

WebXRIT - A lightweight Interaction Toolkit to create Virtual Reality Training Applications for WebXR with Unity

TINO HENTSCHEL, University of Lübeck, Germany

JAN NEUHÖFER, Hamburg University of Applied Sciences, Germany

MARKUS DRESEL, University of Lübeck, Germany

OVE BRENT KLASSEN, University of Lübeck, Germany

PHILIPP GOLDBACH, University of Lübeck, Germany

NICOLE JOCHEMS, University of Lübeck, Germany

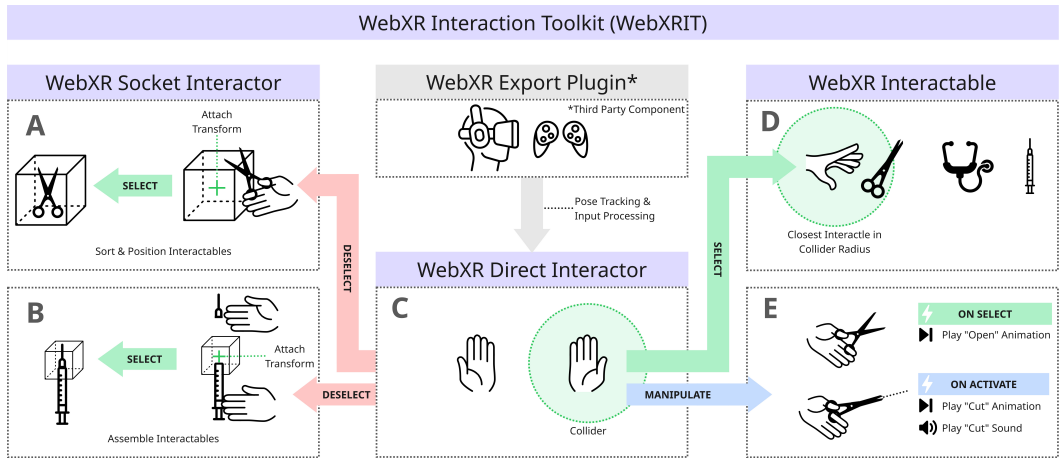


Fig. 1. System overview of the *WebXR Interaction Toolkit* (WebXRIT). (A–B) *WebXRSocketInteractors* automatically select and attach *WebXRInteractables* (e.g., scissors, syringe, needle) released by a *WebXRDirectInteractor*, enabling arrangement and assembling tasks in IVR training. (C–E) The *WebXRDirectInteractor* uses input and pose tracking from the *WebXR Export* plugin to control virtual hands to select *WebXRInteractables* within collider range and trigger *On Select* and *On Activate* events on interactables (e.g., play animations or sounds).

Immersive Virtual Reality (IVR) offers promising opportunities for education and training, yet its broader adoption in web-based learning environments remains limited due to performance constraints and the lack of support for industry-standard game engines. This paper presents the *WebXR Interaction Toolkit* (WebXRIT), a lightweight framework for creating WebXR-based training applications in Unity. Designed to provide

Authors' Contact Information: Tino Hentschel, Institute of Human-Centered Interactive Systems, University of Lübeck, Lübeck, Germany, t.hentschel@uni-luebeck.de; Jan Neuhöfer, Hamburg University of Applied Sciences, Hamburg, Germany, jan.neuhoefer@haw-hamburg.de; Markus Dresel, Institute of Human-Centered Interactive Systems, University of Lübeck, Lübeck, Germany, m.dresel@uni-luebeck.de; Ove Brent Klasen, University of Lübeck, Lübeck, Germany, ove.klasen@student.uni-luebeck.de; Philipp Goldbach, Institute of Human-Centered Interactive Systems, University of Lübeck, Lübeck, Germany, philipp.goldbach@gmail.com; Nicole Jochems, Institute of Human-Centered Interactive Systems, University of Lübeck, Lübeck, Germany, jochems@imis.uni-luebeck.de.



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

© 2026 Copyright held by the owner/author(s).

ACM 2573-0142/2026/6-ARTEICS015

<https://doi.org/10.1145/3816767>

essential interaction functionalities while minimizing runtime overhead, WebXRIT integrates with the open-source WebXR Export plugin and follows Unity's component-based interaction architecture. To evaluate its runtime performance, we compared WebXRIT with Unity's default *XR Interaction Toolkit* (XRIT) across twelve valid build configurations (varying editor version, render pipeline, and color space) using the *Meta Quest 3*. Results show that WebXRIT consistently outperforms XRIT, achieving significantly higher frame rates in multiple configurations, with mean values up to 78 FPS, exceeding the recommended threshold for native IVR applications. These findings highlight the potential of the WebXRIT to enable performant and accessible IVR training applications on the web. Future work will extend our toolkit with support for hand tracking, AR interaction, and additional navigation and UI components.

CCS Concepts: • **Human-centered computing** → **Virtual reality**; **Web-based interaction**; • **Applied computing** → Interactive learning environments.

Additional Key Words and Phrases: WebXR, Virtual Reality Training, Unity Engine

ACM Reference Format:

Tino Hentschel, Jan Neuhöfer, Markus Dresel, Ove Brent Klasen, Philipp Goldbach, and Nicole Jochems. 2026. WebXRIT - A lightweight Interaction Toolkit to create Virtual Reality Training Applications for WebXR with Unity. *Proc. ACM Hum.-Comput. Interact.* 10, 4, Article EICS015 (June 2026), 22 pages. <https://doi.org/10.1145/3816767>

1 Introduction

Immersive Virtual Reality (IVR) has gained increasing recognition as an effective tool for training and education, promoting various learning outcomes such as factual, conceptual, and procedural knowledge [1, 19, 36]. Systematic literature reviews suggest that IVR-based training can enhance procedural skills and knowledge transfer [23] as well as spatial reasoning and problem-solving abilities [10] across diverse professional contexts such as medicine, emergency response, industrial maintenance, aviation safety, construction, and engineering [23]. Particularly in fields like nursing education, which strongly benefit from simulation-based training [27, 40], multiple meta-analyses [33] have demonstrated that IVR can improve knowledge acquisition, skill performance, and retention, while also fostering learner satisfaction, motivation, and self-efficacy [21].

Despite these advantages, several challenges continue to hinder the broad integration of IVR into educational settings. These include the lack of interoperability between platforms and the absence of open standards [47], limited access to technology and insufficient technical expertise among users and educators [34], as well as the limited availability of personalized training scenarios and suitable authoring tools for the creation and adaptation of IVR learning content [25]. One potential solution to addressing challenges related to interoperability, standardization, and accessibility lies in the use of *WebXR*¹ on consumer-grade mobile Head-Mounted Displays (HMDs) such as the *Meta Quest 3* [32]. WebXR is an open standard that enables the hosting and execution of virtual (VR) and augmented reality (AR) experiences directly on the web, allowing users to access immersive applications through a browser across various devices and platforms without requiring software installation.

However, WebXR has not yet achieved widespread adoption in the development of educational IVR applications. One major limitation concerns runtime performance: WebXR applications often fail to match the frame rates (FPS) of native IVR applications [9, 32, 53]. Such performance deficits can diminish perceptual immersion and increase the risk of motion sickness and dizziness, representing a significant usability concern in VR contexts [49]. From a developer perspective, while WebXR is already supported by web-first frameworks such as *Three.js*, *A-Frame*, or *Babylon.js* [7], support for industry-standard game engines like Unity² remains limited. These engines, however,

¹ <https://immersiveweb.dev/>, accessed on October 20, 2025.

² <https://unity.com/products/unity-engine>, accessed on October 20, 2025.

rank among the most commonly used tools for creating virtual reality training applications [1, 50]. Among the few available solutions for Unity-based WebXR development is the open-source *WebXR Export*³ plugin, which can be used in conjunction with Unity's default cross-reality interaction system, the *XR Interaction Toolkit* (XRIT)⁴. However, XRIT does not provide official WebXR support, nor is it optimized for web deployment, which may negatively impact the runtime performance of WebXR applications.

To address these gaps, we propose a solution for developing WebXR-based training applications in Unity by integrating the WebXR Export plugin with a novel lightweight *WebXR Interaction Toolkit* (WebXRIT). This Paper describes the core components of our solution in detail and the subsequent empirical evaluation compares the WebXRIT to Unity's XRIT across different configuration contexts regarding runtime performance on the mobile HMD Meta Quest 3. Our experimental results demonstrate that WebXR applications built with WebXRIT consistently achieve significantly higher frame rates than those developed with XRIT. In three of the tested configurations, WebXRIT-based applications reach mean frame rates of up to approximately 78 FPS, thereby exceeding the recommended threshold of 72 FPS for native IVR applications on the Quest 3, while no configuration with the XRIT exceeds this threshold.

In summary, the core research contribution of this work is the design of a simple yet robust lightweight IVR interaction toolkit for Unity that is explicitly tailored for WebXR builds and demonstrably achieves significantly higher runtime performance than applications using Unity's default XR Interaction Toolkit. By providing this toolkit, we aim to lower barriers to WebXR development and enable developers within the established Unity ecosystem to explore WebXR without transitioning to entirely different toolchains. While this work focuses on educational and training scenarios as a representative application domain, the proposed *WebXR Interaction Toolkit* (WebXRIT) is not restricted to this context and can be readily applied to other WebXR-based IVR application domains.

2 Related Work

2.1 WebXR in IVR Training and Education Applications

The WebXR Device API (WebXR) is an open standard recommended by the World Wide Web Consortium (W3C). It provides an interface that allows web applications to access and interact with extended reality (XR) hardware, such as HMDs, XR glasses, and motion-sensing devices like smartphones and tablets, directly within a web environment. WebXR manages device pose tracking (position and orientation) as well as input events from controllers or hand tracking devices. Although WebXR is not itself a rendering technology, it integrates seamlessly with browser graphics APIs such as WebGL. These APIs handle 3D asset management and rendering and can be accessed through higher-level JavaScript frameworks such as *Three.js* [9]. WebXR is device- and platform-independent, enabling the hosting of AR and VR experiences on the web and making them accessible across a wide range of devices directly through the browser [35]. By eliminating the need to install or update dedicated standalone XR applications on specific hardware, WebXR-based applications become instantly available through an internet connection [13]. WebXR thus enhances portability and accessibility and simplifies cross-platform development of XR content [32]. Its technological characteristics also include scalability and support for remote collaboration in IVR scenarios [5]. In the context of IVR education and training, WebXR enables accessible learning experiences for students by providing immersive learning environments on mobile VR devices

³ <https://github.com/De-Panther/unity-webxr-export>, accessed on October 20, 2025.

