2025 40th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)

A Test Automation Framework for User Interaction in Extended Reality Applications

Ruizhen Gu and José Miguel Rojas
School of Computer Science, University of Sheffield, Sheffield, UK
{rgu10, j.rojas}@sheffield.ac.uk

***Abstract*—Extended Reality (XR) technologies deliver immersive user experiences across diverse application domains but pose unique testing challenges due to their spatial interaction paradigms. Existing approaches test XR applications through scene navigation and interaction triggering, yet they fail to synthesise realistic spatial input via specialised XR devices. These devices include 6 degrees of freedom controller gestures and are essential for modern XR experiences. To address this gap, we present INTENXION, a test automation framework for validating user interactions in Unity XR applications. We develop a taxonomy of XR user interactions to inform the design of INTENXION, ensuring it supports a diverse range of XR interaction types. We conduct a case study to demonstrate INTENXION's capability, using eight representative interaction object types selected from industrial XR interaction design guidelines. INTENXION provides support for developing intuitive interaction test scripts, including both action sequences and assertions for expected behaviours. Our results demonstrate that INTENXION can cover the intended functionalities of all eight object types.**

***Index Terms*—Extended Reality, Software Testing, Test Automation, Testing Framework**

I. Introduction

Extended Reality (XR) is an umbrella term for Augmented, Mixed, and Virtual Reality (AR, MR, and VR, respectively). XR applications (hereafter, XR apps) are software programs designed for XR-compatible devices that create virtually organised spaces populated with interactive digital content, allowing users to explore and engage with immersive experiences. Realistic, immersive, and interaction-rich experiences are delivered through head-mounted displays (HMDs) such as the Meta Quest 3[1], coupled with input devices like handheld controllers. The development environments for these XR apps have matured considerably in recent years. For example, Unity has emerged as one of the mainstream platforms for XR development[2], supporting major XR operating systems like Meta Horizon[3] and Android XR[4].

As XR apps can span diverse domains and platforms, their development and testing processes have become substantially more complex, highlighting the need for more comprehensive testing approaches [1]. XR app testing can target various objectives, including functionality and usability, and specific targets ranging from object placement accuracy [2] to cybersickness detection [3]. Among these diverse testing concerns, *functionality* and *user interaction* remain the most critical objective and target, respectively [4]. Interaction testing is essential because unpredictable human behaviours and dynamic XR environments prevent reliable outcome prediction [5]. Rigorous testing methodologies must therefore validate consistent software responses across variable conditions.

Existing efforts on XR testing research struggle to simulate realistic user inputs, such as natural gestures or haptic feedback through dedicated XR devices like controllers [4]. Moreover, current work has predominantly focused on conventional UI (user interface) elements in the XR environments. These UI elements are fundamentally the same as mobile GUI (graphic user interface) components, operable through point-and-click interactions (e.g., via cursor or taps). This narrow focus overlooks modern XR's core paradigm of spatial, realistic interactions. The resulting gap prevents effective evaluation of XR apps with advanced spatial interaction capabilities.

To address this gap, we present INTENXION, a test automation framework for validating user interactions in XR apps. Our approach draws inspiration from established GUI testing frameworks like UIAUTOMATOR [6] and APPIUM [7], adapting their principles of simulating user actions (e.g., taps, swipes) to the spatial interactions of XR.

To facilitate systematic validation, we first establish a novel taxonomy of XR user interactions derived from human–computer interaction (HCI) principles and industry standards. The taxonomy presents a three-layered categorisation, including 1) *core tasks* representing user intents, 2) *atomic actions* serving as fundamental building blocks for interactions, and 3) *compositional patterns* governing the combination of atomic actions into complex, meaningful interactions.

This taxonomy directly informs INTENXION's design, mapping the interaction tasks to testable actions. INTENXION incorporates three key features, including 1) simulation of complex interaction behaviours via an intuitive API sets, 2) assertions for validating interaction outcomes, and 3) native support for authoring and executing compositional interaction patterns. To assess INTENXION's capability in supporting the development of XR interaction tests, we select eight distinct XR interactable object types as subjects. These objects represent diverse interaction patterns and principles from our taxonomy, such as cases with constrained movements. For each object type, we develop a dedicated test case using INTENXION. The tests comprise the action sequences to activate the intended object functionalities, and assertions to validate the expected object behaviours. The results show that

[1] https://www.meta.com/quest/quest-3/ [2] https://unity.com/solutions/xr
[3] https://developers.meta.com/horizon/ [4] https://www.android.com/xr/

the test cases developed with INTENXION cover all intended behaviours with appropriate validation mechanisms.

