



University of
Sheffield

Automated Testing of User Interactions in Extended Reality Applications

Ruizhen Gu

Supervisors: Dr José Miguel Rojas *and* Dr Donghwan Shin

*A dissertation submitted in candidature for the degree of
Doctor of Philosophy*

in the

School of Computer Science
The University of Sheffield

October 2025

“ I’m a scientist! Now we’re getting somewhere!
Time for me to use science.
All right, genius brain: come up with something!
...I’m hungry. You have failed me, brain.”

— Andy Weir, *Project Hail Mary*

Abstract

Extended Reality (XR) is a rapidly emerging technology that offers immersive user experiences across diverse application domains. Unlike traditional two-dimensional graphical user interfaces, XR features interactions with six degrees of freedom, introducing new paradigms for user interaction and posing unique software engineering challenges. Among these, validating XR software through effective testing remains a crucial open problem.

Current research in XR software testing has produced approaches that support scene navigation and basic interaction activation, yet significant gaps persist. Current methods cannot automatically generate or execute realistic spatial user inputs, such as grab and trigger actions via hand-held controllers or hand gestures. Furthermore, present metrics do not robustly capture the diverse coverage of these spatial interactions. Moreover, the field still lacks a systematic understanding of key testing methodologies and concerns specific to XR environments.

This thesis addresses these challenges with two main objectives: (1) to investigate the landscape of XR software testing and establish foundational knowledge for XR-specific user interaction testing, and (2) to develop and evaluate new techniques that systematically generate and validate realistic user interactions in XR environments. To achieve these goals, five major contributions are presented: (1) the first systematic mapping study of XR software testing, based on 34 published papers; (2) a novel test automation library designed around a new taxonomy of XR user interactions; (3) an automated technique for generating and executing XR user interactions, informed by a categorisation from open-source XR projects; (4) a benchmark dataset of ten XR scenes and 456 user interactions to support standardised evaluation; and (5) a foundational roadmap for the future of XR devices, software, and testing, outlining structured research challenges and mitigation strategies. Collectively, these contributions advance systematic and scalable testing for XR applications, supporting both future research and practical development in this fast-evolving field.

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Dr José Miguel Rojas and Dr Donghwan Shin, for their constant support, insightful guidance, and encouragement throughout my PhD journey. I have been fortunate to learn from them, not only through their academic expertise but also through their kindness, patience, and integrity. They have been more than supervisors to me, as their mentorship and friendship have been invaluable both professionally and personally.

I would also like to thank my colleagues in the AI $\times$ Software Engineering and Testing (ASET) group at the University of Sheffield: Firhard, Neil, Islam, Owain, Megan, Michael, Richard, Olek, Zálán, Giulia, Guannan, Arwa, Tony, Kirill, Sanjeetha, and Phil. Thank you for fostering such a friendly and supportive research environment. Working alongside you has been an unforgettable experience, and I will always cherish the memories we created together, from our Christmas gatherings, walks in the Peak District, and even running the Sheffield 10K as a department. These experiences have made my PhD life not only meaningful but truly enjoyable.

My heartfelt thanks go to my partner, Jane, for her support and understanding. Having completed a PhD herself, she has been my source of motivation, helping me navigate the challenges and ups and downs of this journey. I am also deeply grateful to all my family members, especially my parents, aunt and uncle, and grandparents, for their unconditional encouragement and belief in me from the very beginning of my studies through to the completion of this thesis.

Contents

1	Introduction	1
1.1	Aims and Objectives	3
1.1.1	Understanding	4
1.1.2	Mitigating	4
1.1.3	Envisioning	5
1.2	Publications and Software	5
1.3	Thesis Outline and Contributions	6
2	Background	9
2.1	Extended Reality	9
2.1.1	Augmented Reality	9
2.1.2	Mixed Reality	10
2.1.3	Virtual Reality	11
2.2	Extended Reality Applications	11
2.2.1	XR Application Development with Unity	12
2.2.2	Interaction with XR Applications	15
2.3	Software Testing	16
2.3.1	Test Adequacy	17
2.3.2	Test Automation	17
2.3.3	Automated Testing	18
2.3.4	Model-based Testing	18
2.4	Testing Graphical User Interface	19
2.4.1	Testing XR Applications	19
2.4.2	Testing Other GUI Applications	20
3	A Systematic Mapping Study of XR Software Testing	22
3.1	Motivation of the Mapping Study	23
3.1.1	Usability of XR Applications	23
3.1.2	Motivation	24
3.2	Mapping Study Methodology	25
3.2.1	Planning the Mapping	25
3.2.2	Conducting the Mapping	29
3.2.3	Reporting the Mapping	31
3.2.4	Threats to Validity	31
3.3	Results	33
3.3.1	RQ1: Research Status	33
3.3.2	RQ2: Testing Facets	42
3.3.3	RQ3: Evaluation	48
3.4	Discussion	49
3.4.1	Mapping Study Methodology	49
3.4.2	Datasets and Tools	50

3.4.3	Implications for Practitioners	54
3.4.4	Challenges and Future Research Directions	54
3.5	Answer to ORQ1	57
3.5.1	How XR Applications are Currently Tested	57
3.5.2	Focuses of Existing Approaches	57
3.5.3	Limitations and Weaknesses	57
3.5.4	Key Research Gaps	58
3.6	Conclusion	58
4	Test Automation for XR User Interactions	59
4.1	Motivation	60
4.1.1	XR Scene Testing	60
4.1.2	From UI Events to Spatial 3D User Interactions	61
4.2	Taxonomy of XR User Interactions	61
4.2.1	Core Task Categories	62
4.2.2	Atomic Actions	62
4.2.3	Compositional Patterns	62
4.3	INTENXION Test Library	64
4.3.1	Testable Interaction Mapping	64
4.3.2	Test Library Implementation	64
4.4	Case Study	67
4.4.1	Results	68
4.5	Discussion	68
4.5.1	Test Case Quality	69
4.5.2	Extensibility	70
4.6	Answers to ORQ2 and ORQ3	71
4.6.1	ORQ2: XR Interaction Synthesis	71
4.6.2	ORQ3: Effectiveness of Synthesised XR Interaction	71
4.7	Conclusion	72
5	Automated Testing for XR User Interactions	73
5.1	Prevalence of XR Interactions	75
5.1.1	Data Collection	75
5.1.2	Results	76
5.1.3	Discussions	77
5.2	Interaction Flow Graph	78
5.2.1	Interaction Prioritisation	80
5.2.2	Interaction Dependency	81
5.2.3	Interaction Brokerage	82
5.3	Automated XR User Interaction Explorer	82
5.3.1	Scene Parser	83
5.3.2	Dynamic Explorer	83
5.3.3	Test Result Checker	85
5.3.4	Interaction Flow Coverage	88
5.4	Evaluation	88
5.4.1	Evaluation Setup	89
5.4.2	RQ1: Coverage	90
5.4.3	RQ2: Efficiency	91
5.4.4	RQ3: Bug Detection	92
5.5	Answers to ORQ4, ORQ5 and ORQ6	93
5.5.1	ORQ4: Prevalent Interaction Types	93
5.5.2	ORQ5: XR Interaction Modelling	94
5.5.3	ORQ6: Design and Evaluation of Automated XR Testing	95

5.6	Discussion	96
5.6.1	Interaction Design Smells	96
5.6.2	Oracle Automation	96
5.6.3	Environment-Aware Testing	97
5.7	Conclusion	98
6	Envisioning the Future of XR Device Software and Testing	99
6.1	Conceptual Framework	100
6.1.1	Perception–processing–rendering Cycle	101
6.1.2	Human-in-the-loop Feedback and Collaboration	102
6.1.3	Implications for Software and Testing	102
6.2	Advanced Capabilities	102
6.2.1	Spatial Intelligence	103
6.2.2	Multimodal Interfaces	104
6.2.3	Distributed Software Architecture	105
6.3	Research Problems	106
6.3.1	Security and Privacy	107
6.3.2	Validation of Spatial Capabilities	108
6.3.3	Explainability	109
6.4	Broader Challenges	110
6.4.1	Ethics Governance	111
6.4.2	Accessibility and Inclusivity	111
6.4.3	Open Development Ecosystems	112
6.5	Answer to ORQ7	112
6.5.1	XR Software Evolution	112
6.5.2	New Challenges	113
6.5.3	Role of Software Testing	113
6.6	Conclusion	113
7	Conclusions and Future Work	115
7.1	Limitations	116
7.1.1	Generalisability	116
7.1.2	Limited Subjects	118
7.2	Future Work	118
7.2.1	Agent-based Testing	118
7.2.2	Testing XR-specific Features	120
7.2.3	Specialised XR Simulation Platforms	120
7.2.4	Investigating Testing Practices Among XR Software Developers	121
7.3	Final Remarks	121
A	Primary Studies of the Systematic Mapping Study	123
A.1	List of Primary Studies	123
A.2	Publication Venues of Primary Studies	125
B	Applying XRINTTEST for Test Automation	127
B.1	Requirement Checking and Installation	127
B.1.1	Verify Unity Installation	127
B.1.2	Add Required Packages and Scripts	127
B.2	Scene Configuration	128
B.2.1	Open the Scene Under Test	128
B.2.2	Configure the Scene	128
B.3	Tool Execution	128
B.3.1	Interaction Flow Graph Construction	128
B.3.2	Execute XRINTTEST	128

C	Applying INTENXION for Test Automation	130
C.1	Requirements	130
C.2	Project Setup	130
	C.2.1 Integrating INTENXION into an Existing Project	130
C.3	Test Assembly Configuration	131
C.4	Writing Interaction Tests with INTENXION	131
C.5	Running Tests	132

Figures

1.1	Overview of how the thesis objectives, chapters, and overarching research questions (ORQs) relate to one another	4
2.1	Reality-Virtuality Continuum	10
2.2	Examples of AR and VR Scenes	10
2.3	Examples of Scene Graphs	13
2.4	Meta Quest Touch Pro Controller (Right)	14
3.1	Process of the mapping study	25
3.2	Literature search and selection process. The numbers on the arrows indicate the number of studies provided to the next stage.	29
3.3	Publication Years of Primary Studies	34
3.4	Distribution of publication venue types for the primary studies	34
3.5	Distribution of publication venue domains for the primary studies	35
3.6	Bubble plot for research topics against research type	41
3.7	Immersive Technologies of Primary Studies	41
3.8	Word cloud based on the test activities of the primary studies	43
3.9	Bubble chart of test objectives against test targets	44
4.1	Overview of the XR Interaction Taxonomy	63
4.2	Overview of INTENXION	65
4.3	A test case developed with INTENXION: Navigating to an object, grab holding, rotating, triggering, and releasing (action chains highlighted)	67
4.4	Format of Unity Text Serialised files	70
5.1	Permutations of Socket Interactions (an example from Scene 10)	78
5.2	Hierarchical Structure and Interaction Flows in VR Scene Modelling (Actions: $trigger_{inst}/grab_{inst}$ denote instant actions; $trigger_{const}/grab_{const}$ denote continuous actions; mov_{fwd} indicates forward controller movement; rot_{rt} indicate rightward controller rotation.)	81
5.3	XR User Interaction Testing with XRINTTEST	82
5.4	Examples of direct <i>grab</i> and compositional <i>grab-and-fire</i> interactions	86
5.5	Examples of Compositional Interaction Sequences	86
5.6	Intersection of Colliders of the Target Interactable (Green Bounds) and Controller (Red Bounds)	87
5.7	Efficiency of XRINTTEST in Achieving VR User Interaction Coverage	92
6.1	A conceptual overview of XR glasses systems	101
B.1	Scene configuration and IFG generation are implemented as editor windows in Unity Editor	129
B.2	Enter Caption	129
C.1	Test Assembly	131

