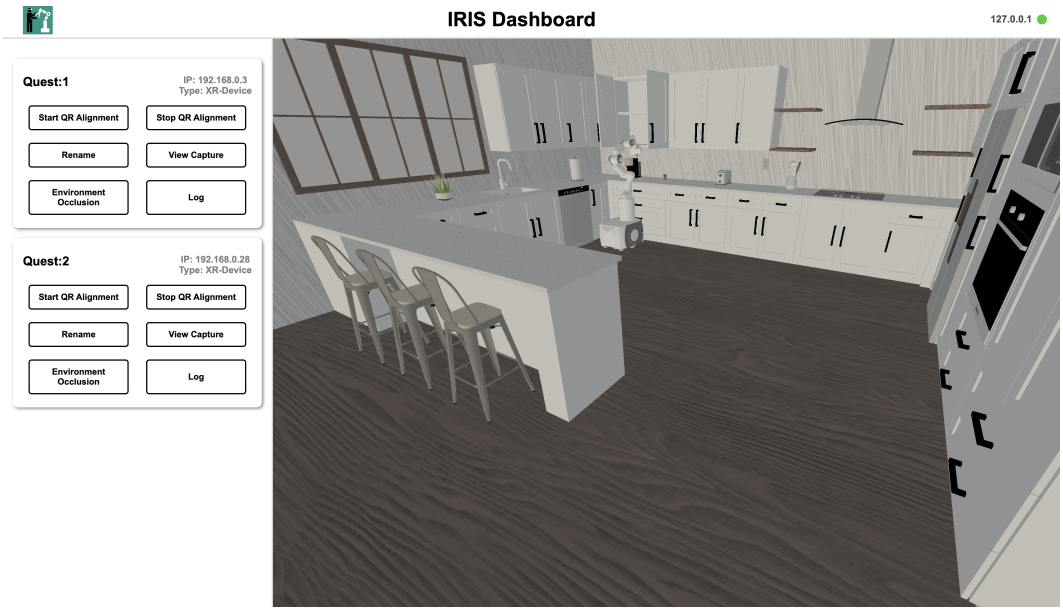



Figure 15: The IRIS Dashboard, accessible via a web interface, allows the control and monitoring of all connected nodes. The scene streamed by IRIS is rendered on the right-hand side. The left panel displays the connected XR devices, providing an interface through which users can control all services made available by each device.

### C.4 Spatial Anchor

A spatial anchor serves as a reference point in a 3D environment to accurately position virtual objects within physical space. It enables virtual objects to maintain their position, orientation, and alignment, even as users move around or leave and return to the environment. IRIS utilizes either QR codes or motion controllers (currently only implemented for the Meta Quest 3) as spatial anchors. This approach allows for the alignment of augmented scenes across multiple headsets in the real world, making it appear as though all users are sharing the same scene. Additionally, this method can synchronize a virtual robot with its real-world counterpart in the Kinesthetic Teaching (C.1) interface and facilitate multiple-collector view alignment. Fig. 16 shows the alignment between real robot and virtual robot.

## D Experiment Details

### D.1 User Study

In this section, we provide more details of the user study we conducted, which assesses different data collection interfaces. As mentioned in the main paper, tasks from the LIBERO benchmark [45] were chosen due to their diversity in various movement patterns. Four representative tasks (see Fig. 17) were selected in the dimension of translation, rotation, and compound movement:

- Task 1: *close the microwave*
- Task 2: *turn off the stove*
- Task 3: *pick up the book in the middle and place it on the cabinet shelf*
- Task 4: *turn on the stove and put the frying pan on it*.

To ensure a fair comparison, the tasks and data were taken directly from LIBERO [45] without any modification. The baselines for this study were two standard control interfaces provided by LIBERO: the Keyboard (KB) and the 3D Mouse (3M). Based on findings from prior research [7],