The contributions of this paper are as follows:

- **A taxonomy of XR interactions** spanning three different layers, covering the core tasks, atomic actions, and compositional interaction patterns.
- **A mapping of XR interaction tasks into testable actions**, establishing the foundation for systematic XR interaction testing approaches.
- **INTENXION, a test automation framework for user interactions in Unity XR apps.** Through a case study covering eight distinct interactable object types, we demonstrate INTENXION's ability to support developing test action sequences and interaction outcome assertions. Source code, case study assets, and test scripts are available at: https://github.com/ruizhengu/IntenXion.

This paper is structured as follows. Section II introduces foundational concepts and related work. Section III presents our XR interaction taxonomy. Section IV details INTENXION's design and implementation. Section V presents our case study and results. Section VI examines implications of this work, and Section VII summarises our work and outlines future work.

II. BACKGROUND AND RELATED WORK

A. XR Development with Unity

Unity XR projects are organised by means of *scenes*, each containing fundamental entities named *GameObjects* (GOs for short). Developers can extend GOs' functionality by configuring *components* and attaching *custom scripts* to define GO's specific behaviours and interaction patterns.

The XR Interaction Toolkit [8] (XRI) is Unity's development framework for XR experiences. It encompasses three primary features: 3D Interaction, UI Interaction, and Locomotion. The 3D interaction system, specifically, is essential for spatial user interaction. It comprises two script-based components attached to GOs: (i) *Interactors* handle user input and initiate interactions with interactable objects, and (ii) *Interactables* respond to interactors, defining specific behaviours for interactions. For instance, a controller interactor can grab a ball object with XRI's `XR Grab Interactable` component. The *interactor-interactable* interaction paradigm is adopted across various XR development platforms, including Meta and Unreal Engine (although the terms may differ).

B. XR Scene Testing

Scene testing is a type of testing that focuses on exploring XR scenes through two primary tasks: scene navigation and interaction event triggering [4]. Wang et al. pioneered VR scene testing with VRTEST [9] and VRGUIDE [10], introducing automated exploration techniques including random and greedy-based methods. These tools established valuable foundations for XR scene testing.

While effective for exploration, existing techniques lack support for modern XR experiences with interactions via dedicated input devices like controllers. INTENXION advances the field by simulating realistic controller inputs, enabling testing of complex spatial interactions. This approach complements scene testing methodologies by adding specialised validation capabilities for XR apps with rich interaction patterns.

C. Test Automation

Test automation and *automated testing* are two related but distinct terms in software testing. Test automation refers to the automation of test *execution* (e.g., using manually created test data), while automated testing automates both test *generation* and *execution*. While we acknowledge varied interpretations, we maintain this distinction throughout this paper.

Test automation typically employs script-based frameworks that execute predefined test cases. Industry-standard tools like APPIUM for Android GUI testing demonstrate how scripted approaches provide intuitive GUI validation [11]. In the XR domain, Youkai [12] is a unit testing framework for Unity VR/AR apps that rely on manually-written test scripts. However, no existing work addresses *3D user interaction testing* for XR, automating the validation of spatial interactions. This gap is the focus of our tool INTENXION.

III. TAXONOMY OF XR USER INTERACTIONS

We establish a three-tiered taxonomy of XR interactions, synthesising from HCI research and industrial practices. It includes high-level tasks, atomic actions as interaction building blocks, and compositional patterns for complex behaviours. It structures interactions from abstract objectives to implementable actions, providing a foundation for systematic XR testing and directly informing INTENXION's design (§ IV).

To ensure term consistency throughout this paper, we clarify that "interaction" denotes a complete bidirectional exchange between a user and the XR app (e.g., selecting an object). On the other hand, "action" refers to a specific, discrete step performed by the user (e.g., pressing a controller button). In other words, an "interaction" is a sequence of "actions".