C.2	Execute test cases	132
-----	------------------------------	-----

Tables

3.1	PICOC criteria applied to this study	27
3.2	Synonyms in $\$XR$, $\$XR_{acr}$, $\$T$, and $\$B$	27
3.3	Data Extraction Form	32
3.4	Research topics in the primary studies	36
3.5	Research types in the primary studies	39
3.6	Test objectives in the primary studies	44
3.7	Test levels in the primary studies	45
3.8	Test types in the primary studies	46
3.9	Test techniques in the primary studies	46
3.10	Evaluation metrics in the primary studies	47
3.11	Evaluation environments in the primary studies	48
3.12	Training datasets used for machine learning-based testing approaches	50
3.13	Industrial tools in the primary studies. OSS indicates if the tool is open-source or not. .	52
3.14	Research tools in the primary studies	53
4.1	Interactions and test actions required for XR interaction testing tasks	65
4.2	Test APIs Provided by INTENXION	66
5.1	Prevalence of 3D Interactions in VR Projects (Manip.: Manipulate)	76
5.2	Effectiveness of XRINTTEST on Interaction Flow Coverage	90
A.1		123
A.2		125

Acronyms and Abbreviations

<i>IFC</i>	Interaction Flow Coverage
6DoF	Six Degrees of Freedom
AI	Artificial Intelligence
AR	Augmented Reality
AST	Abstract Syntax Tree
CI/CD	Continuous Integration/Continuous Delivery or Deployment
DoF	Degrees of Freedom
DSL	Domain-Specific Language
EFSM	Extended Finite State Machine
EMG	Electromyography
FM	Foundation Model
FOV	Field of View
FPS	first-person shooter
FPS	Frames Per Second
FSM	Finite State Machines
GDPR	General Data Protection Regulation
GO	GameObject
GPS	Global Positioning System
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HMD	Head-mounted Display
HMDs	Head-mounted Displays
IFG	Interaction Flow Graph
IL	Imitation Learning
IMU	Inertial Measurement Unit
IoT	Internet of Things
LTS	Long-Term Support
MBT	Model-based Testing
MLOps	Machine Learning Operations
MR	Mixed Reality

ORQs	Overarching Research Questions
OS	Operating System
PR	Pull Request
PS	Primary Studies
RGB	Red, Green, and Blue
RL	Reinforcement Learning
ScUT	Scene Under Test
SDLC	Software Development Life Cycle
SE	Software Engineering
SUT	System Under Test
UI	User Interface
VR	Virtual Reality
W3C	World Wide Web Consortium
WCAG	Web Content Accessibility Guidelines
XAI	Explainable AI
XR	Extended Reality
XRI	XR Interaction Toolkit

Chapter 1

Introduction

Extended Reality (XR), which encompasses Augmented Reality (AR), Mixed Reality (MR), and Virtual Reality (VR) [1], merges or replaces the real world with digitally generated elements. AR overlays digital information onto physical surroundings, MR integrates virtual content that can interact with the physical environment, and VR creates fully immersive virtual environments. The global XR market was valued at approximately USD 77 billion in 2024 and is forecast to exceed USD 3 trillion by 2037¹. This projected growth reflects the increasing adoption and technological maturation of XR across multiple sectors. Major technology companies are investing heavily in XR, driving industry innovation and expanding application areas.

XR hardware ranges from Head-mounted Displays (HMDs), such as Meta Quest 3², which provide immersive experiences for entertainment and productivity, to display-free, Artificial Intelligence (AI)-enabled *smart glasses*, such as Ray-Ban Meta³. These advancements are fuelling a convergence between spatial computing that is traditionally associated with HMDs and AI features found in smart glasses, resulting in a new paradigm of devices: AI-powered everyday XR glasses, or simply XR glasses, and other lightweight spatial wearables [2, 3]. Many experts and analysts view these devices as potential successors to smartphones, with the capacity to fundamentally reshape human-digital interaction and social communication in everyday life [4].

On the software side, in late 2024, Google announced Android XR, a dedicated XR Operating System (OS) built for next-generation computing experiences⁴. The platform is developed in collaboration with Samsung for the Galaxy XR headset⁵, which might represent a significant shift in the XR landscape, further accelerating mainstream adoption. XR applications, or simply XR apps, are software programs specifically designed to run on XR-compatible devices. These apps typically feature virtually organised spaces populated with virtual objects and interactive elements, allowing users to explore digital scenes and engage with digital content. A notable example is *Pokémon Go*⁶, an AR mobile game that leverages device Global Positioning System (GPS) and cameras to overlay virtual content onto real-world locations. Although mobile AR is an important part of the XR ecosystem, the testing approaches developed

¹Market forecast available at: <https://www.researchnester.com/reports/extended-reality-market/4863>

²<https://www.meta.com/quest/quest-3/>

³<https://www.meta.com/ai-glasses/>

⁴<https://www.android.com/xr/>

⁵<https://www.samsung.com/us/xr/galaxy-xr/galaxy-xr/>

⁶<https://pokemongolive.com>

in this thesis focus on HMD-based applications with rich, tracked spatial interactions. Entertainment, particularly video games, is a prominent application domain for XR technologies [5], while fields including education [6], engineering [7], military applications [8], and medicine [9] have also benefited from XR development. This wide range underlines XR’s transformative potential beyond consumer entertainment into practical and professional sectors.

As XR apps continue to expand across diverse domains and platforms, their development and testing processes have become increasingly complex, exposed to the broader **Software Engineering (SE)** research community and underscoring the need for comprehensive testing approaches [10]. The testing of XR apps encompasses concerns beyond functionality and usability, including challenges such as virtual object placement accuracy [11] and cybersickness detection [12]. Among these, *functionality* and *user interaction* remain the most critical aspects in ensuring software quality [13]. User interaction with XR refers to the engagement with the **User Interface (UI)** of XR software or systems, designed to simulate real-world interactions within immersive environments. Unlike traditional 2D **Graphical User Interfaces (GUIs)** found in mobile apps, which primarily rely on widget-based interactions, XR interactions are characterised by real-time responsiveness and complex, spatially rich actions enabling users to *select* and *manipulate* virtual objects or *navigate* through virtual environments [14].

User interaction is particularly essential for GUI testing, which fundamentally involves simulating user actions to interact with the **System Under Test (SUT)** and to verify appropriate interface responses. This complements unit testing by focusing on interaction sequencing and ensuring the software meets requirements and quality standards. In typical 2D GUI apps such as Android apps, GUI testing treats the apps as a black box, interacting directly with the widgets to validate functionality [15]. These kinds of apps can often be modelled as a **Finite State Machines (FSM)** [16], where input events required to reach a specific state are finite and reproducible, thus facilitating controlled and predictable testing. In addition, 3D video games, which share common foundations (e.g., development platforms and certain technology stacks) with XR apps [17], can also be modelled as FSMs for testing purposes.

In contrast to conventional 2D GUIs, XR apps commonly support spatial inputs tracked with six **Degrees of Freedom (DoF)** (the number of independent ways an object can move or rotate in three-dimensional space). Although some non-immersive 3D video games also support complex spatial movement, XR interaction directly couples users’ tracked head, hand, or controller movements with actions in the virtual environment. This distinguishes XR apps from traditional 2D GUIs in both interface presentation and user interaction. Whereas 2D GUIs constrain input to a flat screen and a small, well-defined set of actions such as clicking, tapping, or scrolling, XR apps allow users to move, rotate, and reach in 3D space, which creates effectively unbounded interaction variations. For example, in a 2D media player, pressing “Play” is typically a single, uniform event (a click or tap on a button), while in XR the same intent might be realised by walking around a virtual console, pointing a controller ray, performing a mid-air pinch gesture, each with distinct timing, occlusion, and tracking conditions that introduce substantial unpredictability into testing. Moreover, real-world context further differentiates XR modalities: VR immerses users in a fully synthetic environment with no direct dependence on physical surroundings, whereas AR and MR explicitly anchor content to real-world surroundings, adding another layer of variability beyond what is encountered in conventional 2D GUI testing. Although users may experience similar 3D environments with XR apps and non-immersive video games, their interaction paradigms and real-world integration significantly diverge, leading to unique user interaction patterns and distinct testing challenges. For example, a conventional video game may map object manipulation

to predefined button presses and analogue-stick movements. In XR, the same task may require the user to physically approach an object, align a tracked controller, maintain a grab, and move through three-dimensional space.

Consequently, successful XR user interaction testing requires not only the generation and validation of realistic 3D user actions, but also the exploration of complex spatial scenarios [13]. However, the emerging nature of XR hardware and software leads to the fact that overall user interaction practices are not yet well-established in the software engineering and testing research communities, presenting a fundamental challenge to developing advanced testing techniques. These differences between XR and traditional software pose unique challenges for XR software testing. For example, in the context of generating test sequences, Android apps have finite interaction paths when navigating between different *activities* (i.e., individual screens of an app) [16]. In contrast, XR apps involve virtually infinite interaction possibilities; even a simple task, such as moving towards and interacting with a virtual object, requires accounting for countless variations in interaction sequences [18]. These complexities necessitate advanced software testing methods that can cope with the unpredictable and immersive nature of XR, ensuring reliable operation and meeting user expectations.

Moreover, unlike traditional software, which benefits from well-established surveys covering various testing practices [19, 20], to the best of the author’s knowledge, no comprehensive systematic reviews dedicated to XR software testing were available when this study was conducted. Critical aspects, such as testable user interaction categories, testing practices, tools, frameworks, and general testing guidelines, remain largely unexplored. This gap motivates fundamental questions: *What foundational knowledge currently exists for XR software testing and related areas?* Moreover, *how can this foundation be leveraged to develop novel, effective XR testing techniques?*

1.1 Aims and Objectives

The overarching aim of this thesis is to investigate the emerging field of software testing for XR applications and to advance both fundamental understanding and practical techniques needed for effective testing. Central to this aim is the following high-level research question:

How can XR applications be systematically and automatically tested to improve test performance and more reliably uncover bugs?