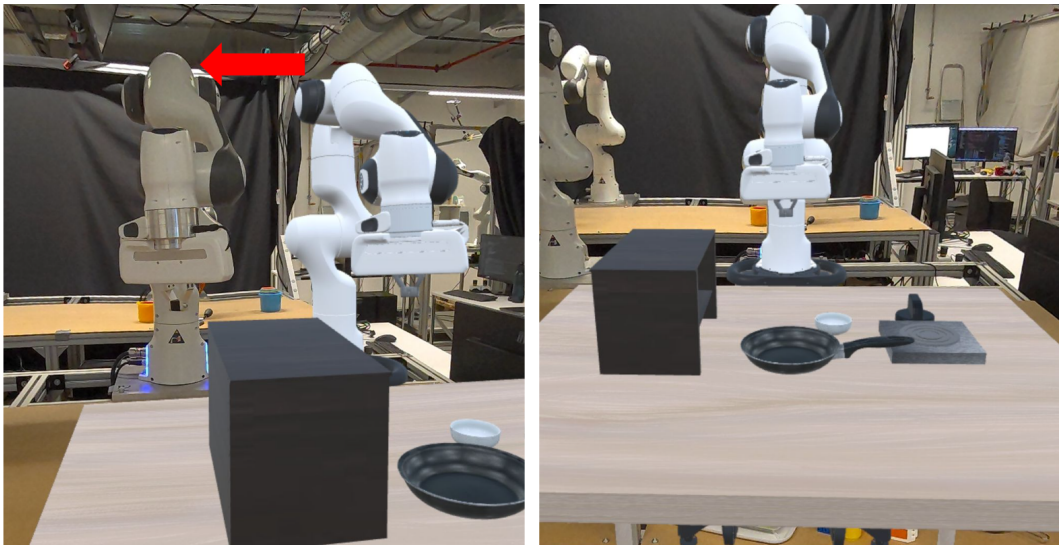


Figure 16: Alignment between real robot and virtual robot with a spatial anchor. It is used for controlling virtual robot by real world kinesthetic teaching or scene sharing among multiple users.

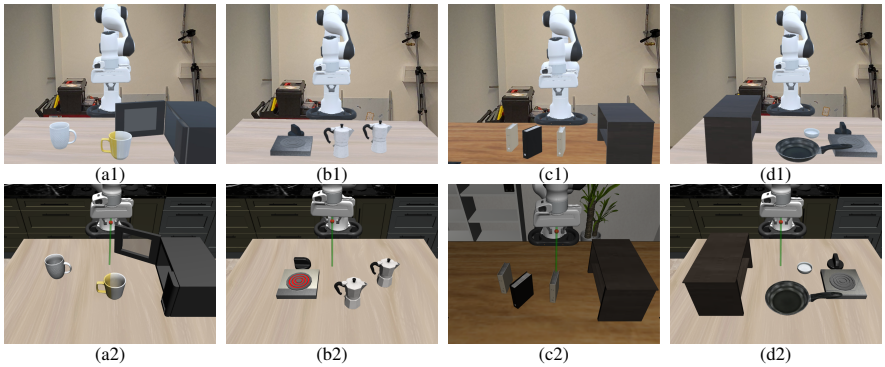



Figure 17: Four tasks from LIBERO in simulation (top row: a1, b1, c1, d1) and corresponding view from Meta Quest 3 (bottom row: a2, b2, c2, d2): (a) Close the microwave, (b) Turn off the stove, (c) Pick up the book and place it on the shelf, (d) Turn on the stove and place the frying pan on it.

hand tracking was found to be less stable than motion controllers. Therefore, we selected Kinesthetic Teaching and Motion Controller as the interfaces for the user study.

The study involved eight participants who evaluated the efficiency and intuitiveness of each interface for collecting demonstrations using both objective and subjective metrics. The objective metrics included the success rate and the average time taken per task. To ensure successful demonstrations, participants executed tasks at a very slow pace, which diluted efficiency measurements (as all interfaces appeared efficient when tasks were performed slowly), which biased participants against later interfaces [7]. To mitigate these biases and ensure high-quality data collection, a time limit per task was introduced, and the time limits for four tasks are 20s, 20s, 30s, and 40s, which are quite enough for finishing the task. The subjective metrics were assessed through a questionnaire evaluating four dimensions: *Experience*, *Usefulness*, *Intuitiveness*, and *Efficiency*. Each participant performed each task five times using all interfaces. After finishing using one interface, they provided ratings on the subjective dimensions using a 7-point Likert scale.