A. Core Task Categories

XR interaction revolves around three core spatial tasks [5]: 1) *Selection:* target acquisition and activation; 2) *Manipulation:* object transformation; 3) *Navigation:* viewpoint movement. These categories represent high-level user intents, forming the key objectives for XR interaction testing.

B. Atomic Actions

The realisation of XR tasks relies on the atomic actions enabled through input devices. Modern XR experiences typically incorporate 6DoF controllers (e.g., Meta Quest Touch Plus[5]). The controllers enable interaction through two essential input modalities: the *trigger* for initiating actions (e.g., weapon firing), and the *grab* for grasping and manipulating objects.

When combined with *controller movement*, these modalities enable four basic interaction tasks [13]. These tasks include: 1) *Pointing:* spatial targeting of objects; 2) *Selection:* initiating/confirming interaction intents; 3) *Translation:* relocating objects; 4) *Rotation:* reorienting objects.

[5] https://www.meta.com/quest/accessories/quest-touch-plus-controller/

C. Compositional Patterns

Complex behaviours emerge through the composition of atomic actions governed by design principles [14, 15, 16, 17]. These principles include: 1) *Constrained Physics:* axis-limited movements (e.g., pulling a lever); 2) *State Persistence:* maintaining action contexts (e.g., grab holding during navigation); 3) *Sequential Patterns:* ordered atomic action sequences, e.g., $\langle$grab and hold $\rightarrow$ spatial translation $\rightarrow$ release$\rangle$.

IV. INTENXION TEST LIBRARY

Building upon the XR interaction taxonomy (§ III), we implement its core tasks, atomic actions, and compositional patterns into a test automation framework. This section details the mapping of interaction tasks to testable actions and the technical implementation for simulating XR interaction and validating interaction outcomes. Figure 1 illustrates the overview of developing XR interaction tests with INTENXION.

A. Testable Interaction Mapping

Building on our interaction taxonomy (§ III), we map the high-level interaction tasks to executable test actions across the categories of *navigation*, *selection*, and *manipulation*. Each category defines specific test responsibilities and required interactions, and test actions. The test responsibilities are explained as follows, and Table I summarises the mapping of interaction tasks and test actions.

1) Navigation: Navigation involves positioning and orienting the user's viewpoint within the environment, independent of controller inputs. This prepares the virtual user (i.e., a simulated agent representing the user within the XR environment) for subsequent interactions by establishing optimal spatial relationships to target objects through viewpoint transformation.

2) Selection: Selection requires controller-driven targeting and activation. Targeting refers to the spatial alignment with interactable objects through controller positioning. Activation is the successful functional engagement with the objects via trigger and grab inputs. Crucially, interaction *initiation* (attempted activation) does not guarantee functional *activation*. For instance, misaligned controller positioning would obstruct intended functionality despite initiation attempts.

Furthermore, interaction activation supports two temporal modalities. This encompasses both instant actions (e.g., discrete button presses) and continuous interactions requiring state persistence (e.g., button holding) as identified in § III.

3) Manipulation: Manipulation enables spatial transformation of objects through sustained controller engagement, utilising the continuous grab actions (§ III). This task encompasses two fundamental operations: translation and rotation (§ III-B).

B. Test Library Implementation

INTENXION is designed as a test automation framework for XR user interactions, compatible with the Unity Test Framework (UTF) [18]. The relationship resembles combining unit testing frameworks like JUnit[6] and unittest[7] with APPIUM for GUI testing. UTF offers a similar test infrastructure, including test case management, execution control, and reporting [11], while INTENXION delivers XR-specific capabilities.

[6] https://junit.org/ [7] https://docs.python.org/3/library/unittest.html

TABLE I: Interactions and test actions required for XR interaction testing tasks

Tasks	Required Interactions	Test Actions
Navigation	Position viewpoint near target	Viewpoint movement
Selection	Controller alignment, Trigger (instant/continuous), Grab (instant/continuous)	Controller movement, Grab, Trigger
Manipulation	Spatial transformation, Continuous grab	Controller movement, Grab

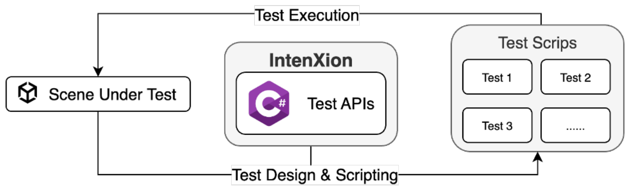

Fig. 1: Overview of INTENXION

We use Unity's XR Interaction Simulator [19] to generate controller inputs for atomic actions. This simulator indirectly manipulates camera or controller positions through keyboard and mouse emulations, rather than direct pose control. For instance, pressing the W key moves the controller forward.