To answer this question, the thesis is organised around three interrelated objectives:

- **Understanding:** To characterise the current landscape of XR software testing, including its main focuses, limitations, and research gaps in the existing literature.
- **Mitigating:** To design, implement, and empirically evaluate novel techniques for systematically and sufficiently testing XR applications, with an emphasis on functionality and user interactions.
- **Envisioning:** To reflect on the future evolution of XR software and articulate the prospective roles, challenges, and opportunities for XR software testing in that context.

For each objective, a set of **Overarching Research Questions (ORQs)** is defined to guide the thesis, yielding seven ORQs in total. Figure 1.1 summarises how these objectives map onto the thesis chapters:

Chapter 3 addresses the *understanding* objective and ORQ1, Chapters 4 and 5 address the *mitigating* objective and ORQs 2–6, and Chapter 6 addresses the *envisioning* objective and ORQ7. The following sections describe each objective in more detail together with its corresponding ORQs.

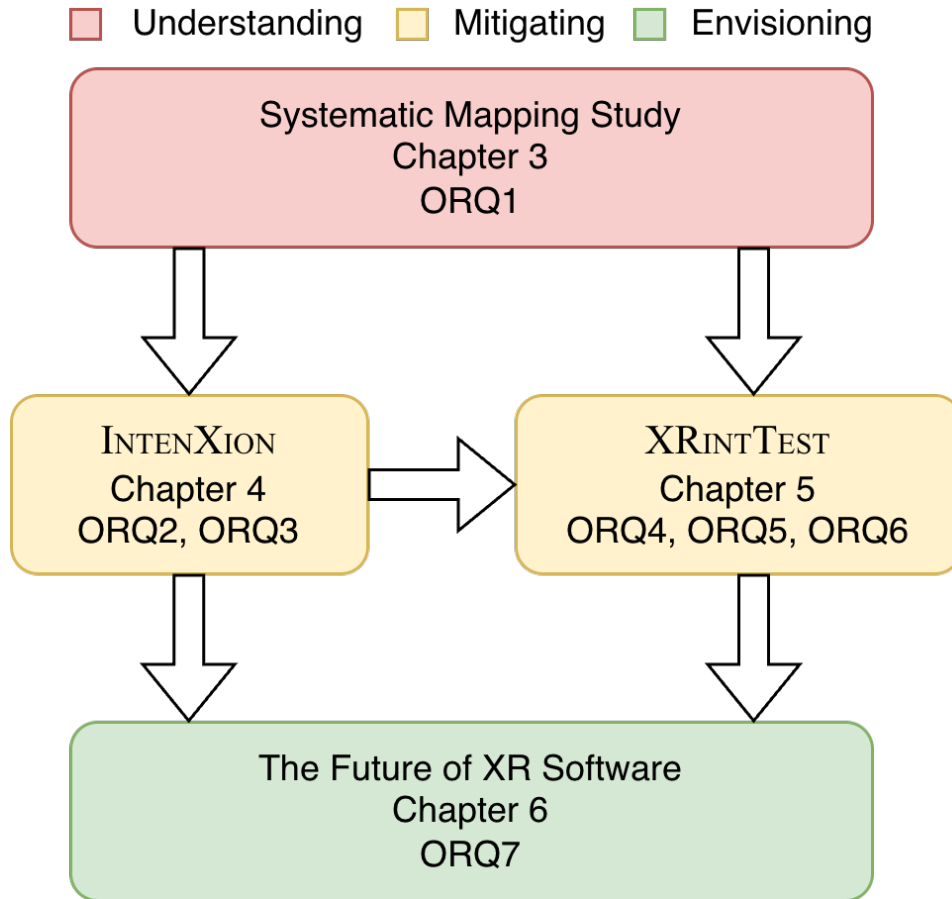


Figure 1.1: Overview of how the thesis objectives, chapters, and overarching research questions (ORQs) relate to one another

1.1.1 Understanding

The first objective is to establish a solid foundational understanding of how XR applications are currently tested. This involves characterising existing approaches, their strengths, and their limitations, as well as identifying gaps that motivate new research directions. The corresponding research question is:

ORQ1 How has XR testing been approached in existing research, and what key focuses, limitations, and research gaps have been identified?

1.1.2 Mitigating

Building on this foundation, the second objective shifts from understanding to intervention. Here, the thesis aims to design, implement, and empirically evaluate new techniques that address the identified

gaps, with a particular emphasis on realistic user interactions in XR environments. This objective is pursued through the following research questions:

ORQ2 How can individual XR interactions be systematically decomposed and synthesised as executable test actions?

ORQ3 How effective are the synthesised XR interactions at evaluating XR applications?

ORQ4 What XR interaction types are prevalent in open-source XR projects?

ORQ5 How can interactions among objects in XR scenes be systematically modelled to support automated test generation?

ORQ6 How can automated testing strategies be designed and evaluated to effectively assess XR user interactions?

1.1.3 Envisioning

Finally, the third objective broadens the perspective from current techniques to future trajectories. Informed by the insights and empirical findings of the previous objectives, the thesis reflects on how XR software may evolve and what roles software testing should play in supporting that evolution. This leads to the following research question:

ORQ7 How might XR software evolve in the future, what new challenges are likely to arise, and in what ways can software testing help to address these challenges?

1.2 Publications and Software

Throughout this thesis, the following works have been published in peer-reviewed conferences and journals:

P1 Ruizhen Gu, José Miguel Rojas and Donghwan Shin, “Software Testing for Extended Reality Applications: A Systematic Mapping Study,” *Automated Software Engineering*, 2025. (Journal paper.)

P2 Ruizhen Gu, José Miguel Rojas and Donghwan Shin, “XRINTTEST: An Automated Framework for User Interaction Testing in Extended Reality Applications,” in *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering*, 2025. (Tool Track paper.)

P3 Ruizhen Gu and José Miguel Rojas, “A Test Automation Framework for User Interaction in Extended Reality Applications,” in *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering Workshops*, 2025. (Workshop paper.)

P4 Ruizhen Gu, Jingqiong Zhang, José Miguel Rojas and Donghwan Shin, “On the Promises and Challenges of AI-Powered XR Glasses as Embodied Software,” in *Proceedings of the 2nd ACM/IEEE International Conference on AI-powered Software*, 2025. (Conference paper.)

P5 Ruizhen Gu, José Miguel Rojas and Donghwan Shin, “Automated Testing of Prevalent 3D User Interactions in Virtual Reality Applications,” *Automated Software Engineering*, 2026. (Journal paper.)

In addition, this thesis resulted in the development and release of several open-source software artefacts and datasets:

S1 INTENXION

GitHub Repository: <https://github.com/ruizhengu/IntenXion>

A test automation library for validating user interactions within Unity XR applications. The library enables the creation of intuitive test scripts, supporting both action sequences and assertions on expected behaviours.

S2 XRINTTEST

GitHub Repository: <https://github.com/ruizhengu/XRintTest>

An automated testing tool for Unity XR applications. It constructs an abstraction of XR scenes, modelling interaction targets and required action sequences, and leverages this abstraction to automatically explore the scene and synthesise user interactions.

S3 XRBENCH3D

GitHub Repository: <https://github.com/ruizhengu/XRBench3D>

A benchmark dataset for XR user interaction testing, consisting of ten VR scenes and 456 distinct user interactions for evaluating VR testing approaches. The dataset covers four prevalent interaction types: *fire*, *manipulate*, *socket*, and *custom*.

1.3 Thesis Outline and Contributions

The remainder of this thesis is structured as follows.

Chapter 2 introduces the fundamentals of XR, general software testing, and current research efforts on GUI software testing, establishing the context for unique challenges in the XR domain. It provides the necessary background and clarifies why XR testing demands distinct attention from conventional techniques.

Chapter 3 represents a systematic mapping study of existing literature on XR software testing. The primary contribution of this chapter is the first comprehensive review of the field, identifying research trends, current methodologies, and gaps in knowledge. This sets the stage for identifying unresolved challenges and research needs in XR software testing. Key contributions of this chapter include:

- An in-depth survey of current software testing methods for XR applications, providing insights into the state-of-the-art techniques utilised in this emerging domain. A data extraction template employed in this study is made publicly available to advance this domain.
- A curated repository of existing tools and datasets relevant to XR software testing, designed to support and facilitate future research efforts. This repository, alongside the data extraction results, is publicly accessible at: sites.google.com/view/xr-testing.
- A detailed identification of critical challenges facing XR software testing, coupled with proposed research directions aimed at addressing these challenges.

Chapter 4 progresses by introducing the first test automation library for XR apps. The chapter begins by proposing a novel taxonomy of XR user interactions, which enables the formation of testable XR action sequences, thereby informing effective and reproducible testing strategies. It brings new insights into XR interaction formalisation and discusses the implications for synthesising robust, testable actions. Key contributions include:

- **XR Interaction Taxonomy:** Developed through synthesis of [Human-Computer Interaction \(HCI\)](#) research and industrial practices, the taxonomy captures 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 guiding testing methodologies.
- **Test Automation Library:** Leveraging the XR interaction taxonomy, we designed INTENXION to support a wide range of XR interaction types. This library facilitates the creation of intuitive interaction test scripts, encompassing both action sequences and assertions of expected behaviours. Our case study demonstrates that INTENXION achieves comprehensive coverage across all object types defined in the taxonomy.

Building on these findings, Chapter 5 introduces, implements, and evaluates the first automated testing tool specifically designed for realistic XR user interactions. The chapter starts with an empirical analysis of user interactions within open-source XR projects, which leads to the development of an abstraction for XR user interactions. This model allows automatic generation of test data, i.e., user interactions, that effectively exercise XR scene functionalities, demonstrating practical approaches to automating the uniquely complex behaviours and scenarios found in XR apps. The model and the approach are represented as significant foundational and technical contributions to the field. This chapter makes the following contributions:

- **XR Interaction Categorisation:** We present the first categorisation of 3D interactions through the analysis of open-source XR projects. This offers empirical grounding for the taxonomy and enhances understanding of prevalent interaction types in XR apps.
- **Interaction Model for XR Testing:** We propose a novel, systematic abstraction that formally represents diverse interactions within XR environments. This interaction model serves as the foundational framework guiding the development of effective XR testing strategies.
- **Dataset:** We construct XRBENCH3D, a novel benchmark comprising ten XR scenes with 456 3D interactions. It facilitates standardised evaluation of XR testing approaches, addressing the prior absence of dedicated resources.
- **Automated Testing Tool:** We develop XRINTTEST, a model-based testing technique that leverages the interaction model to explore XR scenes and activate interactions automatically. XRINTTEST achieves 93% interaction coverage on XRBENCH3D, outperforming a random-action testing baseline by approximately a factor of 12 in effectiveness and six in efficiency.