⁴ <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.2/manual/index.html>, accessed on October 20, 2025.

anytime and anywhere. This flexibility has been shown to improve learning outcomes in domains such as nursing training compared to scheduled sessions on fixed PC-VR setups [21].

Several recent research projects have demonstrated the potential of WebXR for educational use. The *WebXR Health Learning Module Application* [18, 20], for instance, allows nursing students to train the intramuscular injection procedure using a single WebXR application that runs across three devices with different XR capabilities: the Microsoft HoloLens (AR), Oculus Quest 2 (VR), and Android smartphones (AR). The project *MolecularWebXR* [13] further extends this potential by offering a multi-user educational platform for chemistry and biology that supports both VR and AR modes, as well as controller input and hand tracking. Other examples include WebXR-based applications for assembly training [41], construction worker safety training [3], and primary school learning [17].

The educational domain was selected as the primary application area for this work because it features a wide variation in spatial contexts, hardware availability, and technical expertise among both developers and end users, making it a representative example for demonstrating the applicability of the WebXRIT. At the same time, the toolkit is not limited to educational settings and can be readily applied to other application domains.

Despite its promise, WebXR has not yet achieved widespread adoption in educational software development. Two major limiting factors are (1) runtime performance and (2) support for industry-standard game engines.

(1) *Runtime Performance.* One of the main challenges faced by WebXR creators is the performance disadvantage of WebXR applications compared to native XR applications [53]. This limitation is largely due to the constrained processing power and memory available in web browsers [32]. Although browser technologies have advanced considerably in recent years, WebXR applications still often fail to meet the high frame rate requirements of immersive XR experiences (e.g., 72 frames per second on the Meta Quest platform) when compared to their native counterparts [9, 32]. Lower frame rates can negatively affect usability and increase the risk of motion sickness [49]. Conversely, higher frame rates have been shown to significantly enhance the sense of presence experienced by users [4, 38], which in turn can positively influence learning outcomes in virtual reality [28, 30].

(2) *Limited Support for Industry-Standard Game Engines.* While WebXR is already supported by web-first development frameworks such as *Three.js*, *A-Frame*, or *Babylon.js* [7], there is currently limited support for industry-standard game engines like Unity. These engines, however, represent some of the most widely used tools for developing virtual reality applications [50], and Unity in particular is frequently employed for creating IVR education and training applications [1]. The limited compatibility between Unity and WebXR can thus pose an additional barrier to the broader adoption of WebXR in educational contexts.

2.2 Unity and WebXR

Unity is a real-time 3D engine whose editor enables the construction of interactive *Scenes* composed of *GameObjects* that gain functionality through the attachment of programmable *Components*⁵. Unity supports deployment across a wide range of target platforms, including desktop, mobile, and virtual or augmented reality devices. Its extensive plugin ecosystem further provides numerous extensions that enable developers to build complex interactive 3D applications by configuring and combining reusable components [9].

Evaluations of Unity's learnability report mixed findings, ranging from comparatively easy learnability [9, 20] to reports of a moderate learning curve [16]. While no major usability issues

⁵ <https://docs.unity3d.com/Manual/index.html>, accessed on October 20, 2025.

are generally reported, novice developers may encounter difficulties navigating the editor's user interface or interpreting error messages, particularly during early stages of development [39]. Despite these challenges, Unity offers several advantages for novice and intermediate IVR developers. These include strong support for rapid prototyping [22] and access to a large and active online community that can provide peer support [22, 44]. Recent advances in Large Language Model (LLM)-based code synthesis and prompt-driven AR/VR application generation, such as the work by de la Torre et al. [14], offer additional support for novice developers by enabling the generation of Unity scenes and source code with little to no programming through textual input. Although this approach currently faces limitations, including the inherent unpredictability of generative LLMs and the need for manual implementation of IVR interactions such as object grabbing, it already demonstrates the potential of LLMs to assist non-expert users in creating VR experiences within the Unity ecosystem. In addition, Unity is one of the most widely used development tools for IVR applications [50] and is commonly employed in IVR research [31] as well as in the development of IVR training applications [1]. Based on these advantages, Unity was selected as the engine for the implementation of the WebXR Interaction Toolkit.

At the time of this publication, Unity does not offer official support for WebXR. Nevertheless, it is possible to enable WebXR functionality in Unity by using third-party solutions, most prominently the open-source and community-maintained *WebXR Export* plugin. This plugin adds WebXR as a build target to Unity and provides an additional *WebXR Interactions* package that allows developers to access the position and orientation of the HMD and its controllers and to process user input from controllers or via hand tracking.

Although research on the combination of Unity and WebXR is still limited, existing work demonstrates the viability of this approach. Lautenbach et al. [29] presented the *ProjectVerse* framework, which uses the WebXR Export plugin to port existing Unity-based VR applications from various research projects to WebXR. The framework integrates these applications into a browser-based VR platform designed to make educational experiences more accessible to students and instructors. Guo and Mogra [17] employed Unity and WebXR Export to develop an interactive mobile VR application for mining education. Their approach combined 360-degree video footage with 3D models of mining environments and equipment, allowing users to explore the mining process from multiple perspectives.

For VR training scenarios that require more complex interaction systems, developers can extend the WebXR Export plugin with custom code or integrate Unity's *XR Interaction Toolkit* (XRIT). XRIT is a high-level interaction framework designed for creating cross-platform VR and AR applications, leveraging Unity's component-based architecture. Recent versions of the WebXR Export plugin support partial integration of XRIT.

However, since Unity itself does not natively support WebXR, XRIT cannot be assumed to be optimized for WebXR applications. This assumption is supported by a recent study by Cattaneo et al. [11], who developed a WebXR-based telemedicine training application in Unity. The authors reported frequent input delays when using VR controllers and suggested that interaction handling should be refined. Another limiting factor in using XRIT with WebXR stems from the restricted XR capabilities of web browsers. To enable cross-platform compatibility and for security reasons, browsers do not consistently expose all XR hardware features across devices [9]. As of October 2025, approximately 80% of web browsers support the WebXR Device API, however only partially, since several features remain under development or lack finalized specifications.⁶ Consequently, not all high-level XRIT functionalities can currently be utilized with WebXR. Given these constraints, XRIT may impose unnecessary overhead for WebXR development.

⁶<https://caniuse.com/webxr>, accessed October 20, 2025.

To implement comprehensive yet performant interactive IVR training scenarios, we therefore propose a lightweight approach that provides essential IVR interaction functionality through a custom *WebXR Interaction Toolkit* (WebXRIT). The following chapter provides a system overview of our solution, including detailed descriptions of all components.

3 WebXR Interaction Toolkit

The *WebXR Interaction Toolkit* (WebXRIT) provides a set of Unity components for implementing interaction functionality in IVR training applications. In its architecture and component structure, WebXRIT follows the conventions of Unity's *XR Interaction Toolkit* (XRIT). This design choice aims to facilitate adoption among Unity developers by providing familiar code structures and interaction patterns, thereby also simplifying the porting of existing Unity-based XR projects to WebXR.

Our solution differs from XRIT in two key aspects: (1) a lightweight architectural approach, and (2) a simplified decentralized interaction management system.

3.1 Lightweight Architectural Approach

WebXRIT adopts the XRIT concept of dividing interaction functionality into two component types: *Interactors* and *Interactables*⁷. Interactables are objects in the virtual environment (VE) that can be selected (e.g., tools that are grabbed by the user) and manipulated (e.g., changing state upon activation, such as scissors opening and closing). Interactors are responsible for performing these actions on the interactables. The most common type of interactor is the *DirectInteractor*, which is attached to the user's virtual hands and enables them to pick up and manipulate virtual objects that have corresponding interactable components attached, such as task-related tools or equipment in a typical IVR training scenario. Other types of interactors in the XRIT enable interactions via gaze-based or raycast-based mechanisms, or even without direct user input, such as *SocketInteractors*, which can automatically select interactables as soon as they enter a designated area of the VE (e.g., a keyhole that automatically holds a key when it is placed inside it).

We adopt this division of interaction components in WebXRIT because it represents a core design principle of Unity's default XR Interaction Toolkit (XRIT). Novice and intermediate Unity XR developers are therefore likely to be familiar with this interaction model, and existing Unity-based standalone XR projects implemented with XRIT can be ported to WebXR by replacing XRIT components with their corresponding WebXRIT counterparts (see Section 3.3).

Compared to XRIT, however, the interaction components of WebXRIT provide a deliberately reduced functional scope that primarily implements the interaction types of selection and manipulation of virtual objects. These interaction types were chosen because they have been identified by several authors as the most fundamental forms of interaction in IVR applications [6, 15, 51] and are also among the most frequently used interaction design elements in educational VR applications [42]. High-level XRIT features, such as interaction filtering, are intentionally omitted in WebXRIT. Such features introduce additional checks and delegation mechanisms for each interaction between interactors and interactables, which can negatively affect the runtime performance of WebXR applications.

In its current version, WebXRIT exclusively relies on event-based collision detection between interactors and interactables using Unity's physics system (see Section 3.2) as the underlying selection and manipulation technique. Continuous per-frame interaction monitoring or additional interaction mechanisms based on physics raycasts, as provided by XRIT, are not supported at this stage. Currently WebXRIT offers a more limited feature set than Unity's XRIT. However, due to this

⁷ <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.2/manual/interaction-components.html>, accessed on October 20, 2025.

lightweight architectural approach, WebXRIT achieves reduced system complexity and improved runtime performance when using basic interaction types (selection and manipulation) in WebXR builds, as demonstrated in Section 4.