Table II lists all the APIs provided by INTENXION, covering the tasks of *Navigation*, *Selection*, and *Manipulation*, aligning with our mapping (§ IV-A). Moreover, an *Assertion* API set is provided to enable the verification of the interaction outcomes. Note that while the manipulation API supports pitch/yaw rotations (e.g., `RotationUp`), it do not support roll (longitudinal) rotations due to simulator constraints.

INTENXION is designed to be extensible, enabling custom interactions and assertions for specialised interaction requirements. We now detail two core capabilities: compositional interaction handling and assertion mechanisms.

1) Compositional Interaction Handling: XR interactions differ fundamentally from 2D interactions. While 2D interactions often produce discrete state changes (e.g., activity navigation via button taps in Android apps), XR requires *sequential composition* of actions to achieve meaningful interactions.

To handle the compositional nature of XR interactions (§ III), INTENXION adopts APPIUM's *action chains* mechanism [20]. It facilitates 1) programmatic construction of ordered action sequences, 2) intuitive authoring of complex interaction patterns, and 3) human-readable test scripting.

Figure 2 demonstrates a test case developed with INTENXION to validate a compositional interaction that involves grab, trigger, and rotation actions. The test begins with viewpoint navigation to the target object (lines 5-7), followed by controller alignment with the target to initiate the interaction (lines 9-11). The action sequence applied to the target involves continuous grabbing (line 14), yaw rotation (line 15), an

TABLE II: Test APIs Provided by INTENXION

Tasks	Test APIs	Functions
Navigation	`NavigateTo`	Position user near object
Selection	`MoveControllerTo`	Aligns controller with object
	`Grab`	Instant object grabbing
	`GrabHold`	Continuous object holding
	`Trigger`	Instant action triggering
	`TriggerHold`	Continuous action triggering
Manipulation	`MoveUp`	Translation: heave upward
	`MoveDown`	Translation: heave downward
	`MoveForward`	Translation: surge forward
	`MoveBackward`	Translation: surge backward
	`MoveLeft`	Translation: sway leftward
	`MoveRight`	Translation: sway rightward
	`RotateUp`	Rotation: pitch down
	`RotateDown`	Rotation: pitch up
	`RotateLeft`	Rotation: yaw left
	`RotateRight`	Rotation: yaw right
	`ReleaseAllKeys`	Release all pressed keys
Assertion	`AssertGrabbed`	Confirm object is grabbed
	`AssertTriggered`	Confirm object is triggered
	`AssertTranslated`	Confirm object translation
	`AssertRotated`	Confirm object rotation

```
1  var blaster = FindGameObject("Blaster");
2  var blasterRot = blaster.transform.rotation;
3  Assert.IsNotNull(blaster, "Blaster not found");
4  // 1. Navigate to the blaster
5  yield return new ActionBuilder()
6         .NavigateTo(origin, blaster)
7         .Execute();
8  // 2. Move the controller to the blaster
9  yield return new ActionBuilder()
10        .MoveControllerTo(controller, blaster)
11        .Execute();
12 // 3. Grab hold and trigger the blaster
13 var action = new ActionBuilder();
14 action.GrabHold(1.0f)
15        .RotateLeft(0.1f)
16        .Trigger()
17        .ReleaseAllKeys();
18 yield return action.Execute();
19 AssertGrabbed(blaster, "Blaster ungrabbed");
20 AssertTriggered(blaster, "Blaster untriggered");
21 AssertRotated(blaster, blasterRot, "Blaster
   ↪  unrotated");
```

Fig. 2: A test case developed with INTENXION: Navigating to an object, grab holding, rotating, triggering, and releasing (action chains highlighted)

instant trigger (line 16), and releasing all keys at the end (line 17). Finally, assertions validate the interaction outcomes, confirming successful grabbing, triggering, and rotation of the target object (lines 19-21). The example showcases INTENXION's ability to compose test actions into meaningful interactions that exercise object functionality, coupled with validations of expected outcomes through assertions.