Chapter 6 adopts a forward-looking perspective, outlining a foundational roadmap to guide and inspire future research in XR software testing. This chapter specifically focuses on XR glasses, an emerging category of XR devices, and examines their potential to establish a distinct and transformative

software paradigm. To support the development of robust immersive software, this chapter presents a structured vision of research challenges and potential mitigation strategies for XR software testing. It provides the following interdisciplinary contributions:

- Establishes a conceptual overview of XR glasses as a new class of software systems.
- Maps their computational and interaction capabilities within a unified context bridging AI and SE.
- Identifies potential interdisciplinary research problems at the intersection of AI and SE.
- Highlights potential societal and ethical implications associated with XR glasses.

Finally, in Chapter 7, we conclude the thesis by reflecting on the research objectives and addressing the overarching research questions. Drawing on the key findings, we outline prospective directions for future work within the evolving domain of software engineering and testing for extended reality.

Chapter 2

Background

This chapter provides background on extended reality, covering key concepts and terminology across various immersive technologies that form the foundation of this mapping study. We introduce the nature of XR applications and their user interaction models, followed by relevant software testing focusing on automated testing, test automation, and GUI testing approaches.

2.1 Extended Reality

XR is an umbrella term for all immersive technologies, including AR, MR, and VR. Over recent years, advancements in these technologies have allowed users to immersively interact with virtual objects and virtual environments with specific devices, such as HMDs and controllers. XR capabilities vary by technology and device. HMD-based VR and MR systems commonly employ stereoscopic displays, head and motion tracking, and dedicated controllers or hand-tracking sensors. Mobile AR devices typically provide a more limited experience through a handheld screen, camera, touchscreen, and device-motion sensors. These differences affect the level of immersion and the interaction techniques available to applications.

Figure 2.1 illustrates the differences between XR technologies using the reality-virtuality continuum introduced by Milgram et al. [1], which is widely accepted in the research community. The continuum spans from fully real environments (reality, on the left) to entirely virtual ones (virtual reality, on the right). The proportion of real versus virtual elements shifts along the continuum: reality diminishes while virtuality increases, and vice versa. Across this continuum, AR primarily overlays digital content onto a view of the physical world, while MR more deeply integrates virtual content by accounting for the geometry and behaviour of the physical environment. VR replaces the user’s view of the physical environment with a fully virtual one. However, the boundaries between these categories are not universally agreed upon, and the terms AR and MR are sometimes used interchangeably.

2.1.1 Augmented Reality

Augmented reality (AR) is a technology that overlays digital content onto the real-world visual environment, enhancing the user’s perception of reality. Positioned near the reality end of the continuum, AR superimposes virtual objects onto the physical world in real time. A prominent example is the

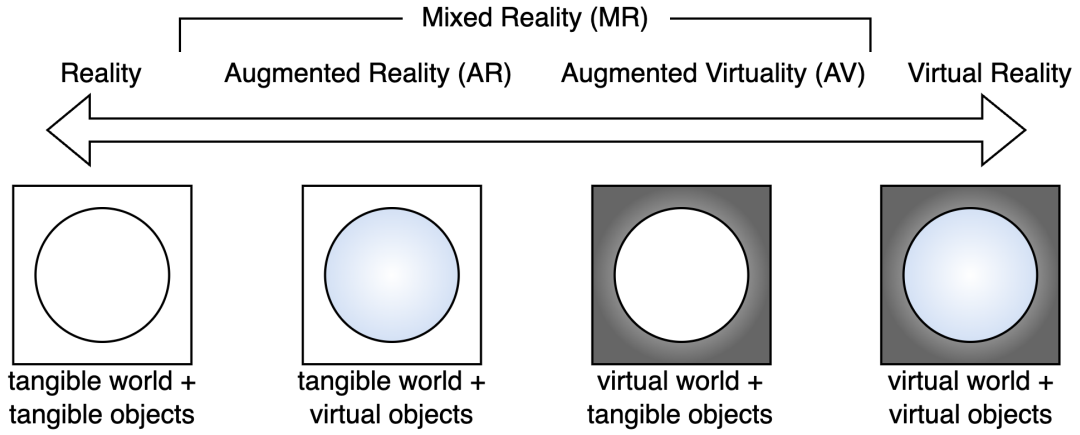
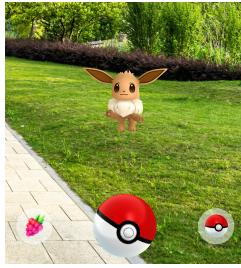


Figure 2.1: Reality-Virtuality Continuum



(a) AR Example



(b) MR Example



(c) VR Example

Figure 2.2: Examples of AR and VR Scenes

mobile game *Pokémon Go* (Figure 2.2a), which overlays virtual creatures and interactive widgets onto a camera view of the physical environment. Unlike more deeply integrated MR content, these virtual elements do not necessarily respond to real-world geometry, occlusion, or lighting.

2.1.2 Mixed Reality

Mixed reality (MR) represents a more immersive blend of real and virtual worlds, enabling physical and digital objects to interact dynamically. It bridges AR and VR by blending real and virtual environments, enabling real-time interaction between physical and digital elements that behave as if they co-exist. For instance, car designers can use MR to manipulate 3D models of car components, refining designs with seamless interaction between real and virtual elements (Figure 2.2b¹). Unlike AR's focus on overlaying information, MR provides deeper integration and interaction, enabling virtual objects to appear spatially integrated with the physical environment and to respond to its geometry and objects.

The distinction between AR and MR remains contested, and both terms are used inconsistently across industrial and academic contexts [14]. For clarity, this thesis uses AR for experiences that primarily overlay digital content onto the physical world, and MR for experiences in which virtual content is spatially integrated with and responds to the physical environment.

¹<https://newsroom.porsche.com/en/2024/innovation/porsche-mixed-reality-workshop-augmented-reality-34998.html>

2.1.3 Virtual Reality

Virtual Reality (VR) immerses users entirely in a computer-generated digital environment, typically through HMDs and motion tracking. Positioned at the virtuality end of the continuum, VR replaces the physical world with a fully simulated, often three-dimensional space in which users can interact. For instance, Figure 2.2c shows *Resident Evil 4 VR Mode*², a VR video game, where players perform actions like shooting and reloading within a fully virtual setting, blocking out the real world completely.

2.2 Extended Reality Applications

The development of most XR apps revolves around versatile 3D engines and software platforms, with Unity³ and Unreal Engine⁴ standing out as industry leaders [21]. These platforms empower developers to design interactive and realistic XR experiences while supporting development across a diverse range of devices and operating systems, including Android, iOS and web browsers [22, 23]. Given the mature integrations with toolkits like ARCore⁵ and ARKit⁶, mobile OSs like Android and iOS have become particularly popular for deploying XR apps. The ubiquity of smartphones and tablets equipped with various sensors and cameras, combined with the accessibility of these development kits, has made XR experiences more mainstream and user-friendly [22].

Complementing mobile-centric experiences, the web is rapidly emerging as a promising platform for XR content [23]. The [World Wide Web Consortium \(W3C\)](#) has released WebXR Device API⁷, enabling developers to create XR apps directly in web browsers without the need for explicit app installations. Frameworks like A-Frame⁸ and Three.js⁹ have further accelerated innovation in web-based XR by providing robust abstractions for 3D scene management and interaction across devices.

As XR technologies and supporting platforms continue to mature, the ongoing convergence of web, mobile, and dedicated hardware environments (e.g., HMDs, XR glasses) is shaping a vibrant and rapidly expanding landscape for XR application development. Moreover, developers now benefit from robust engines, cross-platform standards, and powerful frameworks, which streamline the creation and deployment of immersive experiences across a range of devices and operating systems.

XR applications span mobile, web-based, and dedicated immersive platforms. This thesis focuses on HMD-based applications that support rich spatial interactions through tracked controllers or comparable input modalities. Mobile AR applications remain within the broader XR domain, but their predominantly touchscreen-based interaction model differs from the spatial interaction paradigm investigated in this thesis. Applications developed exclusively for mobile-centric interaction are therefore outside the scope of the proposed testing approaches.

²https://store.playstation.com/en-gb/product/EP0102-PPSA07412_00-RE4RDLC000000028

³<https://unity.com/>

⁴<https://www.unrealengine.com/>

⁵<https://developers.google.com/ar>

⁶<https://developer.apple.com/augmented-reality/arkit/>

⁷<https://www.w3.org/TR/webxr/>, <https://immersiveweb.dev/>

⁸<https://aframe.io/>

⁹<https://threejs.org/>

2.2.1 XR Application Development with Unity

The development environments for these XR apps have matured considerably. For example, Unity³ has emerged as one of the mainstream platforms for XR development, supporting major XR OSs like Meta Horizon¹⁰ and Android XR¹¹. Comprehensive development frameworks like Unity’s **XR Interaction Toolkit (XRI)**¹² provide developers with a high-level, component-based XR interaction system. It is designed for creating diverse XR experiences, including VR and AR. The toolkit standardises common interaction tasks, such as object selection and manipulation, allowing for rapid prototyping and development of engaging XR apps.

To ensure consistency, this thesis emphasises XR app development using Unity-based technology stacks and adopts its relevant definitions and terminologies. For instance, “interactor” and “interactable” are specific terms from XRI, referring respectively to the conceptual input source and target in XR user interactions (see Section 2.2.2). These terms are used consistently throughout this thesis, regardless of the underlying implementation framework.

In Unity’s development paradigm, an XR application is organised through *scenes*, which serve as containers representing discrete parts of an application. A typical XR app consists of interconnected *scenes*, analogous to *activities* in Android apps, each representing a unique virtual environment. Within each scene, the core elements are **GameObjects (GOs)** and their associated *components*. GOs are the graphic entities that users can interact with, while components provide functionalities to these objects, such as animation, physics, or video playback [24]. For instance, adding a *Rigidbody* component to a GO enables physics-based motion simulation. When combined with a *Collider* component, the GO can physically interact with other objects in the environment.

Beyond these built-in components, developers can extend GO capabilities by attaching custom *scripts*, the source code files written in languages like C# for Unity and C++ for Unreal Engine. These scripts define specific behaviours and interaction patterns for GOs, implementing the custom logic necessary for immersive and interactive experiences. Despite being built with various frameworks and languages, they share a common architectural principle. The hierarchical structure of XR scenes, including object relationships and properties (e.g., behaviours, appearances), is managed using specialised data structures called *scene graphs* [25].

Figure 2.3a depicts the scene graph of an XR environment, including objects such as a *Road* and a *Car*. The *Car* object is further divided into sub-objects like *Body*, *Spoiler*, and *Wheels*. Figure 2.3b presents the corresponding XR scene in the Unity Editor, showcasing the hierarchical relationships among these objects. Users of XR apps typically begin at certain positions within the scene, with a camera attached to them, allowing them to view parts of the scene based on their current positions and perspective [26]. They can navigate the scene and adjust their viewpoints to explore its content further.