The XRIT provides components for processing input through both handheld controllers and optical hand tracking. While hand tracking may be more suitable for certain tasks within specific domains [2, 8, 45], our toolkit is designed to be broadly applicable across multiple domains and to ensure backward compatibility with IVR systems lacking hand tracking. Since controllers in IVR training applications have also been shown to achieve equal or superior task performance in terms of accuracy, speed, and error rate [24, 26, 37], the current iteration of WebXRIT uses controllers as the default input modality. Input processing and the tracking of controller position and orientation are handled via the *WebXR Export* plugin. This lightweight design supports the interoperability of WebXR training applications built with WebXRIT, allowing them to run not only on current state of the art HMDs but also on previous generations of stationary or mobile HMDs, such as Oculus Quest or HTC Vive.

3.2 Interaction Management

In the XRIT, a dedicated *manager component*⁸ must be present in every Unity scene to serve as an intermediary between all interaction components. At runtime, in each frame, this manager queries all interactors and interactables in the scene and updates their interaction states (e.g., which interactor is currently selecting or manipulating which interactable) through an event system. While this centralized architecture ensures consistency across a wide range of specialized interaction components (e.g., components for gaze-based interaction), it can also introduce significant performance overhead in training scenarios that require only basic interactions.

To optimize runtime performance, WebXRIT employs a simplified and decentralized interaction management approach. In our system, interactors directly manage nearby interactables through Unity's physics components and parent-child object relationships within the scene hierarchy.

Interaction management in the WebXRIT is proximity-based and relies on Unity's physics *Colliders*⁹, which are attached to all interaction components. During runtime, when the collider of an interactor begins to overlap with the collider of an interactable (e.g., when a virtual hand reaches toward an item), Unity's physics event function *OnTriggerEnter* is invoked on the interactor. This event causes the interactor to add the corresponding interactable to an internal list of *contactInteractables* in the frame in which the overlap occurs. This process applies to all interactables within the collider range of the interactor. Conversely, when an interactable ceases to overlap with the interactor's collider, the *OnTriggerExit* event is triggered and the interactable is removed from the *contactInteractables* list. Through this mechanism, each interactor continuously maintains an up-to-date set of all interactables in its immediate proximity. When a selection action is performed (e.g., when the user presses the grip button on the controller to grab an item), the interactor selects the closest interactable from the *contactInteractables* list and attaches it by making it a child object of the interactor within the scene hierarchy (see Section 3.3.3). Once selected, an interactable can be manipulated or released by the interactor (see Sections 3.3.1 and 3.3.3). Figure 2 illustrates the sequence of proximity detection, selection, and manipulation events between a *DirectInteractor* and an *Interactable* in the WebXRIT using a UML sequence diagram.

This interaction management approach renders interactions purely event-driven, as interaction logic is executed only in response to collision or input events and only in the frames in which

⁸ <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.3/manual/xr-interaction-manager.html>, accessed on October 15, 2025

⁹ <https://docs.unity3d.com/ScriptReference/Collider.html>, accessed on February 08, 2026.

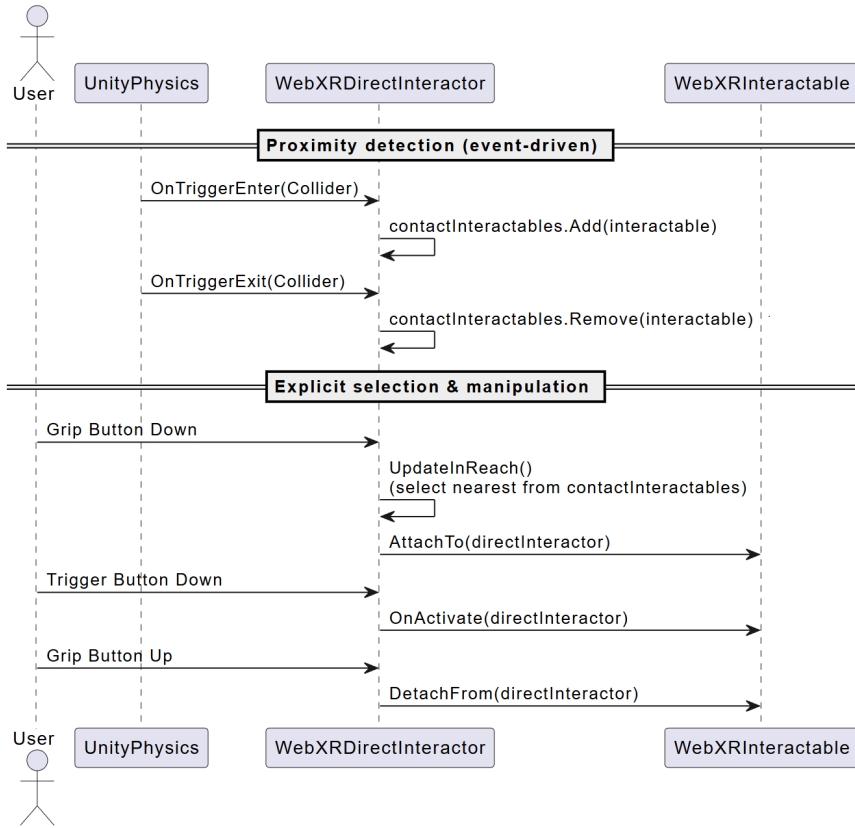


Fig. 2. Simplified sequence diagram of the primary interaction loop of the WebXRIT, illustrated using the interaction between a *DirectInteractor* and an *Interactable*. Potential interaction candidates (interactables) are managed in a proximity-based, event-driven manner by the *DirectInteractor* using Unity physics callbacks (*OnTriggerEnter* / *OnTriggerExit*), which update a local set of nearby *Interactables*. Selection and manipulation are resolved through explicit user input: when the grip button is pressed, the *DirectInteractor* selects the nearest candidate and attaches it to itself (i.e., the user's hand); trigger input invokes activation events, and releasing the grip detaches the *Interactable*.

these events occur. Consequently, WebXRIT avoids continuous per-frame updates of interaction states for all components, which are employed in the XRIT. Rather than relying on a centralized interaction manager that processes all interaction components in the scene, WebXRIT adopts a decentralized design in which each interactor manages interactions exclusively with interactables in its immediate proximity. This design is not intended to replicate the full interaction feature set of XRIT. Instead, it focuses on implementing the fundamental interaction types of selection and manipulation by proximity with reduced computational overhead, thereby improving runtime performance in WebXR builds. The decentralized structure also facilitates extensibility, allowing developers to easily integrate custom components with additional functionality into the toolkit.

3.3 Interaction Components

WebXRIT currently includes two types of interactors, a *DirectInteractor* (*WebXRDirectInteractor*) and a *SocketInteractor* (*WebXRSocketInteractor*), as well as a general-purpose *Interactable* (*WebXRInteractable*). These components are modeled after their XRIT counterparts, *XRDirectInteractor*¹⁰, *XRSocketInteractor*¹¹, and *XRGrabInteractable*¹², and represent lightweight WebXR equivalents of these Unity components.

Figure 1 illustrates the system overview of WebXRIT, showing all components and their relationships within the Unity environment and the following sections describe each component in detail.

3.3.1 *WebXRDirectInteractor*. The *WebXRDirectInteractor* is an interactor component that is attached to the user's virtual hands (e.g., 3D hand models), with one instance per hand. Each instance holds a reference to its corresponding controller. For example, the left-hand *WebXRDirectInteractor* references the left controller, and processes input received from it.

A *WebXRDirectInteractor* enables users to perform direct selection of interactable components in the virtual environment by "grabbing" them with their virtual hands, typically by pressing the *grip button* on the controller. Before an interactable is selected, the *WebXRDirectInteractor* enters a hover state in which it uses a Unity *Collider* component to detect interactables within range, i.e., those overlapping with its collider boundaries (Figure 1d). By default, the closest interactable within range is targeted for selection. Interactables currently held by the user can also be manipulated or "activated" by pressing the *trigger button* on the controller. When activated, a dedicated *On Activation* event may be executed if one is defined for the interactable. For example, a pair of medical scissors held by the user could trigger a cutting animation when activated (Figure 1e).

3.3.2 *WebXRSocketInteractor*. The *WebXRSocketInteractor* is used to hold interactables via a socket and serves as a virtual storage or placement area for objects. Unlike the *WebXRDirectInteractor*, it is not connected to any controller and can be attached to any *GameObject* equipped with a collider component.

An active *WebXRSocketInteractor* automatically selects and attaches the closest interactable located within the boundaries of its collider. The position and orientation of an attached interactable can be defined by an optional *Transform*¹³ component, otherwise the *Transform* of the *WebXRSocketInteractor* itself is used. When an interactable held by a *WebXRDirectInteractor* (i.e., the user's virtual hand) enters the socket's collider, a preview of the interactable can be displayed at the socket's attach position (Figure 3a). If the user releases the interactable by releasing the grip button, it automatically snaps into place and becomes attached to the socket (Figure 3b). A *WebXRSocketInteractor* can hold only one interactable at a time. Interactables can be removed from sockets simply by grabbing them again with a *WebXRDirectInteractor*.

Sockets are particularly versatile in IVR training scenarios, as they can be used to design a variety of training tasks, such as sorting, arranging, or assembling objects, for example, combining medical components to assemble a syringe before an injection (Figure 3c-d). Additionally, if a scenario provides a virtual body for the user a *WebXRSocketInteractor* can also be attached to it to simulate tasks like putting on protective work clothing.

¹⁰ <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.3/manual/xr-direct-interactor.html>, accessed on October 15, 2025.

¹¹ <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.3/manual/xr-socket-interactor.html>, accessed on October 15, 2025.

¹² <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.3/manual/xr-grab-interactable.html>, accessed on October 15, 2025.