For the objective metrics, Table 2 presents the success rates for each interface across the tasks. A trial was considered unsuccessful if the participant failed to complete the task or exceeded the time limit. The time limits were set based on task difficulty: 20 seconds for Task 1 and Task 2, 30 seconds

for Task 3, and 40 seconds for Task 4. The data shows a success rate of over $90\%$ across all four tasks when using the KT and MC interfaces from IRIS. In contrast, the Keyboard and 3D Mouse methods from LIBERO often resulted in failures. For example, the 3D Mouse interface achieved only a $37.5\%$ success rate on Task 3.

As is clear in Fig. 6, the results of this user study show that IRIS received higher scores than the baseline interfaces across both objective and subjective metrics. This indicates that the system offers a more intuitive and efficient approach for data collection.

### D.2 Deformable Object Data Collection

Demonstrations involving deformable objects and cloth as seen in Fig. 18 can be recorded when using IsaacLab for physics simulation. Each point of a simulated deformable object is transmitted to IRIS. This allows for an accurate representation of object deformation during simulation. Transmitting each point individually adds a performance penalty compared to rigid objects. IRIS provides hand-tracking information that is used to control simulated robots and manipulate objects. The success criteria for the three example tasks as seen in Sec. 4.2 are as follows. *Fold Cloth*: The distance between two of the corner particles is below a threshold. This allows for folding either front-to-back or side-to-side. *Lift Teddy*: The center of the teddy must be lifted above a certain threshold. *Stow Teddy*: The center of the teddy must be inside the box and below a certain threshold. The initial object positions are chosen randomly without rotation. The box position for the *Stow Teddy* task is fixed. All example policies are using a U-Net diffusion model with reduced down dimension sizes of $[256, 512, 1024]$, and a diffusion step embedding dimension size of $256$. Other parameters are at their default values. The models are trained for 100 to 120 epochs. Observations are robot EEF information, and a mix of depth and image data. The last two observations are stacked and used as input. The models predict 16 actions at a time. Between 60 and 90 demonstrations are used for training.

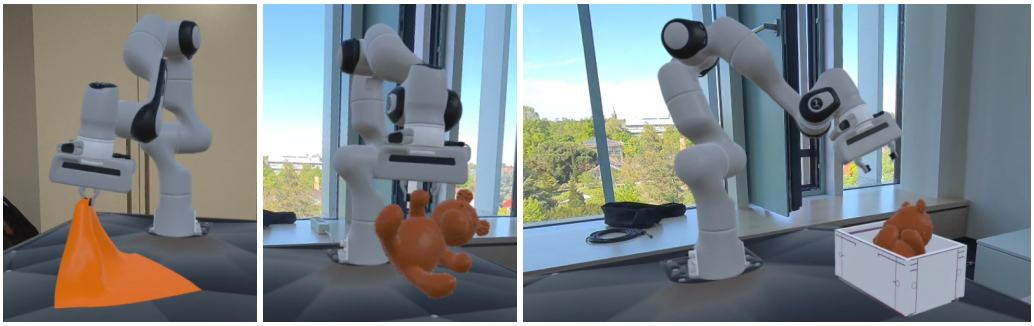


Figure 18: Views wearing a VR headset and collecting task demonstrations using IRIS with IsaacLab. Tasks from left to right are: folding a cloth in half, lifting a deformable teddy, and stowing a deformable teddy in a slightly undersized box.

### D.3 High Dynamic Data Collection

Traditional data collection methods struggle to acquire successful demonstrations in dynamic tasks that require high action timeliness and precision. To validate the practicality and effectiveness of IRIS, we applied it to a custom-designed table tennis task in MuJoCo simulation environment. IRIS was used to collect high-frequency strike data, and the collected demonstrations were evaluated using state-of-the-art imitation learning algorithms. In the task, a human demonstrator controls a virtual bat via a handheld controller to return randomly generated incoming balls, closely mimicking real bat manipulation. The system captures 73 around 5-second demonstrations at 100 Hz, where each demonstration movement consists of a forward stroke and natural backward release. The observation includes bat proprioceptive state and dual camera images, and the action is the desired bat position and orientation in task space. Fig. 8a shows the performance of imitation learning models