XR Interaction Toolkit

The XR Interaction Toolkit¹² (XRI) is a high-level framework within Unity for developing XR experiences. It encompasses three core functional domains:

- **3D Interaction:** Facilitates manipulation of GOs through the interaction paradigm based on

¹⁰<https://developers.meta.com/horizon/>

¹¹<https://www.android.com/xr/>

¹²<https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.1>

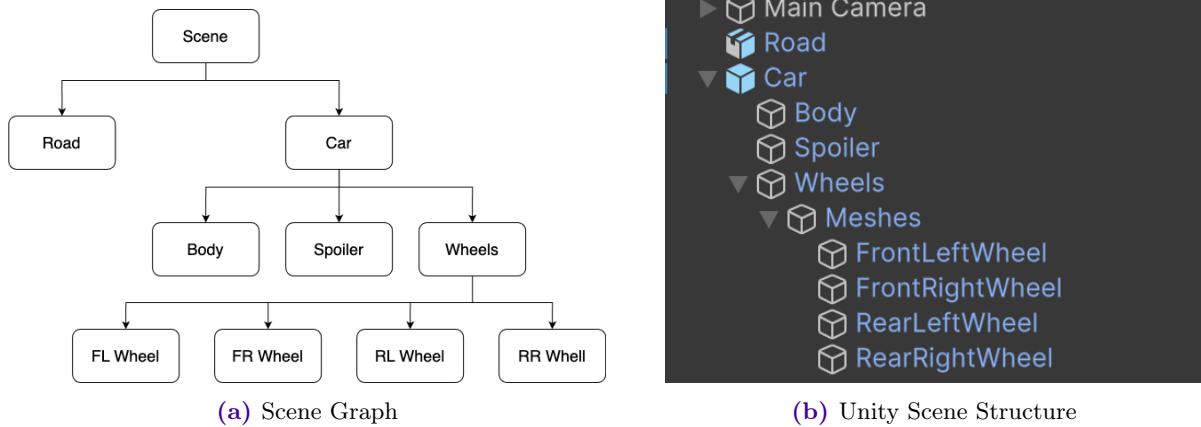


Figure 2.3: Examples of Scene Graphs

interactors and *interactables*;

- **UI Interaction:** Provides components for interaction with UI *controls* (e.g., buttons, sliders, dropdown menus), both in 2D and 3D space;
- **Locomotion:** Enables versatile scene navigation, including teleportation, continuous movement, and advanced mechanics like climbing.

As the core of the modern XR interaction paradigm, the 3D interaction system is built around two fundamental elements:

- **Interactors:** Components that represent user input sources, responsible for handling inputs and initiating interactions with interactable objects.
- **Interactables:** Objects configured to respond to interactors, defining specific behaviours for interactions such as triggering and grabbing.

Consider a scenario where the user is required to press a virtual button: in this case, the interactor corresponds to the user's input modality (e.g., controller, gesture), while the interactable is the button itself, which reacts to the incoming interaction (e.g. pressing) and activates the associated action. This interaction paradigm is closely related to intuitive engagement with virtual objects through physical controllers, such as the Meta Quest Touch Plus controller¹³ displayed in Figure 2.4, to interact with the virtual environments. Controllers like the Quest Touch Plus typically feature two primary buttons for intuitive user interaction: the *trigger* button for actions like firing weapons, and the *grab* button for grasping and manipulating objects.

Interactables and Interactors

In the XRI framework, both interactors and interactables are implemented as scripts attached to GOs. The framework offers a suite of predefined components to facilitate common interaction scenarios, such as the XR Grab Interactable¹⁴ that implements the basic grab event functionality. These predefined

¹³<https://www.meta.com/quest/accessories/quest-touch-plus-controller/>

¹⁴<https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.2/manual/xr-grab-interactable.html>



Figure 2.4: Meta Quest Touch Pro Controller (Right)

components also serve as the foundation to implement customised interaction patterns tailored to specific app requirements.

Conceptually, all XR interactions could adopt the “interactor-interactable” user interaction paradigm, where an XR system relies on user input devices to perform actions on target objects within the environment. Importantly, this interaction paradigm is not limited to specific immersive technologies (e.g., VR or AR) but is central to all the technologies across the reality-virtuality continuum (those in Figure 2.1). A more detailed discussion of how this assumption affects the generalisability of the test approaches proposed in Chapters 4 and 5, including their potential limitations, is presented in Section 7.1.

GameObject Layers and Interaction Layers

When using Unity to develop VR applications, two types of *layers* are relevant to interaction: *GameObject Layers* and *Interaction Layers*.

- **GameObject Layers¹⁵:** Unity’s built-in layer system, used by subsystems like physics, cameras. A key use is configuring which objects can collide. For example, the player avatar and non-player character (NPCs) can be assigned to a “Player” layer so they collide with the environment, while UI elements on a “UI” layer are excluded from physics collisions.
- **Interaction Layer Masks¹⁶:** A layer system used by the XRI framework to decide which *Interactors* (e.g., controllers, sockets) can interact with which *Interactables* (e.g., grabbable objects). Interactors and Interactables must share at least one common interaction layer in their Interaction Layer Mask to allow interactions. For instance, a controller interactor assigned to the “Desk” interaction layer will only interact with interactables that also include the “Desk” interaction layer, and will ignore those configured only with a “Chair” interaction layer. More details regarding how interaction layer masks affect socket interactions are discussed in Section 5.1.3.

Misconfigured layers can cause failures in XR testing. GameObject Layers affect whether interactors can physically detect objects, while Interaction Layers determine whether XR interaction can be properly activated (e.g., grabbed). Automated testing strategies should consider both layer configurations to ensure interaction coverage across different layers.

¹⁵<https://docs.unity3d.com/6000.3/Documentation/ScriptReference/GameObject-layer.html>

¹⁶<https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.1/manual/interaction-layers.html#interaction-layers-settings>

2.2.2 Interaction with XR Applications

Building upon the foundational concepts of interactors and interactables discussed previously, user interaction within XR applications typically encompasses several fundamental tasks. These tasks define how users engage with and manipulate the virtual environment, shaping the overall immersive experience. Specifically, XR interaction usually revolves around three main categories [14, 27]:

- **Navigation:** Controls the user's position and viewing direction within the virtual environment;
- **Selection:** Chooses a point, area, volume, or specific virtual object;
- **Manipulation:** Modifies the parameters of virtual objects, such as their location, orientation, or size.

While these tasks are conceptually similar to those in traditional 2D graphical user interfaces, their execution in XR introduces significantly greater complexity and nuance [14, 28].

This increased complexity arises primarily from the DoF involved, in this context, the number of ways an object can move in space. In traditional 2D interfaces, interactions typically involve rigid bodies with three DoF: two translations (horizontal and vertical) and one rotation. In contrast, 3D objects in XR apps operate with six DoF (*6DoF*), encompassing three translational movements (forward/backward, up/down, left/right) and three rotational movements (yaw, pitch and roll). The additional freedom of movement provided by 6DoF introduces multiple layers of complexity, demanding more sophisticated interaction techniques and testing methodologies to ensure reliable user experience.

Beyond DoF, XR system also have two fundamental functional requirements: *occlusion* and *collision* [14, 29]. Occlusion refers to the phenomenon where virtual or physical objects block the view of one another. It affects how users perceive depth and spatial relationships within the virtual environment, playing a key role in rendering realistic and immersive scenes. Accurate occlusion enhances realism and immersion by ensuring proper visual layering within virtual scenes. Collision, on the other hand, refers to the interaction between objects (virtual or physical) when they come into contact. Proper collision detection ensures that virtual objects behave realistically, such as preventing them from passing through one another or accurately responding when interacting with physical objects [22]. Together, occlusion and collision are essential for crafting believable and immersive XR experiences, presenting challenges that are far less pronounced in conventional 2D interfaces.

XR Interaction Devices

The choice of input device is a critical factor in enabling effective interaction within XR environments. Unlike traditional 2D apps, which rely on menus, buttons, and toolbars, XR apps often require specialised hardware to support their unique interaction paradigms. While mobile and web-based XR apps typically run on conventional devices like smartphones and web browsers, delivering more immersive XR experiences usually demands dedicated devices, such as HMDs. These devices are specifically designed to handle complex and dynamic interactions in XR apps, including 6DoF tracking. This capability enables precise mapping of the user's physical actions, such as movement, rotation, and gestures, into immersive environments, allowing more natural and intuitive interactions.

These devices are designed to manage the complex and dynamic interactions inherent to XR apps. Most HMDs are equipped with hand-held controllers that support six degrees of freedom (6DoF) track-

ing. This functionality enables precise mapping of the user's physical actions, such as movement, rotation, and gestures, within the immersive environments.

XR Action Sequencing

In a virtual environment, user inputs are processed in real time, and small variations can alter scene behaviour or the sequence required to complete a task [18]. These variations may be intentional, such as choosing a different movement path or interaction order, or unintentional, arising from hand tremor, tracking noise, timing differences, or imprecise controller positioning. Consequently, reproducing an exact XR input sequence can be challenging. In contrast, traditional GUI software, such as 2D mobile applications, can often be modelled as finite state machines (FSMs) [16], where input events required to reach a specific state are finite and reproducible. This allows for controlled and predictable interactions.

However, XR apps rely on 6DoF interactions and real-world context, introducing unpredictability. For instance, an XR app may present varying virtual content depending on the user's current location or physical surroundings, making it far more complex to test and replicate specific input sequences compared to traditional 2D GUI apps.

Similarly, although XR apps and 3D video games share common foundations, such as development with the same 3D engines [17], they differ significantly in interaction mechanisms and real-world integration. This systematic mapping study distinguishes XR apps from 3D video games, acknowledging their shared technological roots but unique user experiences and testing challenges.

2.3 Software Testing

Software testing is a systematic and empirical process within the [Software Development Life Cycle \(SDLC\)](#) that aims to assess and improve the quality, reliability, and correctness of a software [System Under Test \(SUT\)](#) [30]. It involves executing the system or its components under controlled conditions to reveal defects, validate functionality, and provide evidence that the software behaves as intended in representative scenarios. Beyond simply detecting faults, testing serves as a critical feedback mechanism that informs design decisions, supports maintenance, and builds confidence in the dependability of complex systems [30].

A central element of software testing is the *test case*, which specifies the conditions for executing the SUT in a certain way. Test cases typically include inputs, execution conditions, and expected results, known as *oracles*, to validate the software's behaviour [31, 32] and detect faults.

Testing spans multiple levels, each with distinct objectives:

- **Unit Testing:** verifies individual components, such as methods or classes, in isolation;
- **Integration Testing:** examines interactions between components, such as method calls across modules;
- **System Testing:** assesses overall behaviour, including non-functional requirements like security and usability.

These levels ensure comprehensive evaluation, targeting specific aspects of the system's design and functionality.

Testing can demonstrate the presence of failures but never prove their absence [30]. Exhaustive testing, which would require exploring all possible inputs and scenarios, is generally infeasible due to the vast input space and combinatorial explosion of configurations [33]. To address this limitation, various testing techniques have been proposed, such as combinatorial testing, which systematically samples combinations of inputs [33], and search-based testing, which applies optimisation techniques to generate effective test cases [34].