¹³ <https://docs.unity3d.com/ScriptReference/Transform.html>, accessed on October 20, 2025

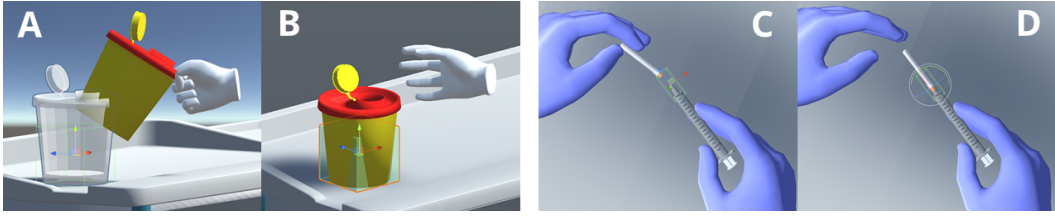


Fig. 3. Use cases of *WebXRSocketInteractors*. (a–b) Arrangement task: placing a medical item on a dressing trolley. (c–d) Assembling task: combining a syringe and a needle. In both cases, the *WebXRSocketInteractor* automatically selects and attaches *Interactables* released by a *WebXRDirectInteractor*.

3.3.3 *WebXRInteractable*. A *WebXRInteractable* is a component that can be selected and manipulated by any interactor (e.g., grabbed by a *WebXRDirectInteractor*). During selection, the interactable becomes a child of the selecting interactor within the scene hierarchy (Figure 4), meaning it follows the interactor’s movement by matching its local position and rotation, for instance, following the user’s hand movements while being held. Each *WebXRInteractable* requires both a collider and a *Rigidbody*¹⁴ (Unity’s 3D physics component) to detect interactions through Unity’s physics engine. However, once an interactable is selected, its physics simulation is disabled by default to avoid unnecessary processing and unintended collisions, and it becomes a kinematic object. Upon release, the parent–child relationship between the interactor and the interactable is removed, and the physics simulation is re-enabled.

The *WebXRInteractable* supports custom responses to selection and manipulation events through its *On Select* and *On Activate* properties. These properties are implemented as *Unity Events*¹⁵ that can hold multiple serialized function calls to other *GameObjects* or components within the scene. Typical uses include triggering Unity API functions such as playing animations or audio clips, which can be configured directly in the Unity Editor without additional scripting. For example, when the user holds a pair of medical scissors during a nursing training scenario, activating the interactable could play both a cutting animation and a corresponding sound effect. Furthermore, the *On Select* and *On Activate* properties can call functions on custom components attached to the interactable, thus providing interfaces for easily extending interaction behaviors through user-defined scripts if required. As an example for the integration of a custom component into the WebXRIT, this paper introduces a *Hand Pose Container*, an optional component that can be attached to interactables to define the specific hand gesture the user’s virtual hand should adopt when “grabbing” (i.e., selecting) that interactable (Figure 4).

4 Performance Test Design

To evaluate the runtime performance of the proposed WebXR Interaction Toolkit (WebXRIT) in comparison to Unity’s XR Interaction Toolkit (XRIT), we used the same Unity training scene to create WebXR applications (builds) based on both frameworks, that are functionally equivalent. The following sections describe the evaluation metric, the virtual environment, relevant configuration factors, and the measurement procedure.

4.1 Performance Metric

Performance was assessed using the *frame rate*, expressed in frames per second (FPS). The frame rate denotes the number of images (frames) that an application can generate and pass to a display

¹⁴ <https://docs.unity3d.com/ScriptReference/Rigidbody.html>, accessed on October 20, 2025.

¹⁵ <https://docs.unity3d.com/6000.2/Documentation/ScriptReference/Events.UnityEvent.html>, accessed on October 20, 2025.

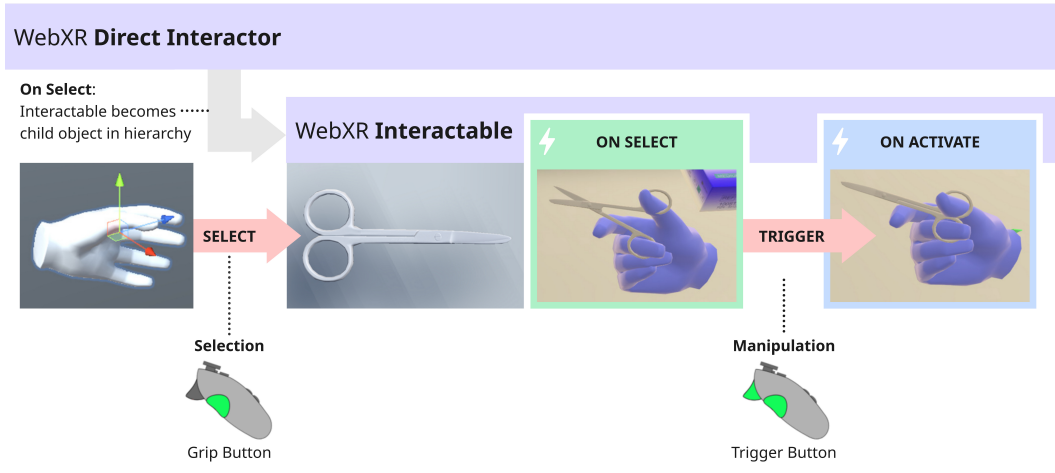


Fig. 4. Relationship between *WebXRDirectInteractors* and *WebXRInteractables*. When a user selects an interactable by pressing the grip button on the controller, it becomes a child of the *DirectInteractor* in the scene hierarchy, raising an *On Select* event. Likewise, when the user presses the trigger button, the interactable's *On Activate* event is raised. Both events can be used to control component behaviours such as hand poses and object animations (e.g., performing a “cut” animation with scissors).

in one second. It is a critical determinant of user experience, particularly in virtual reality, where higher frame rates are strongly preferred to reduce visual discomfort and enhance the perception of smooth motion [49].

To minimize perceptual artifacts such as motion blur or judder, the frame rate should ideally match the fixed refresh rate of the display in hertz (Hz) [52]. On the Meta Quest 3, the default refresh rate is 72 Hz for native applications¹⁶ and 90 Hz for WebXR applications executed in the integrated browser¹⁷. Based on these specifications, we defined 90 FPS or higher as the *optimal target frame rate* for WebXR applications, while 72 FPS was selected as the *minimum acceptance threshold*, representing the default performance of native applications.

4.2 Virtual Environment

The Unity scene used in all performance tests implements a training scenario for the nursing procedure “Aseptic Change of Wound Dressing”. Figure 5 shows the virtual environment. It represents a patient room with a patient lying in bed and a dressing trolley containing various medical materials (tweezers, swabs, compresses). The scene is designed as a fully functional training environment, enabling nursing students to practice procedural steps with optional text-based guidance via a User Interface (UI) in world space.

From a technical perspective, the scene consist of **724 GameObjects** in total, of which 183 are 3D objects with mesh geometry. It employs 67 unique materials and references 47 unique textures. In terms of geometric complexity, the combined mesh content amounts to approximately 58,000 triangles. The largest meshes include the patient model ($\approx 10,000$ triangles) and the dressing trolley ($\approx 7,000$ triangles). To support rendering optimizations, static objects were flagged accordingly, while dynamic ones remained unflagged. No additional manual performance optimizations were applied to the meshes, textures, or materials.

¹⁶ <https://developers.meta.com/horizon/documentation/unity/unity-set-disp-freq>, accessed on October 20, 2025.

¹⁷ <https://developers.meta.com/horizon/documentation/web/browser-specs/>, accessed on October 20, 2025.



Fig. 5. Virtual environment of the Unity test scene showing an overview of all main GameObjects (left) and a close-up of the medical materials on the dressing trolley (right).

4.3 Configuration Factors

Runtime performance in Unity-based WebXR applications is not only influenced by the interaction toolkit but also by several engine- and rendering-related configuration factors. These include the Unity editor version, the render pipeline and the chosen color space.

Unity Editor Versions. Our test design included two Unity editor versions: the Long-Term Support release Unity 2021 (U21, version 2021.3.45f1, released 2024), which is the version currently used for WebXRIT development, and the more recent Unity 6 (U6, version 6000.1.8f1, released 2025). The latter introduces improved support for web runtimes and render pipeline optimizations that may impact WebXR performance¹⁸.

Render Pipelines. Unity supports three major render pipelines¹⁹: the legacy Built-in Render Pipeline (BiRP), the Universal Render Pipeline (URP), and the High Definition Render Pipeline (HDRP). Since HDRP is tailored for photorealistic physically-based rendering and unsuitable for mobile VR due to its high computational demands, it was excluded. Our evaluation therefore focused on BiRP and URP:

- **Built-in Render Pipeline (BiRP):** Unity's legacy default pipeline, providing a general-purpose solution that remains widely used in VR projects.
- **Universal Render Pipeline (URP):** A lightweight, scriptable render pipeline designed for scalability across a wide range of platforms, from high-end PCs to mobile VR devices.

To ensure comparability, all builds employed the default render quality settings of the chosen pipeline, adjusted in line with Unity's²⁰ and Meta's²¹ recommendations for mobile VR. These included:

- Graphics API: WebGL 2
- High Dynamic Range (HDR): disabled
- Antialiasing: MSAA 4×
- Static Batching: enabled
- Dynamic Batching: enabled

¹⁸ <https://unity.com/blog/unity-6-features-announcement>, accessed on October 20, 2025.

¹⁹ <https://docs.unity3d.com/6000.2/Documentation/Manual/render-pipelines.html>, accessed on October 20, 2025.

²⁰ <https://learn.unity.com/tutorial/optimizing-your-vr-ar-experiences>, accessed on October 20, 2025.

²¹ <https://developers.meta.com/horizon/blog/tech-note-unity-settings-for-mobile-vr/>, accessed on October 20, 2025.

All builds used baked lighting with an identical lightmap and excluded post-processing effects. In addition, all builds employed the same version of the WebXR Export plugin (v0.22.1)²². In the Unity build settings, we enabled ASTC texture compression, set code optimization to “Speed,” and selected “Faster runtime” for code generation.