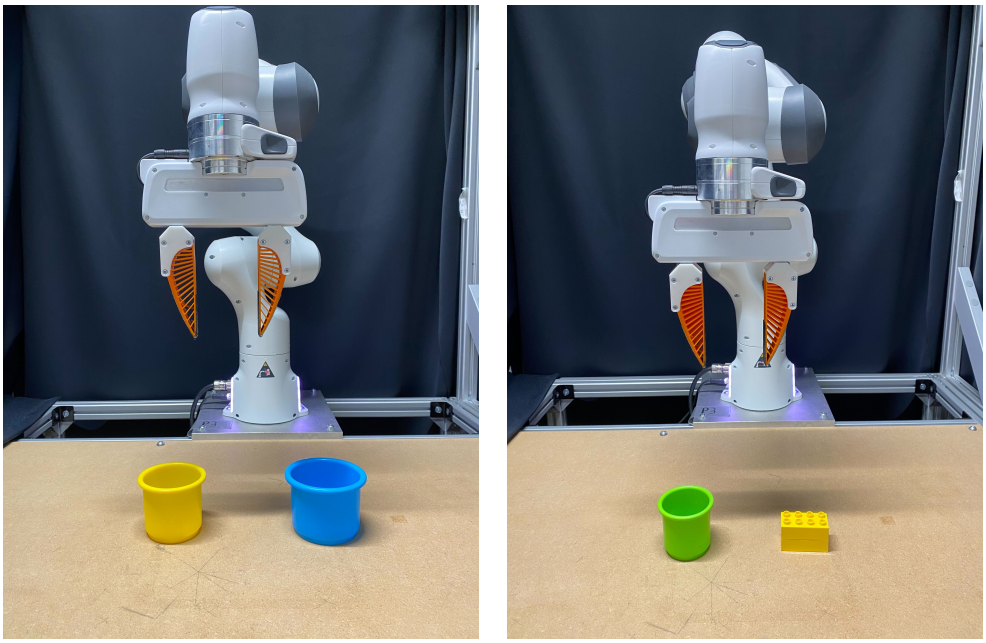


Figure 19: Two real-world manipulation tasks: Cup Inserting and Picking Up Lego.

trained on this data, using ball interception rate and successful return rate as evaluation metrics. The trained policy should first reach the incoming ball with natural stroke movement as human players and then hit the ball to opposite side table.

### D.4 Real Robot Experiment

To evaluate the real-world applicability of IRIS, we conducted a comparative study against a standard teleoperation baseline (Tele-Op) using two real-world manipulation tasks: Cup Inserting and Picking Up Lego (Fig. 19). For each task, 30 demonstrations were collected using both IRIS and Tele-Op, under identical experimental conditions. These datasets were then used to train two separate BC-Transformer policies, using consistent hyperparameters. We assessed both the data collection success rate (i.e., the percentage of successful demonstrations during collection) and the policy success rate (i.e., performance of trained policies in executing the tasks). As illustrated in Fig. 8b, IRIS not only achieved a higher success rate during demonstration collection, but also resulted in better-performing policies compared to those trained on Tele-Op data. These results indicate that IRIS facilitates higher-quality real-world data collection, which in turn leads to more effective imitation learning outcomes.

## E System Performance Analysis

The performance analysis focuses on two key aspects: network latency and headset FPS (frames per second). Since IRIS uses asynchronous bidirectional data transfer, network latency is low, averaging around 20-30 ms. Transmission bandwidth depends on Wi-Fi capacity. Even for large scenes from RoboCasa, which contain over 200 MB of compressed assets, it takes no more than 5 seconds to transfer and generate a full scene with more than 300 objects. For real robot teleoperation, the system handles point cloud data efficiently, achieving a transmission speed of 10,000 points at 60 Hz—exceeding the camera's frame rate.

The FPS performance depends on the hardware. On the Meta Quest 3, a scene with one robot runs at approximately 70 FPS, while on HoloLens 2, it runs around 40 FPS. As the scene size increases, FPS gradually decreases. With around 200 objects, the Meta Quest 3 headsets struggle to keep up with head movement, causing virtual objects to lag or become stuck. However, performance is

additionally influenced by the complexity of the meshes and textures, as these require significant computational resources from the XR headset.

In our experiments by using Meta Quest 3, IRIS successfully handled all benchmark scenarios listed in the paper, except for some scenes from RoboCasa [46]. These scenes have over 700 MB of assets in one single instance, which is closed to the Meta Quest 3 RAM limit. Nonetheless, IRIS was able to manage most of scenes from RoboCasa without any significant performance issues. For real robot data collection, the optimal point cloud size is around 10,000 points, which achieves a balance between point cloud quality and FPS, maintaining a frame rate of approximately 40 FPS.