2.3.1 Test Adequacy

A central goal of software testing is to reveal faults by executing a program with concrete inputs and observing deviations between its specified and actual behaviour. A test adequacy criterion indicates when a program has been tested “enough”, in the sense that a sufficiently rich set of inputs has been explored to provide confidence that no or few faults remain in the system [35].

Coverage is a widely used basis for defining such criteria. Common examples include branch coverage, which requires tests to exercise all control-flow branches, and instruction (or statement) coverage, which requires executing all lines of code at least once [30]. Graph-based models often underpin these analyses: for instance, control-flow graphs capture the logical structure of a program and support systematic coverage assessment. In the XR domain, as discussed in Section 2.2, scene graphs are used to model immersive environments and can similarly serve as abstractions for defining test criteria and evaluating test quality [36].

Achieving high coverage and reliably checking behavioural correctness is challenging for human testers. Automated testing techniques address this by systematically generating test cases to exercise targeted behaviours and by constructing test oracles (e.g., assertions) to determine whether the observed outcomes are correct [37]. Traditional adequacy metrics, however, typically focus on program elements such as statements or branches, aiming to cover each unit at least once. Empirical studies have shown that there is often a weak correlation between such code coverage and a test suite’s actual fault-detection effectiveness [38, 39].

For Unity applications, code coverage also provides only a partial view of test adequacy. These apps combine logical source code with rich scene configurations (e.g., hierarchies, object properties, and spatial relationships) that drive runtime behaviour. Metrics that consider only the executed code can therefore overlook configuration-dependent functionality, leading to an overestimation of test quality. For example, executing a method handling grab actions contributes to code coverage, but does not confirm that an object is actually grabbed. A misconfigured collider or interaction layer may prevent the grab from activating even when the relevant code has been executed. Code coverage can therefore report successful execution while overlooking faults encoded in the scene configuration. Coverage metrics tailored to XR software, which focus on the interaction definitions in XR environments, are discussed further in Sections 4.5.1 and 5.3.4.

2.3.2 Test Automation

Manual testing is time-consuming and resource-intensive, making it impractical for exhaustive testing of large and complex programs. While human testers remain indispensable for tasks requiring creativity, domain expertise, or nuanced judgment, automated execution of tests is increasingly used to handle repetitive tasks and to improve coverage and repeatability.

In practice, test automation often relies on manually crafted test data, expressed through script-based testing frameworks. For example, tools such as ESPRESSO and UIAUTOMATOR provide script-driven support for Android GUI testing, offering developers intuitive means to express and execute interaction sequences [40]. By improving scalability, consistency, and repeatability, test automation has become a fundamental component of modern software development processes, complementing manual testing to achieve more comprehensive quality assurance.

2.3.3 Automated Testing

The terms *automated testing* and *test automation* are often used interchangeably in both research and practice, with varying definitions. In this thesis, these terms are distinguished for clarity. *Automated testing* denotes approaches that automate both test *generation* and *execution*, whereas *test automation* refers solely to the automated execution of tests derived from manually created test data (e.g., test scripts).

Automated testing reduces reliance on manual effort by automatically constructing test inputs, driving executions, and checking outcomes. This can substantially improve efficiency, consistency, and thoroughness, particularly for complex or large-scale systems. A crucial component of automated testing is the test oracle: the mechanism used to determine whether observed behaviour is correct. Designing accurate and robust oracles remains a challenging problem [37].

Domains such as GUI applications illustrate the benefits of automated testing, as automated tools can systematically explore diverse user interaction scenarios, simulate realistic usage patterns, and validate expected outcomes at scale. In Chapters 4 and 5, similar principles are extended to XR applications, where automated testing and test automation play complementary roles in validating rich, interactive behaviours.

2.3.4 Model-based Testing

Model-based Testing (MBT) is a systematic method that employs explicit behaviour models to encode the intended behaviours of a system under test, from which test cases are systematically derived and executed [41]. This approach provides a structured framework for test generation that aligns with system design specifications. In the context of GUI testing, MBT is widely adopted as it abstracts complex interactive application behaviours into formal models that serve as a basis to be analysed and traversed for test generation. However, exhaustive test generation from these models can be costly, both in terms of computation and execution time [42]. Consequently, many approaches employ guided or targeted strategies that focus on specific coverage goals or test objectives [43, 44].

Finite State Machines (FSM) is a common technique for modelling GUI behaviour, where states represent screens or interface configurations and transitions capture user actions or system events. Notable examples in Android GUI testing include the *window transition graph* [45] and *screen transition graph* [44], which have been proposed to represent navigational structure and drive systematic exploration. In video game testing, models such as **Extended Finite State Machine (EFSM)** [46, 47], and scenario graph [48] are used to capture game states, player actions, and contextual conditions.

Within XR testing, MBT remains a predominant technique [13]. For instance, Corrêa Souza et al. [49] proposed the *requirements flow graph*, which combines *scene graphs* (hierarchical representations of object relationships in virtual environments [50], see Figure 5.2a as an example) with control flow

graphs to construct requirement specifications for VR apps. Building on these prior works, this thesis proposes a graph abstraction, **Interaction Flow Graph (IFG)**, to model 3D user interactions (Chapter 5). This abstraction focuses on the interactions within XR scenes and directly informs test generation. By capturing interaction flows at this level, **IFG** provides a systematic and expressive foundation for automated testing strategies that are tailored to the unique challenges of XR interactions.

2.4 Testing Graphical User Interface

System testing is crucial for GUI apps, complementing unit testing by focusing on user interactions to ensure the software meets requirements and quality standards.

2.4.1 Testing XR Applications

Many XR platforms now include simulation capabilities that allow developers to test and debug apps without requiring physical headset usage, such as Meta XR Simulator¹⁷ and Unity XR Device Simulator¹⁸. For instance, Meta XR Simulator supports Meta Quest app development by enabling keyboard, mouse, or game controller simulation of XR interactions. The simulator also features a valuable *record and replay* function that captures input sequences and verifies consistent behaviour across executions. While record-and-replay is a common testing approach for GUI apps that simplifies the automation of complex usage scenarios [51, 52], it has limitations, such as poor maintainability, where the captured test frequently breaks when the app’s UI changes, requiring substantial manual updates to remain effective [53].

Recently, a growing body of work has started to address the testing of XR applications, each with a different emphasis. In particular, *XR scene testing* (Section 4.1.1) closely parallels traditional GUI testing for 2D interfaces, as it focuses on navigating and exploring XR scenes (analogous to traversing GUI states or activities in 2D GUIs) and activating the expected behaviours of virtual objects (similar to interacting with widgets or interface elements). Pioneering VR scene testing techniques by Wang et al. [54, 55] automate scene exploration and interaction activation, but their subjects are largely restricted to 2D GUI elements in 3D environments, such as on/off button widgets presented in VR settings. By contrast, modern XR applications support rich, spatial behaviours that go beyond traditional 2D elements, allowing virtual objects to behave more like real-world artefacts. For example, a 3D button can be pressed through a poke interaction performed via hand gestures rather than a discrete click. A key contribution of this thesis is to adapt XR scene testing techniques that were originally limited to 2D elements in XR settings to fully 3D interactive elements, with the motivation for these adaptations discussed in detail in Section 4.1.

XR testing does not only concern scene exploration, but it can also target other aspects, such as detecting object misplacement in AR environments [11] or identifying cybersickness in immersive experiences [56]. These diverse targets and objectives emerge directly from the systematic mapping study on XR software testing presented in Chapter 3, which highlights the breadth of quality concerns and fault types relevant to XR systems. Building on these insights, this thesis also draws from established

¹⁷<https://developers.meta.com/horizon/documentation/unity/xrsim-intro>

¹⁸<https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/xr-device-simulator-overview.html>

work on testing other GUI-based applications, such as mobile apps and video games, as discussed in the following subsection.

2.4.2 Testing Other GUI Applications

This subsection discusses testing in two other major categories of GUI applications: 2D and 3D, using mobile apps and video games as representative examples. It highlights how established testing strategies in these domains inform and contrast with the challenges of testing XR applications.

Mobile Application Testing

In 2D GUI applications such as Android apps, system testing typically adopts a black-box perspective, exercising the application solely through its graphical widgets to validate functionality and robustness [15]. Android apps are composed of a set of interconnected *activities*, where each activity corresponds to a screen through which users perform specific tasks, and is populated with various UI elements such as buttons, sliders, menus, and text input fields.

Test automation (i.e., only automate test execution, not test generation) for these apps can be classified into three generations based on the abstraction level of GUI elements [57]:

- **Coordinate-based:** interactions rely on exact screen coordinates;
- **Layout-based:** GUI elements are identified by properties like unique IDs;
- **Image recognition-based:** components are identified through image matching.

Each generation of test automation strategy has its own advantages and disadvantages. For instance, although image recognition-based strategies can achieve higher test case quality than layout-based ones in terms of correctly exercising visible components, they are more likely to suffer from software regression and evolution, as even minor GUI modifications between versions may invalidate previously working test scripts [57].

Automated testing (i.e., automated test generation and execution) tools build on these automation strategies to automatically create and execute test cases within the SUT, typically by orchestrating one or more generations of test automation techniques through one or more underlying test automation frameworks. For example, Android frameworks such as ESPRESSO and UI AUTOMATOR can be combined to construct test cases that exercise a broader range of system elements and improve testing efficiency [40]. These automated testing tools usually synthesise sequences of user interactions (e.g., button clicks, text inputs, swipes) to systematically explore the application and check its behaviour against expected outcomes. In this context, a central objective of Android app testing is to traverse as many activities as possible and to trigger a wide range of UI elements and event handlers, thereby covering diverse execution paths and revealing faults that may stem from complex interaction sequences, activity transitions, or state-dependent behaviours [16].

Video Game Testing

Testing 3D GUIs, such as those found in video games and XR apps, presents significantly greater challenges due to their fine-grained interactivity and complex spatial relationships, which render automated

testing particularly difficult [58]. Emerging methods leveraging advanced techniques, such as evolutionary algorithms and reinforcement learning, aim to systematically explore the 6DoF spaces [59].

Recent work in the video game domain has further highlighted the need for structured, behaviour-driven testing processes for 3D games [60]. This line of work underscores a crucial trend: as 3D and XR environments grow more complex, testing strategies must move beyond low-level code coverage toward higher-level, scenario-based validation that accounts for spatial, temporal, and multimodal interactions. Despite these advances, substantial challenges remain, such as handling flexible camera movements to adjust the user’s point of view and the difficulty of defining meaningful oracles for 3D interactions [59].

Furthermore, one key distinction from testing video games and testing other types of software is that video games typically have specific tasks that require the player to complete. For instance, classic 3D first-person shooter (FPS) game **DOOM**¹⁹, which is widely used for training game-playing agents [61], require the player to eliminate all enemies in a level. As a result, video game testing often needs to balance coverage of diverse user scenarios with the successful completion of specific in-game tasks, which can be formulated as a multi-objective optimisation problem [59].