Color Spaces. Both render pipelines support Gamma and Linear color spaces for color representation during rendering. While Gamma is the historical default, Linear rendering yields more physically plausible results. Given potential runtime differences, both modes were included in the evaluation.

Factorial Design. To systematically assess these factors, we generated builds for all possible permutations:

- Interaction Toolkit: WebXRIT vs. XRIT
- Editor Version: U21 vs. U6
- Render Pipeline: URP vs. BiRP
- Color Space: Gamma vs. Linear

This resulted in **16 unique build configurations** (e.g., *WebXRIT U21 URP Gamma*).

4.4 Method and Procedure

All builds were hosted on a static web server within a local network and executed independently in the browser of a Meta Quest 3 (software version 77.1026, browser version 39.0).

For each build, performance was measured over a **60-second interval**. A custom C# script sampled the current frame rate every 0.1 seconds (10 Hz). This frequency was chosen to minimize both measurement overhead and temporal autocorrelation.

To ensure consistent rendering workloads, all tests followed a scripted animation sequence consisting of two phases:

- **Phase 1 (20 s):** Simulated user movement by animating the VR rig and virtual camera. The animation replicated the initial steps of training, including entering the patient’s room, passing along the bed, and focusing sequentially on the patient and the dressing trolley.
- **Phase 2 (40 s):** No simulated movement; the camera remained fixed on a predefined view encompassing the dressing trolley with medical materials, the UI, and part of the patient’s bed. This view corresponds to a typical working perspective in the training scenario, as it requires simultaneous rendering of several complex objects with distinct materials and textures.

In addition, the **UI** was kept active during both phases and a guidance text placeholder was displayed continuously to account for the performance impact of Unity UI components (e.g., text fields).

5 Results & Analysis

In total, only 12 of the initial 16 unique build configurations were available for evaluation. Four configurations that combined Unity 6 with the Universal Render Pipeline (URP) were excluded during testing, as it appears that they are incompatible with the WebXR Export plugin and exhibited errors in the display subsystem: Only one side of the stereoscopic display of the HMD was rendered correctly, while the other displayed a blank field. Consequently, no valid performance data could be obtained for these configurations. All results reported below are therefore based on the remaining **12 valid build configurations**.

²²<https://openupm.com/packages/com.de-panther.webxr/>, accessed on October 20, 2025

Table 1. Descriptive statistics of frame rate (FPS) and frame time (FT, in ms) across the 12 valid build configurations. Reported are mean, median, and standard deviation (SD), as well as the 90th percentile FPS.

Configuration	FPS				FT [ms]		
	Mean	Median	SD	90th Perc	Mean	Median	SD
WebXRIT U21 BiRP Gamma	77.69	77.97	9.93	64.28	14.25	12.83	29.51
WebXRIT U21 BiRP Linear	73.69	73.53	10.81	62.84	15.42	13.60	37.65
WebXRIT U6 BiRP Gamma	72.78	72.73	12.02	59.32	15.36	13.75	29.59
WebXRIT U6 BiRP Linear	71.75	72.07	9.35	63.06	15.56	13.88	33.52
WebXRIT U21 URP Gamma	66.83	69.86	11.80	45.13	17.52	14.31	49.84
XRIT U21 BiRP Linear	69.70	69.72	8.58	59.64	16.24	14.34	41.94
XRIT U21 BiRP Gamma	69.20	68.76	10.27	59.06	16.36	14.54	38.71
XRIT U6 BiRP Gamma	66.92	67.31	8.05	59.32	16.78	14.86	40.23
XRIT U21 URP Gamma	62.24	64.70	10.24	45.00	18.41	15.46	35.50
XRIT U6 BiRP Linear	64.15	64.22	9.03	54.05	17.62	15.57	42.82
WebXRIT U21 URP Linear	49.70	45.70	6.91	44.84	22.53	21.88	51.54
XRIT U21 URP Linear	46.29	45.00	4.89	44.76	23.91	22.22	52.16

5.1 Descriptive Statistics

For each configuration, we computed descriptive statistics on the raw frame rate (FPS) time series. The metrics included mean, median, and standard deviation (SD) of FPS, as well as the 90th percentile FPS²³. This measure is frequently used in benchmarking tools for real-time 3D applications such as *FrameView*²⁴, as it provides a stricter characterization of the performance experienced by users in a real use case scenario. In addition, we derived the *frame time* (FT), defined as the rendering duration per frame in milliseconds (ms), calculated for each raw data point as $FT = 1000/\text{FPS}$. FT offers a complementary perspective on the pacing of the WebXR application, as it directly expresses the time interval between two consecutive frames. If FT is unstable, users may perceive stutter even when the average frame rate is high. We therefore report mean, median, and standard deviation of FT alongside the FPS-based measures.

Across all configurations, mean FPS ranged from approximately ~46 (XRIT U21 URP Linear) to ~78 (WebXRIT U21 BiRP Gamma). WebXRIT builds consistently achieved higher average FPS and lower frame times than XRIT builds under otherwise identical conditions. Table 1 summarizes the descriptive statistics for all 12 valid configurations, ordered by median FPS values. The same ordering is applied in Figure 6, which illustrates the FPS distributions as violin plots. Medians (yellow line), means (black diamond), and 90th-percentile FPS (red triangle) are marked for each distribution.

5.2 Inferential Statistics

From the performance measurements of our 12 valid build configurations, we pre-registered contrasts isolating the framework effect (WebXRIT vs. XRIT) on FPS, while holding the other configuration factors (Unity version, render pipeline, and color space) constant, thus resulting in six configuration pairs (e.g., WebXRIT U6 BiRP Gamma vs. XRIT U6 BiRP Gamma) for statistical comparison.

²³In this context, the 90th percentile indicates that 90% of all rendered frames achieved at least this frame rate.

²⁴<https://www.nvidia.com/en-us/geforce/news/nvidia-frameview-power-and-performance-benchmarking-app-download/>, accessed on October 20, 2025.

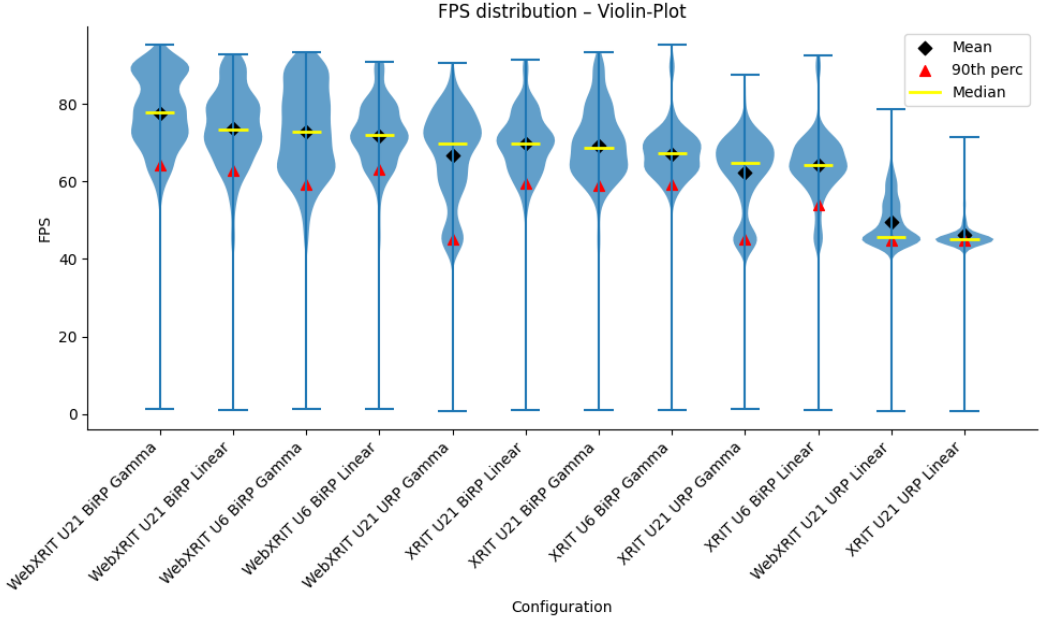


Fig. 6. FPS distributions for the 12 valid configurations (raw samples). Medians (yellow), means (black), and 90th-percentile FPS (red) are indicated.

De-correlation of FPS Data. Due to temporal coherence, adjacent frames in real-time rendering applications are often correlated in content and workload, and therefore frame times (and instantaneous FPS) tend to be serially dependent [43]. To assess this in our data, we applied Ljung–Box tests up to lag 150 (15 s) on the raw FPS time series, which revealed significant autocorrelation in all datasets.

To attenuate this dependence before inferential testing, each 60-second FPS time series was aggregated into non-overlapping 5-second blocks, each consisting of 50 raw samples. This procedure yielded $n = 12$ aggregated observations per configuration. Residual autocorrelation in these aggregates was again evaluated using the Ljung–Box test at lags 1–3 (15 s). The smallest observed p -value was $p = 0.0504$; all other minima exceeded this value. Thus, at $\alpha = 0.05$ we found no evidence of residual autocorrelation, supporting the use of standard unpaired tests on the aggregated data.

Pairwise Framework Comparisons. A Shapiro–Wilk test revealed that the aggregated data was not normally distributed, so for each of the six valid configuration pairs, we compared WebXRIT vs. XRIT using a two-sided Mann–Whitney U test on the 5-second aggregates. Family-wise error was controlled using Bonferroni adjustment across the six tests ($\alpha' = 0.05/6 \approx 0.0083$). We report the mean difference in FPS (ΔMean), the Bonferroni-adjusted p -value (p_{adj}), and the rank-biserial correlation (r_{rb}) as an effect size, which is interpreted via the probability of superiority following Vargha and Delaney [46]. Results are summarized in Table 2.

Three of the six comparisons remain significant after Bonferroni correction (both U6 BiRP conditions and U21 BiRP Gamma). These significant comparisons exhibit large effect sizes ($r_{\text{rb}} = 0.68$ ($A = 0.84$), 0.72 ($A = 0.86$), 0.85 ($A = 0.92$)), indicating a consistent performance advantage of WebXRIT over XRIT. The remaining three comparisons also showed positive mean differences in favor of WebXRIT, but did not reach the adjusted significance threshold.