2) Assertion: Assertions serve as code-based test oracles, validating expected behaviours in specific scenarios [21] and are essential for systematic test automation [22]. INTENXION provides specialised assertion APIs (Table II) that implement two complementary validation approaches:

- **Intearction state validation** that leverages Unity XRI's interaction event system [23], registering callback listeners on target objects. They monitor real-time state changes, directly confirming interaction activation. APIs include `AssertGrabbed` and `AssertTriggered`.
- **Object property comparison** that compares transform properties of objects before and after interactions. APIs include `AssertRotated` and `AssertTranslated`.

While these core assertions capture fundamental XR interaction characteristics, complex custom interactions may require specialised validators. To address this, INTENXION's assertion APIs provide extensibility for developing custom assertions tailored to unique interaction events and object properties. Consider Unity's *VR Template* scene [24], where objects implement custom interaction logic through scripts like `XRKnob`. The script includes a *value* property that represents the dial's current rotational position within a defined range. The expected interaction outcome involves changing this *value* property through grab-and-rotate sequences. To validate this, we retrieve the property before and after the interaction using calls such as `dial.GetComponent<XRKnob>().value`. After executing the test actions, an assertion compares the two values, where a change confirms that the interaction successfully triggered the intended functionality. The example demonstrates how INTENXION's extensibility supports testing of domain-specific interaction patterns.

While these custom validators are not included in INTENXION's built-in assertion API set, developers can adapt our test case implementations to create application-specific validations. This extensibility addresses a fundamental reality that, due to the diverse and custom nature of XR apps, built-in assertions cannot anticipate all domain-specific interaction patterns.

V. CASE STUDY

To demonstrate INTENXION's practical utility in XR testing, we conduct a case study using eight representative interactable object types derived from industrial XR design guidelines [14, 17]. These object types require distinct interactions to fully exercise functionality. They sufficiently cover our established interaction taxonomy while representing common XR interaction scenarios. The object types include: 1) *Button:* instant activation with precise targeting; 2) *Door:* sequential constrained translations of handle and door; 3) *Drawer:* constrained linear translation with persistent grab state; 4) *Dial:* constrained rotation with persistent grab state; 5) *Lever:* constrained translation along fixed paths with persistent grab state; 6) *Slider:* constrained linear translation with persistent grab state; 7) *Socket:* continuous grab with translation for object placement; 8) *Weapon:* continuous grab with trigger activation.

We set up our study environment using Unity's XR Interaction Toolkit Examples [25], as it provides prebuilt assets demonstrating XRI's functionality, including all eight interactable object types. We develop a prototype XR scene containing these objects to showcase INTENXION's test automation capabilities and conduct the study on a MacBook Pro (M3 Pro, 36 GB RAM, macOS 15.6) running Unity 6000.0.45f1 with XRI v3.1.1. The case study scene and test scripts are available at: https://github.com/ruizhengu/IntenXion.

A. Results

To develop the test cases for the eight object types, one developer (the first author) manually created one test suite with eight test cases. The test suite also includes one setup function to configure the test setting, including loading the prototype scene and identifying the virtual user and the controller.

The developer spent around three hours developing and executing the test cases, ensuring that the test cases could sufficiently cover the object interaction functionalities. Figure 2 demonstrates the test case for the *Weapon* object type. Among all eight test cases, the number of lines of code yields the results of both average and median are around 18.

For all eight representative interactable object types, we develop dedicated test cases using INTENXION's APIs. Each test case incorporates an action sequence to exercise the target objects' functionalities. Moreover, it includes assertions to validate expected outcomes, reflecting the objects' functional integrity, after interactions. For example, a test for a *Door* object involves unlocking the handle and then pushing the door to open it. The included `AssertRotated` verifies that the door is rotated (around the hinge) to confirm that the door is opened. Similarly, `AssertTranslated` assesses whether a *Drawer* object is successfully pulled out. We evaluate test quality using coverage metrics that assess how well the tests exercise intended interaction functionalities through action executions and assertion validations (detailed in § VI-A).