However, XR applications, while built on 3D interfaces, do not necessarily inherit the same testing assumptions and strategies as video games, because not all XR apps are game-like or goal-driven. XR apps may still define local tasks or user objectives (e.g., completing a training step), but many are designed as tools like educational and productivity applications, where interaction is more open-ended, workflows are less scripted, and success is measured by usability, accuracy, or safety rather than “winning” or completing a level. This implies that, unlike video game testing that often optimises for both scenario coverage and explicit task completion, XR app testing must additionally account for broader patterns of everyday use and varied user contexts.

¹⁹[https://en.wikipedia.org/wiki/Doom_\(franchise\)](https://en.wikipedia.org/wiki/Doom_(franchise))

Chapter 3

A Systematic Mapping Study of XR Software Testing

This chapter builds directly on the background established in Chapter 2, where we introduced the fundamentals of XR application development and interaction. Having outlined how XR systems are constructed and how users engage with them, the focus now shifts to how these applications are tested in practice. In particular, this chapter examines the state of the art in XR software testing and investigates how existing approaches address (or attempt to address) the distinctive characteristics of XR systems.

To address this gap of no available comprehensive systematic review studies dedicated to XR software testing, we present a systematic mapping study on XR software testing. Systematic mapping is a methodology designed to survey the literature, provide a comprehensive overview of a topic, identify research gaps, and offer insights into future research directions. By carefully following the guidelines proposed by Petersen et al. [62], we selected a total of 34 **Primary Studies (PS)** as the subjects for this mapping (see Appendix A.1 for the details of the studies).

Drawing on these studies, we characterise publication trends, research topics, test activities and concerns, techniques, and validation practices in XR testing. We also compile and report the tools and datasets used in the primary studies to support replication and facilitate future work. Finally, we discuss the limitations and challenges revealed by the mapping and outline avenues for advancing XR software testing, thereby laying the groundwork for the test automation approaches developed in the subsequent chapters.

The main contributions of this systematic mapping study are as follows:

- We provide an in-depth survey of the current software testing methods for XR apps, shedding light on the state-of-the-art in this emerging domain. The data extraction template used to derive these findings is included as part of the study.
- We compile a repository of existing tools and datasets used in XR software testing to support future research in the field. The repository, along with the data extraction results, is publicly available at: sites.google.com/view/xr-testing.
- We identify critical challenges in XR software testing and outline potential research directions to address these challenges and advance the field.

This chapter is structured as follows. Section 3.1 discusses the motivation behind this work and summarises relevant related studies. Section 3.2 details our methodology, including the process for searching and selecting relevant literature. Section 3.3 presents the results of our study and answers to our research questions. Section 3.4 discusses the findings and explores their implications for the field of XR software testing. Section 3.5 answers ORQ1 from Section 1.1, while Section 3.6 summarises the key contributions and concludes this chapter.

The content of this chapter is based on “Ruizhen Gu, José Miguel Rojas and Donghwan Shin. Software Testing for Extended Reality Applications: A Systematic Mapping Study” published in the special issue of *Virtual and Augmented Reality Software Engineering* in the journal *Automated Software Engineering*, 2025 [13].

3.1 Motivation of the Mapping Study

To explore related work on XR software testing, we conducted a preliminary literature search. During the process, we identified a few secondary studies, such as systematic mapping studies and literature reviews, vaguely related to testing XR applications. For example, Börsting et al. [63] conducted an informal review of software engineering techniques for AR apps, analysing their applicability across various engineering phases, including requirement engineering, implementation and testing. With regard to testing, the study emphasised the importance of interaction testing and test automation for AR user interfaces. However, it noted significant challenges, such as the lack of formal definitions for AR-specific interactions and the need for tailored testing approaches for unique AR components (e.g., animations, transformations). These challenges pose significant obstacles to achieving effective automated interaction testing. Additionally, the study discussed the prevalent reliance on user-based usability testing in AR, which often involves labour-intensive user studies. This reliance underscores the need for more reproducible and efficient testing approaches to reduce manual effort and improve scalability. While this study offered a broad overview of software engineering for AR, our work specifically focuses on the unique challenges and methodologies of software testing for XR apps (i.e., including but not limited to AR).

Kuri et al. [64] conducted a mapping study focusing on software quality metrics for validating VR products (e.g., code quality, audio quality, quality of experience) rather than software testing techniques. The study found that the existing metrics are primarily tailored to specific app types, such as educational apps, making them less applicable to other domains like manufacturing, for instance. Researchers tend to develop custom quality metrics and evaluation methodologies, highlighting the need for a general framework that assesses VR app quality across various dimensions (e.g., code, video, audio). With the majority of existing metrics focused on the quality of experience and relying on manual evaluation, the study highlights the need for automated, objective methods or metrics to assess software quality.

3.1.1 Usability of XR Applications

While our preliminary literature search did not uncover any systematic review dedicated explicitly to XR software testing, we did find several studies evaluating XR systems’ *usability*, i.e., the ease of use of specific software systems [65].

Ramaseri Chandra et al. [66] reviewed usability and performance evaluation in VR systems. They identified key usability issues including *health and safety issues*, *social issues* and *sensory constraints*. The study also identified usability evaluation methods, such as cognitive evaluation [67], user analysis [68], and group testing [69].

Dey et al. [70] conducted a systematic review of AR usability studies from 2005 to 2014. They analysed 369 user studies across various application domains such as education, entertainment, and industry. The most common data collection method was questionnaires, resulting in subjective ratings being the most widely used measure. Kim et al. [27] reviewed VR systems from a human-computer interaction (HCI) perspective. The findings aligned with those of Dey et al. [70], highlighting *subjective measures* as the dominant approach for evaluating VR/AR usability.

Both Ramaseri Chandra et al. [66] and Kim et al. [27] identified *cybersickness* as a significant usability issue in XR systems. Cybersickness, a form of visually-induced motion sickness experienced in immersive environments, manifests through symptoms such as nausea, disorientation and headaches [71]. Various factors may cause cybersickness regarding individuals (e.g., illness, posture), devices (e.g., lag, calibration), and tasks (e.g., control, duration) [71]. Studies aiming to comprehensively assess cybersickness by employing subjective and/or objective measures exist. Subjective measures, e.g., the Simulator Sickness Questionnaire (SSQ) [72], evaluate participants' self-reported symptoms. Objective measures, in contrast, primarily involve real-time physiological data collection such as heart rate variability (HRV) or eye tracking while participants perform specific tasks [73–75].

Yang et al. [76] conducted a systematic review focusing on the use of machine learning (ML) techniques to study cybersickness. The review examined 26 studies that utilised ML approaches with biometric and neuro-physiological signals, such as electroencephalogram (EEG) and electrocardiogram (ECG) data obtained from wearable devices, for the automated detection of cybersickness.

These studies emphasise the unique usability challenges posed by XR systems compared to traditional software. Testing user interactions in XR is essential, especially because human behaviour in these environments is highly complex and cannot be mathematically modelled to guarantee predictable outcomes [14]. While automated approaches show potential in addressing usability issues like cybersickness detection, they largely depend on user involvement. This reliance on manual testing or *live* user data collection is time-consuming and costly.

Although these findings highlight the importance of *user-centric* evaluations, they also expose a gap in exploring *software-centric* testing approaches. Unlike user-centric methods, software-centric testing can detect failures earlier in the development process and offer more efficient, systematic, and automated testing capabilities. Our mapping study seeks to bridge this gap by examining studies that address usability issues from a software-centric perspective.

3.1.2 Motivation

There is a noticeable gap in the literature regarding a comprehensive overview of testing practices for XR applications. While recent research on VR app testing highlights the scarcity of literature on software engineering practices specifically for the VR domain [18], this observation extends to the broader XR domain, which remains significantly underexplored. The gap underscores the need for a formal and in-depth mapping study to analyse existing evidence on XR app testing challenges and techniques, identify research gaps, and suggest further research directions, potentially including systematic literature reviews

on specific aspects of XR testing.

Most secondary studies related to XR software testing have primarily focused on usability, a trend that aligns with the findings of Kuri et al. [64]. While usability is important for XR experiences, this limited focus has a significant gap in understanding the broader landscape of XR software testing methodologies. To fill this gap, we conduct this systematic mapping study to provide a comprehensive overview of software testing methodologies for XR apps. Our focus is on techniques and frameworks that prioritise *software requirements*, specifically addressing XR software testing challenges.

Our study adopts an inclusive approach by incorporating empirical studies that, while not directly proposing testing methods, offer valuable insights or theories beneficial to testing. For example, studies analysing common bug types in XR apps provide foundational knowledge that can inform the development of testing strategies. This broader inclusion ensures a more holistic understanding of XR software testing.

3.2 Mapping Study Methodology

This systematic mapping study follows the guidelines proposed by Petersen et al. [62] and is inspired by other systematic mapping studies [19, 77]. As shown in Figure 3.1, our mapping process consists of three stages:

- **Planning the mapping**, where the research questions and the scope of the literature search are formulated;
- **Conducting the mapping**, where the authors specify a search strategy, search, and select primary studies, then apply classification and data extraction processes to them subsequently;
- **Reporting the mapping**, presenting the outcomes of the study, with complete details of primary studies and extracted data available in the appendix and repository.

3.2.1 Planning the Mapping

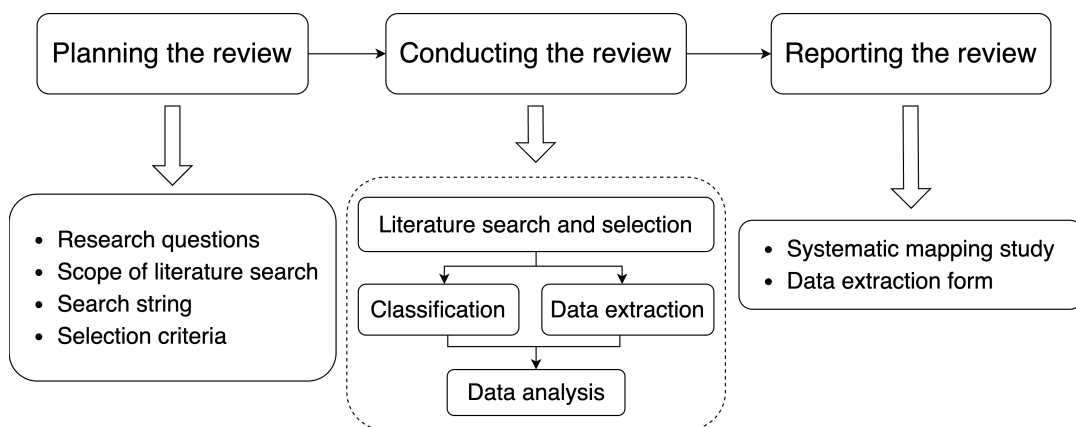


Figure 3.1: Process of the mapping study

Research Questions

This study aims to develop a comprehensive classification scheme by analysing relevant evidence and insights from the existing literature on software testing for XR applications. The scope extends beyond the studies that introduce novel testing techniques for XR apps, encompassing a broader range of research, including empirical studies that provide valuable information for XR software testing (e.g., analysing common bug types within XR software). Moreover, the study seeks to identify research gaps and challenges and outline future research directions. We therefore formulate the following research questions (RQs):