Table 2. Pairwise comparisons of WebXRIT vs. XRIT on 5-second aggregates using Mann–Whitney U tests. ΔMean denotes the difference in mean FPS (WebXRIT minus XRIT). p_{adj} indicates Bonferroni-adjusted p -values ($\alpha' = 0.0083$). Effect sizes are reported as the rank-biserial correlation r_{rb} together with the equivalent Vargha–Delaney probability of superiority ($A = (r_{\text{rb}} + 1)/2$). Effect size magnitudes are interpreted following Vargha and Delaney [46]: negligible ($|r_{\text{rb}}| < 0.11$), small ($0.11 \leq |r_{\text{rb}}| < 0.28$), moderate ($0.28 \leq |r_{\text{rb}}| < 0.43$), and large ($|r_{\text{rb}}| \geq 0.43$). Asterisks mark results significant after Bonferroni correction.

Pair	ΔMean	U	p_{raw}	p_{adj}	r_{rb} (A)	Effect
WebXRIT U6 BiRP Gamma vs. XRIT U6 BiRP Gamma	5.87	121	0.0051	0.0306*	0.68 (0.84)	Large
WebXRIT U6 BiRP Linear vs. XRIT U6 BiRP Linear	7.61	124	0.0029	0.0177*	0.72 (0.86)	Large
WebXRIT U21 BiRP Gamma vs. XRIT U21 BiRP Gamma	8.46	133	0.0005	0.0029*	0.85 (0.92)	Large
WebXRIT U21 BiRP Linear vs. XRIT U21 BiRP Linear	4.00	110	0.0304	0.1823	0.53 (0.76)	Large
WebXRIT U21 URP Gamma vs. XRIT U21 URP Gamma	4.61	115	0.0141	0.0848	0.60 (0.80)	Large
WebXRIT U21 URP Linear vs. XRIT U21 URP Linear	3.40	113	0.0194	0.1162	0.57 (0.78)	Large

Mixed-effects model. To estimate a single overall framework effect, we fit a linear mixed-effects model to the 5-s FPS aggregates with FPS as the dependent variable, *Framework* (reference = WebXRIT; contrast = XRIT) as a fixed effect, and a random intercept for the *configuration context* (the six combinations of Unity version $\times$ render pipeline $\times$ color space). The model was estimated via REML. We analyzed 144 aggregated observations across six configuration contexts (24 per context; 12 WebXRIT and 12 XRIT within each context). The model converged successfully and indicated a robust framework effect (Appendix A, Table 3, Table 4). After accounting for systematic differences between configuration contexts, WebXRIT builds were associated with an average increase of approximately 5.7 FPS relative to XRIT, representing a statistically and practically meaningful performance advantage.

6 Discussion

Across all configurations, our results demonstrate superior runtime performance of the *WebXR Interaction Toolkit* (WebXRIT) compared to Unity’s *XR Interaction Toolkit* (XRIT). The four best-performing configurations (Table 1) all employ WebXRIT. Each of these configurations achieved a median frame rate of 72 FPS or higher, meeting or exceeding the defined minimum acceptance threshold for acceptable performance in WebXR applications. The best overall configuration, *WebXRIT U21 BiRP Gamma*, reached a median of 78 FPS. In contrast, the best configuration using XRIT (*XRIT U21 BiRP Linear*) achieved a median frame rate of approximately 70 FPS, falling just below the defined threshold. In terms of the 90th percentile FPS, none of the tested configurations reached the target threshold. The highest value observed among the WebXRIT configurations was approximately 64 FPS (*WebXRIT U21 BiRP Gamma*), whereas the top XRIT configuration achieved about 60 FPS (*XRIT U21 BiRP Linear*).

According to Unity’s best practices for performance optimization²⁵, potential differences in performance between the *WebXRIT* and the *XRIT* may be explained by variations in the number

²⁵ <https://unity.com/how-to/best-practices-for-profiling-game-performance>, accessed on October 20, 2025.

of draw calls, or in the CPU and GPU time per frame. In the Unity engine, a draw call denotes a command issued by the CPU to the graphics API, instructing the GPU to render a specific mesh with defined materials and shaders²⁶. Each draw call introduces CPU overhead for command submission and state management, while the corresponding GPU rendering time depends on the geometric and shading complexity processed per call. Applications with a large number of draw calls can therefore become CPU-bound, whereas those with complex shaders or large meshes may become GPU-bound. As described in section 4.2, all performance tests used the same virtual environment (with identical 3D meshes, materials, animations) and the same browser graphics API (WebGL 2.0). Consequently, neither the number of draw calls on the CPU side nor the geometric complexity processed by the GPU should vary between builds. However, an application can also become CPU-bound if certain tasks within the application logic take excessive processing time or if the implementation inadvertently increases the number of draw calls (e.g., by affecting draw call batching). To verify these assumptions, future measurements should include draw call counts as well as CPU and GPU time per frame.

It is notable that the four best-performing configurations, as well as all configurations in which performance differences between frameworks were statistically significant, use the *Built-in Render Pipeline* (BiRP). This finding is somewhat unexpected, as one might assume that Unity's more recent *Universal Render Pipeline* (URP), which is also recommended by Unity for web-based applications²⁷, would outperform the older BiRP. A plausible explanation lies in the maturity of the BiRP, which represents Unity's longest-maintained rendering system. Its extended development history suggests extensive optimization and stability across a wide range of platforms. However, the proprietary nature of Unity's rendering system prevents a more detailed analysis of the internal causes for the observed performance differences.

For our current solution, the BiRP appears to provide the best runtime performance. This also offers a practical advantage: many existing VR projects in Unity that were originally developed with the BiRP could be ported to WebXR builds with minimal modifications using WebXRIT. Nevertheless, URP performance could potentially be improved by leveraging new Unity engine technologies not available to BiRP. These include advanced draw-call batching through the *Scriptable Render Pipeline (SRP) Batcher*, which could reduce CPU overhead²⁸. Additionally, the URP already provides experimental support for the emerging browser graphics API *WebGPU*. WebGPU²⁹ is currently in the W3C Candidate Recommendation phase and is expected to succeed WebGL as the primary graphics standard for the web [9, 12]. Even when used with URP, WebXRIT configurations achieved median frame rates of up to approximately 70 FPS (*WebXRIT U21 URP Gamma*), and across all configurations with the URP, WebXRIT consistently outperformed XRIT, although these differences did not reach statistical significance.

The combination of the older Unity editor version (*U21*) and the current render pipeline (*URP*) resulted in the weakest overall performance across all tests, with median frame rates of approximately 46–50 FPS—slightly above half the display's refresh rate. This result can likely be explained by the fact that the newer *URP* is more tightly integrated into recent Unity editor versions. Consequently, the combination of *U21* and *URP* may suffer from suboptimal compatibility and limited optimization. The findings associated with this configuration are therefore of minor practical relevance for future WebXR application development, as upcoming projects are more likely to employ *Unity 6* in conjunction with the *URP*, which continues to receive performance and integration improvements in newer Unity releases.

²⁶ <https://docs.unity3d.com/6000.2/Documentation/Manual/optimizing-draw-calls.html>, accessed on October 20, 2025.

²⁷ <https://unity.com/how-to/profile-optimize-web-build>, accessed on October 20, 2025.

²⁸ <https://docs.unity3d.com/6000.2/Documentation/Manual/SRPBatcher.html>, accessed on October 20, 2025.

²⁹ <https://www.w3.org/TR/webgpu/>, accessed on October 20, 2025.

The interplay between the game engine, interaction toolkit, build configurations, and web graphics API in the creation of WebXR applications with Unity is inherently complex, with performance depending on a multitude of interacting factors. To enable the future development of easily accessible, browser-based IVR applications that offer rich interaction capabilities, these factors require continued investigation, and software solutions supporting the integration of WebXR into 3D engines should be further refined and optimized. Nevertheless, as demonstrated by our performance tests, the proposed *WebXR Interaction Toolkit* (WebXRIT) already enables the creation of performant IVR training scenarios within Unity.

7 Limitations & Future Work

The current version of the *WebXR Interaction Toolkit* (WebXRIT) is available as an open-source project at: <https://github.com/tino-hentschel/webxrit>. The toolkit is still under active development, and several technical limitations remain that will be addressed in these upcoming versions.

Currently, WebXRIT supports only controller-based input and does not yet provide hand-tracking capabilities, which are offered by many modern HMDs [48]. Although hand tracking can already be enabled in WebXR via the WebXR Export plugin, WebXRIT does not yet include dedicated components for handling such input modalities.

Through Unity's build pipeline, it is also already possible to create AR applications using the WebXR Export plugin and WebXRIT. However, specific AR components have not yet been developed or tested. Future iterations of the toolkit aim to extend all basic interaction types to AR contexts.

Although the VE used in our performance tests includes a UI that displays text elements, the current version of WebXRIT does not yet provide components supporting interaction with Unity UI elements such as buttons, sliders, or drop-down menus. These components are planned for inclusion in future versions of the toolkit to facilitate the development of hybrid 3D and 2D user interfaces in WebXR applications.

In terms of navigation, the current version of WebXRIT supports only room-scale real walking and joystick-based continuous movement. The latter method can induce motion sickness in some users [26]. Accordingly, future versions of the toolkit will include additional navigation techniques such as teleportation and snap turning to provide more comfort and accessibility across user groups.

In this paper, frame rate was selected as the primary performance metric to provide an initial assessment of the toolkit's runtime efficiency across different build configurations and in comparison to XRIT. Future studies should include additional performance metrics, such as CPU and GPU time per frame, and input latency, to better identify potential performance bottlenecks and to inform optimization strategies. Moreover, comparative analyses with XR frameworks and interaction toolkits from other engines, such as Unreal Engine³⁰, Godot Engine³¹, or the Wonderland Engine³², could provide further insights to derive best practices for WebXR development.