The case study confirms INTENXION's practical utility for real-world XR testing grounded in industrial XR interaction design principles. INTENXION successfully enables test case development across all interaction requirements for diverse object types. The incorporated assertions utilised either built-in INTENXION APIs or custom validators developed for object-specific behaviours (discussed in § VI-B).

The approach ensures sufficient coverage of interaction patterns while validating functional correctness for all test subjects. These results validate INTENXION as an effective XR spatial interaction testing solution, providing real-world applicability through industrial-aligned scenarios. It offers intuitive, testable functionality across diverse interaction designs, as well as actionable validation capabilities.

VI. DISCUSSION

A. Test Case Quality

We assess test case quality by examining whether tests developed with INTENXION fully exercise object functionalities. For our eight representative interactable object types, INTENXION achieves 100% coverage, incorporating test action executions and assertions for validating expected behaviours. While traditional metrics like code coverage offer quantitative insights, they provide incomplete assessments of Unity apps due to their implementation nature. Unity apps are developed with logical source code and scene configurations (e.g., structures, object properties) that define runtime behaviours.

Focusing solely on code coverage neglects configuration-dependent functionality, potentially misrepresenting actual test case quality. Alternative approaches like mutating testing show promise, having been successfully applied to GUI apps where both source code and interface definitions can be mutated [26]. For Unity apps, serialised scene files [27] offer ideal mutation targets. Future research should therefore explore mutation testing methodologies specifically designed for XR environments, capable of evaluating test case quality.

B. Extensibility

INTENXION's extensibility supports both custom interactions and assertions (§ IV-B) beyond the current focus on 3D interactions, including scenarios like XR locomotion. However, effectively leveraging this extensibility requires testers to possess a substantial understanding of the scenes and objects under test. This foundational domain knowledge is essential for designing meaningful tests and implementing tailored interactions and validators for specialised functionality.

Given the practical focus of our contribution, the current framework implementation and test case design represent an initial step rather than definitive best practices. Future empirical studies with industry practitioners will be crucial to evaluate broader applicability and refine INTENXION's extensibility for industrial settings.

VII. CONCLUSION AND FUTURE WORK

This paper presents INTENXION, a test automation framework designed to address the challenges of validating user interactions in XR apps. We establish a taxonomy of XR interaction derived from interaction designs and industrial practices, which informs INTENXION's design. We conduct a case study to demonstrate INTENXION's practical utility using eight representative interactable object types from established XR design guidelines. The results show that test cases developed with INTENXION successfully cover all these object types, incorporating dedicated action sequences and assertions to validate intended behaviours.

While INTENXION currently focuses on *test automation*, we aim to extend it toward *automated testing*, enabling automatic generation of test cases and action sequences. We also plan to extend platform support beyond Unity to enable broader XR ecosystem compatibility. Additionally, we plan to assess INTENXION's industrial applicability through empirical studies with practitioners in real-world XR development settings.

REFERENCES

[1] S. A. Andrade, A. J. U. Quevedo, F. L. S. Nunes, and M. E. Delamaro, "Understanding VR Software Testing Needs from Stakeholders' Points of View," in *22nd Symposium on Virtual and Augmented Reality (SVR)*. IEEE, 2020, pp. 57–66.

[2] X. Yang, Y. Wang, T. Rafi, D. Liu, X. Wang, and X. Zhang, "Towards automatic oracle prediction for ar testing: Assessing virtual object placement quality under real-world scenes," in *Proc. of the 33rd ACM SIGSOFT*

Intl. Symposium on Software Testing and Analysis, ser. ISSTA 2024. ACM, 2024, p. 717–729.

[3] S. Li, C. Gao, J. Zhang, Y. Zhang, Y. Liu, J. Gu, Y. Peng, and M. R. Lyu, "Less cybersickness, please: Demystifying and detecting stereoscopic visual inconsistencies in virtual reality apps," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, 2024.

[4] R. Gu, J. M. Rojas, and D. Shin, "Software testing for extended reality applications: A systematic mapping study," *Automated Software Engineering*, 2025.