In addition, our performance evaluation was conducted using a single VE with a fixed level of scene complexity. This design choice was made to isolate the effect of the interaction toolkit itself by holding key complexity parameters, such as geometric detail and asset composition, constant across all conditions. While this allows for controlled comparisons between WebXRIT and XRIT, it limits the generalizability of the results to scenarios with different levels of visual and geometric complexity. Future work should therefore evaluate performance across multiple VEs with systematically varied complexity levels.

³⁰ <https://www.unrealengine.com>, accessed on October 20, 2025.

³¹ <https://godotengine.org>, accessed on October 20, 2025.

³² <https://wonderlandengine.com>, accessed on October 20, 2025.

Beyond performance, future research should also assess the user experience of WebXR training applications developed with WebXRIT, as well as the usability and developer experience of the toolkit itself.

Acknowledgments

The *WebXR Interaction Toolkit* (WebXRIT) was developed as part of the research project *Pflege Digital 2.0*³³ at the Hamburg University of Applied Sciences. The project focused on developing and evaluating digital solutions for nursing education and training and was funded by the German Federal Ministry of Health.

Some graphical elements and icons used in this work were obtained from Freepik and Flaticon (available at <https://www.freepik.com> and <https://www.flaticon.com>) and are used under the Creative Commons Attribution 4.0 License (CC BY 4.0). In addition, parts of the visualization design were adapted from example materials provided in the Vuer documentation (https://docs.vuer.ai/en/latest/examples/20_motion_controllers.html), licensed under the MIT License. We acknowledge the contributions of the respective authors and designers for making these resources publicly available.

References

- [1] Pedro Acevedo, Alejandra J. Magana, Bedrich Benes, and Christos Mousas. 2024. A Systematic Review of Immersive Virtual Reality in STEM Education: Advantages and Disadvantages on Learning and User Experience. *IEEE Access* 12 (2024), 189359–189386. doi:10.1109/ACCESS.2024.3489233
- [2] Alex Adkins, Ryan Canales, and Sophie Jörg. 2024. Hands or Controllers? How Input Devices and Audio Impact Collaborative Virtual Reality. In *30th ACM Symposium on Virtual Reality Software and Technology*. ACM, Trier Germany, 1–12. doi:10.1145/3641825.3687718
- [3] Lan Bao, Si Tran, Truong Nguyen, Hai Chien Pham, Lee Dongmin, and Chansik Park. 2022. Cross-platform virtual reality for real-time construction safety training using immersive web and industry foundation classes. *Automation in Construction* 143 (Nov. 2022), 104565. doi:10.1016/j.autcon.2022.104565
- [4] Woodrow Barfield and Claudia Hendrix. 1995. The effect of update rate on the sense of presence within virtual environments. *Virtual Reality* 1, 1 (June 1995), 3–15. doi:10.1007/BF02009709
- [5] María Lucía Barrón-Estrada, Ramón Zatarain-Cabada, Manuel Alberto Sotelo-Rivas, María Blanca Ibáñez, and Víctor Manuel Bátiz-Beltrán. 2025. BabloXR: An Authoring Tool for Developing WebXR Educational Applications. *Multimodal Technologies and Interaction* 9, 6 (May 2025), 52. doi:10.3390/mti9060052
- [6] Joanna Bergström, Tor-Salve Dalsgaard, Jason Alexander, and Kasper Hornbæk. 2021. How to Evaluate Object Selection and Manipulation in VR? Guidelines from 20 Years of Studies. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, Yokohama Japan, 1–20. doi:10.1145/3411764.3445193
- [7] Weichen Bi, Yun Ma, Deyu Tian, Qi Yang, Mingtao Zhang, and Xiang Jing. 2023. Demystifying Mobile Extended Reality in Web Browsers: How Far Can We Go?. In *Proceedings of the ACM Web Conference 2023*. ACM, Austin TX USA, 2960–2969. doi:10.1145/3543507.3583329
- [8] Gavin Buckingham. 2021. Hand Tracking for Immersive Virtual Reality: Opportunities and Challenges. *Frontiers in Virtual Reality* 2 (Oct. 2021). doi:10.3389/frvir.2021.728461
- [9] Peter W. S. Butcher, Andrea Batch, David Saffo, Blair MacIntyre, Niklas Elmqvist, and Panagiotis D. Ritsos. 2024. Is Native Naïve? Comparing Native Game Engines and WebXR as Immersive Analytics Development Platforms. *IEEE Computer Graphics and Applications* 44, 3 (May 2024), 91–98. doi:10.1109/MCG.2024.3367422
- [10] Augusto Cabrera-Duffaut, Ana María Pinto-Llorente, and Ana Iglesias-Rodríguez. 2024. Immersive learning platforms: analyzing virtual reality contribution to competence development in higher education—a systematic literature review. *Frontiers in Education* 9 (July 2024), 1391560. doi:10.3389/educ.2024.1391560
- [11] Andrea Cattaneo, Anna Savoldelli, Amin Amroussi, Maria Sofia Cavallo, Debora Pesenti, Maria Ragazzi, Cristian Rota, and Andrea Vitali. 2025. Virtual Reality with Moodle for Training in Telemedicine. *Studies in Health Technology and Informatics* 324 (April 2025), 228–233. doi:10.3233/SHTI250193
- [12] Satyadhyam Chickerur, Sankalp Balannavar, Pranali Hongekar, Aditi Prerna, and Soumya Jituri. 2024. WebGL vs. WebGPU: A Performance Analysis for Web 3.0. *Procedia Computer Science* 233 (Jan. 2024), 919–928. doi:10.1016/j.procs.2024.03.281

³³<https://www.pflegedigital20.de/>

- [13] Fabio J. Cortés Rodríguez, Gianfranco Frattini, Sittha Phloi-Montri, Fernando Teixeira Pinto Meireles, Danaé A. Terrien, Sergio Cruz-León, Matteo Dal Peraro, Eva Schier, Kresten Lindorff-Larsen, Taweetham Limpanuparb, Diego M. Moreno, and Luciano A. Abriata. 2025. MolecularWebXR: Multiuser discussions in chemistry and biology through immersive and inclusive augmented and virtual reality. *Journal of Molecular Graphics and Modelling* 135 (March 2025), 108932. doi:10.1016/j.jmkgm.2024.108932
- [14] Fernanda De La Torre, Cathy Mengying Fang, Han Huang, Andrzej Banburski-Fahey, Judith Amores Fernandez, and Jaron Lanier. 2024. LLMR: Real-time Prompting of Interactive Worlds using Large Language Models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA, 1–22. doi:10.1145/3613904.3642579
- [15] Ralf Doerner, Wolfgang Broll, Paul Grimm, and Bernhard Jung (Eds.). 2022. *Virtual and Augmented Reality (VR/AR): Foundations and Methods of Extended Realities (XR)*. Springer International Publishing, Cham. doi:10.1007/978-3-030-79062-2
- [16] Aritra Ghosh. 2022. Challenges in Educational Game Development. *International Journal of Darshan Institute on Engineering Research & Emerging Technology* 11 (July 2022), 54–60. doi:10.32692/IJDI-ERET/11.1.2022.2209
- [17] Xi Guo and Imran Mogra. 2022. Using Web 3D and WebXR Game to Enhance Engagement in Primary School Learning. In *2022 IEEE International Symposium on Multimedia (ISM)*. IEEE, Italy, 181–184. doi:10.1109/ISM55400.2022.00040
- [18] Ilham Achmad Al Hafidz, Sritrusta Sukaridhoto, Muhammad Udin Harun Al Rasyid, Rizqi Putri Nourma Budiarti, Rachma Rizqina Mardhotillah, Rizki Amalia, Evianita Dewi Fajrianti, and Naufal Adi Satrio. 2021. Design of Collaborative WebXR for Medical Learning Platform. In *2021 International Electronics Symposium (IES)*. IEEE, Surabaya, Indonesia, 499–504. doi:10.1109/IES53407.2021.9593951
- [19] D. Hamilton, J. McKechnie, E. Edgerton, and C. Wilson. 2021. Immersive virtual reality as a pedagogical tool in education: a systematic literature review of quantitative learning outcomes and experimental design. *Journal of Computers in Education* 8, 1 (March 2021), 1–32. doi:10.1007/s40692-020-00169-2
- [20] Kirana Hanfati, Sritrusta Sukaridhoto, Dwi Kurnia Basuki, Rizqi Putri Nourma Budiarti, Evianita Dewi Fajrianti, and Ilham Achmad Al Hafidz. 2022. Design and Implementation of WebXR Health Learning Module Application. In *2022 International Electronics Symposium (IES)*. IEEE, Surabaya, Indonesia, 632–637. doi:10.1109/IES55876.2022.9888325
- [21] Panpan Huai, Yao Li, Xiaomeng Wang, Linghui Zhang, Nan Liu, and Hui Yang. 2024. The effectiveness of virtual reality technology in student nurse education: A systematic review and meta-analysis. *Nurse Education Today* 138 (July 2024), 106189. doi:10.1016/j.nedt.2024.106189
- [22] Afzal Hussain, Haad Shakeel, Faizan Hussain, Nasir Uddin, and Turab Latif Ghouri. 2020. Unity Game Development Engine: A Technical Survey. *University of Sindh Journal of Information and Communication Technology* 4 (Oct. 2020), 73–81.
- [23] Janine Jongbloed, Rawad Chaker, and Elise Lavoué. 2024. Immersive procedural training in virtual reality: A systematic literature review. *Computers & Education* 221 (Nov. 2024), 105124. doi:10.1016/j.compedu.2024.105124
- [24] Vix Kemanji Ketoma, Rene Mpwadina, and Gerrit Meixner. 2022. Virtual Reality Assembly of Physical Parts: The Impact of Interaction Interface Techniques on Usability and Performance. In *Virtual, Augmented and Mixed Reality: Applications in Education, Aviation and Industry*, Jessie Y. C. Chen and Gino Fragomeni (Eds.). Springer International Publishing, Cham, 350–368. doi:10.1007/978-3-031-06015-1_24
- [25] Vix Kemanji Ketoma, Jean Vanderdonckt, and Gerrit Meixner. 2023. Towards Flexible Authoring and Personalization of Virtual Reality Applications for Training. *Proceedings of the ACM on Human-Computer Interaction* 7, EICS (June 2023), 1–37. doi:10.1145/3593241
- [26] Chaowanan Khundam, Varunyut Vorachart, Patibut Preeyawongsakul, Witthaya Hosap, and Frédéric Noël. 2021. A Comparative Study of Interaction Time and Usability of Using Controllers and Hand Tracking in Virtual Reality Training. *Informatics* 8, 3 (Sept. 2021), 60. doi:10.3390/informatics8030060
- [27] Konstantinos Koukourikos, Areti Tsaloglidou, Lambrini Kourkouta, Ioanna Papathanasiou, Christos Iliadis, Aikaterini Fratzana, and Aspasia Panagiotou. 2021. Simulation in Clinical Nursing Education. *Acta Informatica Medica* 29, 1 (2021), 15. doi:10.5455/aim.2021.29.15.20
- [28] Aliane Loureiro Krassmann, Miguel Melo, Darque Pinto, Bruno Peixoto, Maximino Bessa, and Magda Bercht. 2019. What Is the Relationship between the Sense of Presence and Learning in Virtual Reality? A 24-Year Systematic Literature Review. *PRESENCE: Virtual and Augmented Reality* 28 (Jan. 2019), 247–265. doi:10.1162/pres_a_00350
- [29] Tom Lautenbach, Iana Podkosova, and Hannes Kaufmann. 2024. ProjectVerse: Making VR Research Visible in VR. *GI VR / AR Workshop*. doi:10.18420/vrar2024_0009
- [30] Weichao Lin, Liang Chen, Wei Xiong, Kang Ran, and Anlan Fan. 2025. Measuring the sense of presence and learning efficacy in immersive virtual assembly training. *International Journal of Mechanical Engineering Education* 0, 0 (Jan. 2025), 03064190241308659. doi:10.1177/03064190241308659
- [31] Benjamin G. P. Linares-Vargas and Segundo E. Cieza-Mostacero. 2024. Interactive virtual reality environments and emotions: a systematic review. *Virtual Reality* 29, 1 (Nov. 2024), 3. doi:10.1007/s10055-024-01049-1