[5] R. Doerner, W. Broll, P. Grimm, and B. Jung, Eds., *Virtual and Augmented Reality (VR/AR): Foundations and Methods of Extended Realities (XR)*. Springer, 2022.

[6] Google, "UIAutomator," https://developer.android.com/training/testing/other-components/ui-automator, accessed: 2025-07-24.

[7] Appium, "Appium," https://appium.io/, accessed: 2025-07-24.

[8] Unity Technologies. XR Interaction Toolkit Package. https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.1. Accessed: 2025-07-24.

[9] X. Wang, "VRTest: An Extensible Framework for Automatic Testing of Virtual Reality Scenes," in *IEEE/ACM 44th Intl. Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2022, pp. 232–236.

[10] X. Wang, T. Rafi, and N. Meng, "VRGuide: Efficient Testing of Virtual Reality Scenes via Dynamic Cut Coverage," in *38th IEEE/ACM Intl. Conference on Automated Software Engineering (ASE)*, 2023, pp. 951–962.

[11] R. Gu and J. M. Rojas, "An empirical study on the adoption of scripted gui testing for android apps," in *2023 38th IEEE/ACM Intl. Conference on Automated Software Engineering Workshops (ASEW)*, 2023, pp. 179–182.

[12] T. Figueira and A. Gil, ""Youkai: A Cross-Platform Framework for Testing VR/AR Apps," in *HCI – Late Breaking Papers: Interacting with eXtended Reality and Artificial Intelligence*. Springer-Verlag, 2022, pp. 3–12.

[13] B. Spittle, M. Frutos-Pascual, C. Creed, and I. Williams, "A review of interaction techniques for immersive environments," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 9, pp. 3900–3921, 2023.

[14] Meta, "Object Interaction, Part 1: Common Interaction Patterns," https://developers.meta.com/horizon/blog/object-interaction-part-1-common-interaction-patterns/, 2017, accessed: 2025-07-24.

[15] ——, "Object Interaction, Part 2: Locomotion and Held Objects," https://developers.meta.com/horizon/blog/object-interaction-part-2-locomotion-and-held-objects/, 2017, accessed: 2025-07-24.

[16] ——, "Object Interaction, Part 3: Releasing Objects," https://developers.meta.com/horizon/blog/object-interaction-part-3-releasing-objects/, 2017, accessed: 2025-07-24.

[17] ——, "Object Interaction, Part 4: Constrained Interactions," https://developers.meta.com/horizon/blog/-object-interaction-part-4-constrained-interactions/, 2017, accessed: 2025-07-24.

[18] Unity Technologies, "Unity Test Framework," https://docs.unity3d.com/Packages/com.unity.test-framework@2.0/, accessed: 2025-07-24.

[19] ——. XR Interaction Simulator Overview. https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.1/manual/xr-interaction-simulator-overview.html. Accessed: 2025-07-24.

[20] Appium, "Actions," https://appium.github.io/appium.io/docs/en/commands/interactions/actions/, accessed: 2025-07-24.

[21] F. Molina, A. Gorla, and M. d'Amorim, "Test Oracle Automation in the Era of LLMs," *ACM Trans. Softw. Eng. Methodol.*, 2025.

[22] I. Arcuschin, L. Di Meo, M. Auer, J. P. Galeotti, and G. Fraser, "Brewing up reliability: Espresso test generation for android apps," in *IEEE Conference on Software Testing, Verification and Validation (ICST)*, 2024.

[23] Unity Technologies, "Interaction Events," https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.1/manual/interactable-events.html, accessed: 2025-07-24.

[24] ——, "VR Template Quick Start Guide," https://docs.unity3d.com/Packages/com.unity.template.vr@9.0/, accessed: 2025-07-24.

[25] ——, "XR Interaction Toolkit Examples," https://github.com/Unity-Technologies/XR-Interaction-Toolkit-Examples, accessed: 2025-07-24.

[26] L. Deng, J. Offutt, P. Ammann, and N. Mirzaei, "Mutation operators for testing android apps," *Information and Software Technology*, vol. 81, pp. 154–168, 2017.

[27] Unity Technologies, "Text-based Scene Files," https://docs.unity3d.com/6000.2/Documentation/Manual/TextSceneFormat.html, accessed: 2025-07-24.