- [32] Kaiyan Liu, Nan Wu, and Bo Han. 2023. Demystifying Web-based Mobile Extended Reality Accelerated by WebAssembly. In *Proceedings of the 2023 ACM on Internet Measurement Conference*. ACM, Montreal QC Canada, 145–153. doi:10.1145/3618257.3624833
- [33] Kai Liu, Weiwei Zhang, Wei Li, Ting Wang, and Yanxue Zheng. 2023. Effectiveness of virtual reality in nursing education: a systematic review and meta-analysis. *BMC Medical Education* 23, 1 (Sept. 2023), 710. doi:10.1186/s12909-023-04662-x
- [34] David Llanos-Ruiz, Víctor Abella-García, and Vanesa Ausín-Villaverde. 2025. Virtual Reality in Higher Education: A Systematic Review Aligned with the Sustainable Development Goals. *Societies* 15, 9 (Sept. 2025), 251. doi:10.3390/soc15090251
- [35] Blair MacIntyre and Trevor F. Smith. 2018. Thoughts on the Future of WebXR and the Immersive Web. In *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. IEEE, Munich, Germany, 338–342. doi:10.1109/ISMAR-Adjunct.2018.00099
- [36] Guido Makransky and Gustav B. Petersen. 2021. The Cognitive Affective Model of Immersive Learning (CAMIL): a Theoretical Research-Based Model of Learning in Immersive Virtual Reality. *Educational Psychology Review* 33, 3 (Sept. 2021), 937–958. doi:10.1007/s10648-020-09586-2
- [37] Alexander Masurovsky, Paul Chojeci, Detlef Runde, Mustafa Lafci, David Przewozny, and Michael Gaebler. 2020. Controller-Free Hand Tracking for Grab-and-Place Tasks in Immersive Virtual Reality: Design Elements and Their Empirical Study. *Multimodal Technologies and Interaction* 4, 4 (Dec. 2020), 91. doi:10.3390/mti4040091
- [38] Michael Meehan, Brent Insko, Mary Whitton, and Frederick P. Brooks. 2002. Physiological measures of presence in stressful virtual environments. *ACM Trans. Graph.* 21, 3 (July 2002), 645–652. doi:10.1145/566654.566630
- [39] Şahin Mercan and Pinar Onay Durdu. 2017. Evaluating the Usability of Unity Game Engine from Developers' Perspective. In *2017 IEEE 11th International Conference on Application of Information and Communication Technologies (AICT)*. IEEE, Moscow, Russia, 1–5. doi:10.1109/ICAICT.2017.8687303
- [40] Rakhi Mishra, Hemlata, and Divya Trivedi. 2023. Simulation-based learning in nursing curriculum- time to prepare quality nurses: A systematic review and meta-analysis. *Heliyon* 9, 5 (May 2023), e16014. doi:10.1016/j.heliyon.2023.e16014
- [41] Leon Pietschmann, Paul-David Zuercher, Erik Bubík, Zhutian Chen, Hanspeter Pfister, and Thomas Bohné. 2023. Quantifying the Impact of XR Visual Guidance on User Performance Using a Large-Scale Virtual Assembly Experiment. doi:10.48550/arXiv.2308.03390 arXiv:2308.03390 [cs].
- [42] Jaziar Radianti, Tim A. Majchrzak, Jennifer Fromm, and Isabell Wohlgenannt. 2020. A systematic review of immersive virtual reality applications for higher education: Design elements, lessons learned, and research agenda. *Computers & Education* 147 (April 2020), 103778. doi:10.1016/j.compedu.2019.103778
- [43] Daniel Scherzer, Lei Yang, Oliver Mattausch, Diego Nehab, Pedro V. Sander, Michael Wimmer, and Elmar Eisemann. 2012. Temporal Coherence Methods in Real-Time Rendering. *Computer Graphics Forum* 31, 8 (Dec. 2012), 2378–2408. doi:10.1111/j.1467-8659.2012.03075.x
- [44] Swati Singh and Amanpreet Kaur. 2022. Game Development using Unity Game Engine. In *2022 3rd International Conference on Computing, Analytics and Networks (ICAN)*. IEEE, Piscataway, NJ, USA, 1–6. doi:10.1109/ICAN56228.2022.10007155
- [45] Anthony Steed and Jiaqi Lai. 2025. Comparison of hand tracking-based and controller-based interaction in a consumer virtual reality game. *Virtual Reality* 29, 3 (July 2025), 120. doi:10.1007/s10055-025-01190-5
- [46] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the "CL" Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132. doi:10.2307/1165329
- [47] Rafael Villena-Taranilla and Pascual D. Diago. 2025. Challenges and Implications of Virtual Reality in History Education: A Systematic Review. *Applied Sciences* 15, 10 (May 2025), 5589. doi:10.3390/app15105589
- [48] Jan-Niklas Voigt-Antons, Tanja Kojić, Danish Ali, and Sebastian Möller. 2020. Influence of Hand Tracking as a way of Interaction in Virtual Reality on User Experience. doi:10.48550/arXiv.2004.12642 arXiv:2004.12642 [cs].
- [49] Jialin Wang, Rongkai Shi, Wenxuan Zheng, Weijie Xie, Dominic Kao, and Hai-Ning Liang. 2023. Effect of Frame Rate on User Experience, Performance, and Simulator Sickness in Virtual Reality. *IEEE Transactions on Visualization and Computer Graphics* 29, 5 (May 2023), 2478–2488. doi:10.1109/TVCG.2023.3247057
- [50] Biao Xie, Huimin Liu, Rawan Alghofaili, Yongqi Zhang, Yeling Jiang, Flavio Destri Lobo, Changyang Li, Wanwan Li, Haikun Huang, Mesut Akdere, Christos Mousas, and Lap-Fai Yu. 2021. A Review on Virtual Reality Skill Training Applications. *Frontiers in Virtual Reality* 2 (April 2021), 645153. doi:10.3389/frvir.2021.645153
- [51] Difeng Yu, Tilman Dingler, Eduardo Velloso, and Jorge Goncalves. 2025. Object Selection and Manipulation in VR Headsets: Research Challenges, Solutions, and Success Measurements. *Comput. Surveys* 57, 4 (April 2025), 1–34. doi:10.1145/3706417
- [52] Chumin Zhao, Andrea S. Kim, Ryan Beams, and Aldo Badano. 2022. Spatiotemporal image quality of virtual reality head mounted displays. *Scientific Reports* 12, 1 (Nov. 2022), 20235. doi:10.1038/s41598-022-24345-9

[53] Misbahu Zubair and Nnamdi Anyameluhor. 2021. How long do you want to maintain this thing? Understanding the Challenges Faced by WebXR Creators. In *The 26th International Conference on 3D Web Technology*. ACM, Pisa Italy, 1–4. doi:10.1145/3485444.3495181

A Mixed-Effects Model Tables

Table 3. Mixed-effects model summary for FPS (5-s aggregates).

Model	MixedLM (REML)
Dependent variable	FPS
No. Observations	144
No. Groups	6
Min. group size	24
Max. group size	24
Mean group size	24.0
Scale (σ^2)	29.6961
Log-Likelihood	−457.1475
Converged	Yes

Table 4. Fixed effects and random-effects summaries from the mixed-effects model (p-values from two-sided Wald z-tests).

Term	Coef.	SE	<i>z</i>	<i>p</i>	2.5%	97.5%
Fixed effects						
Intercept	68.738	3.825	17.970	3.36×10^{-72}	61.241	76.236
Framework[T.XRIT]	-5.658	0.908	-6.230	4.67×10^{-10}	-7.438	-3.878
Random effects						
Random intercept vari- ance (configuration con- text)	85.318	10.227				
Random intercept SD (configuration context)	9.24					
Residual SD ($\sqrt{\text{Scale}}$)	5.45					