- RQ1: *What is the current status of XR application testing research?*** This question provides an overview of the current landscape in XR software testing research. It will explore general aspects such as the number of publications over recent years, major publication venues, and common research types. Additionally, we will investigate the most discussed and emerging topics within XR testing research and which XR technologies (e.g., VR, AR) are the primary focus of current research.
- RQ2: *What are the test facets involved in XR applications?*** This research question aims to provide a comprehensive overview of software testing practices for XR applications, including test activities, concerns, and techniques.
- RQ2.1: *What test activities are involved in XR applications?*** This sub-question seeks to identify and categorise the test activities relevant to XR app testing, such as test data generation and test execution. By exploring these activities, we aim to understand the current practices in XR app testing and identify potential areas for improvement.
- RQ2.2: *What are the primary test concerns in XR applications?*** This sub-question focuses on the key concerns in testing XR apps, particularly the *objectives* (e.g., verifying functionality, improving usability) and *targets* (e.g., user interfaces, XR-specific requirements) of testing. Understanding these concerns helps to clarify the goals and challenges in XR app testing.
- RQ2.3: *What test techniques are employed in XR applications?*** This sub-question investigates the specific testing techniques used on XR apps, such as random testing, mutation testing, and model-based testing. By analysing these techniques, we aim to investigate the common testing approaches for XR systems.
- RQ3: *To what extent are XR testing approaches validated?*** This RQ explores how the testing methodologies for XR apps are validated. We assess the metrics and environments (e.g., simulation or real devices) used to evaluate their effectiveness.

Search String

As the research questions aim to investigate the current research status of XR software testing, it is possible that some studies do not directly focus on testing techniques but instead analyse other aspects related to testing. For example, some studies may investigate the characteristics or challenges of XR systems, such as identifying issues or limitations in XR apps, which can indirectly inform testing

Table 3.1: PICOC criteria applied to this study

Criterion	Description
Population	XR-related software
Intervention	Testing techniques or relevant studies addressing testing aspects
Comparison	Not applicable
Outcome	Insights into methodologies or practices for testing XR applications
Context	Peer-reviewed publications

Table 3.2: Synonyms in $\$XR$, $\$XR_{acr}$, $\$T$, and $\$B$

Synonyms	Metadata
$\$XR$ “virtual reality” OR “augmented reality” OR “mixed reality” OR “extended reality”	title, full text
$\$XR_{acr}$ VR OR AR OR XR OR MR	title
$\$T$ test OR detect OR detection OR verify OR verification	title
$\$B$ bug OR fault OR defect OR error	title

practices. Specifically, we tackle this by also including studies that explore the nature of *bugs* (or *faults*, etc.) in XR apps, aiming to collect studies analysing them or proposing techniques to detect them. Since XR software testing is still in its early stages, identifying and understanding such issues may still be underexplored.

To ensure that the search process identifies primary studies addressing the RQs, we followed the guidelines by Kitchenham and Charters [78] to break down the research questions into individual facets using the PICOC model (population, intervention, comparison, outcomes, and context), which then serves as the foundation for designing the search query. The PICOC model is defined in Table 3.1 and based on this model, the search query for the digital libraries is:

$$\text{Search String} = (\$XR \text{ AND } \$XR_{acr}) \text{ AND } (\$T \text{ OR } \$B)$$

Here, $\$XR$ denotes the synonyms of *extended reality*; $\$XR_{acr}$ are the acronyms corresponding to these terms; $\$T$ represents the synonyms of *testing*; and $\$B$ are the synonyms of *bugs*. The synonyms used in the search query are detailed in Table 3.2.

The search string is searched with the studies’ titles, and $\$XR$ is additionally searched with full text to ensure $\$XR_{acr}$ in the titles genuinely refers to extended reality. For example, MR also stands for “magnetic resonance” [79], which would yield irrelevant results. By structuring the search string this way, we avoid retrieving extraneous findings related to unrelated fields like medicine.

Search Evaluation

To evaluate the quality of the search string, we follow the guidelines of Petersen et al. [62], using a test set of relevant papers, all of which should be found by the search string. We identified eight studies during our initial literature review [10, 26, 36, 80–84] to compose the test set. These studies cover different testing aspects (e.g., functionality, usability, and empirical studies), ensuring that the search

results include studies with diverse focuses. We refine the search string iteratively until the search results contain all the studies from the test set.

Digital Library

To cover as many relevant studies as possible, we conduct our search using OpenAlex¹, a bibliographic database that indexes scientific papers from major digital libraries, including IEEE Xplore Digital Library², ACM Digital Library³, and Scopus⁴ [85]. OpenAlex's filter features allow us to restrict the search to studies in the fields of **Computer Science** and **Engineering**, reducing irrelevant search results [19, 86]. This restriction excludes papers focused on the applications of XR in other disciplines, such as **Medicine** and **Social Sciences**.

Selection Criteria

After executing the search string in the digital library, a list of potentially relevant studies is retrieved. To ensure that we only include studies aligned with the mapping study's objectives and capable of answering the research questions, we developed a set of selection criteria [87]. As suggested by Petersen et al. [62], we piloted the selection criteria (using a sample of 100 studies from the search results) and refined them until consensus was reached collaboratively by myself and my two supervisors that the criteria effectively included relevant studies and excluded irrelevant ones. As a result of this process, we applied the following inclusion criteria (ICs) and exclusion criteria (ECs):

- IC1:** Studies must involve software testing techniques, challenges or limitations for extended reality software applications.
- IC2:** Studies published between January 2000 to July 2024.
- IC3:** Studies written in English, published in peer-reviewed journals or conference proceedings, and available in full text.
- IC4:** Studies must be primary studies rather than secondary studies such as systematic literature reviews.
- EC1:** The focus of the studies is not testing but other software development aspects, such as analysis, design or implementation.
- EC2:** Studies do not focus on software-related aspects, such as requirements or integrity, but instead emphasise other areas like user perceptions or hardware configurations.
- EC3:** Studies are duplicated in the search results, including extended versions of existing results.
- EC4:** Studies published in the form of abstract or panel discussion.

General non-XR 3D interactive environments, such as video games, may share challenges with testing XR apps, for instance, interaction modelling, manipulation, and interaction constraints. However,

¹<https://openalex.org/>

²<https://ieeexplore.ieee.org/>

³<https://dl.acm.org/>

⁴<https://www.scopus.com/>

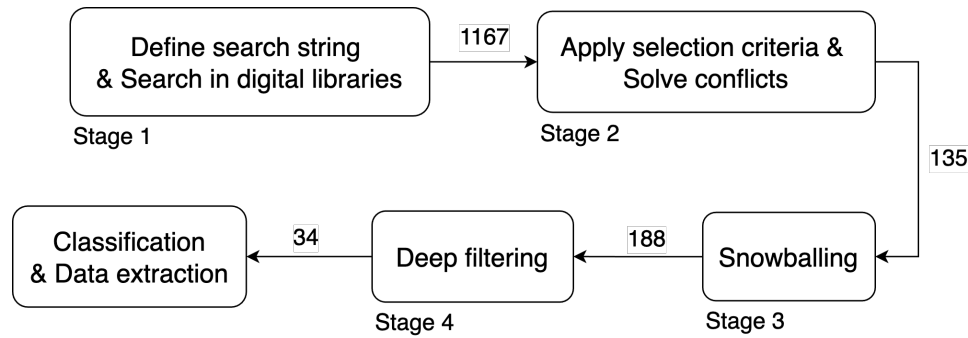


Figure 3.2: Literature search and selection process. The numbers on the arrows indicate the number of studies provided to the next stage.

they were excluded under IC1 because this mapping study specifically investigates XR software. Including non-XR subjects would broaden the study into general interactive-system testing and make it more difficult to identify findings directly associated with XR. Nevertheless, techniques developed for testing non-XR 3D environments may be transferable to XR testing and are considered relevant in the background (Chapter 2).

3.2.2 Conducting the Mapping

To streamline the methodological information, this subsection provides the fundamental and essential information for conducting this mapping study, including search and selection strategy, classification scheme and data extraction, with more details that can be found at the online repository at <https://sites.google.com/view/xr-testing>.

Search and Selection Strategy

The search strategy consists of an automated search and manual snowballing. The steps of the search and selection process are illustrated in Figure 3.2 and detailed below:

- **Stage 1:** The search process begins with the formulation of an initial search string (Section 3.2.1). The search string is used to retrieve an initial set of potentially relevant studies from the digital library (Section 3.2.1).
- **Stage 2:** To refine the results from the initial search, the selection criteria defined in Section 3.2.1⁵ are applied by reviewing the title and abstract of each study. Based on relevance, each study is labelled as either “include”, “exclude” or “unclear”. To minimise bias (as advised by Petersen et al. [62]), the three authors independently review each study and the *most inclusive* decision rules are used: Only studies unanimously excluded by all authors are discarded immediately; Studies with conflicting labels are considered “conflicts” and resolved through a consensus meeting among the authors. This approach ensures rigour and reduces the risk of overlooking relevant studies, as further detailed in Section 3.2.4.

⁵Excluding IC2 and IC3, which are applied automatically using OpenAlex’s filtering feature. The full filters applied by OpenAlex can be found at <https://bit.ly/3zTxwCS>.

- **Stage 3:** Backward snowballing is conducted on the studies retained after applying the selection criteria ([62]). This stage is performed in parallel with deep filtering, classification, and data extraction. Following the guidelines by Wohlin [88], each study’s reference list is reviewed in two steps: (1) basic exclusion: exclude studies that do not meet basic criteria, such as language, publication year, and venue; (2) relevance assessment: evaluate relevance based on titles and context in which each study is referenced, excluding irrelevant works. Snowballing is an iterative process aimed at minimising the risk of missing relevant studies, as further detailed in Section 3.2.4; we conducted three iterations of snowballing, ensuring comprehensive coverage of potential primary studies.
- **Stage 4:** Some selection criteria, particularly *E1* and *E2* (Section 3.2.1), required a detailed evaluation beyond titles and abstracts. In this stage, the retained studies undergo a full-text review, focusing specifically on exclusion criteria *E1* and *E2* to ensure only relevant studies are given to the following stages.

After completing this process, the remaining studies form the *primary studies* for this mapping study. These studies proceed to subsequent phases, including classification and data extraction. The full list can be found in Appendix A.1.

Classification Scheme

The classification scheme organises the primary studies into broad categories to provide a structured overview of the research field [78]. Following the guidelines by Petersen et al. [62], we applied both topic-independent and topic-specific classifications.

For topic-independent classifications, studies are categorised by their publication venue type and research type, as suggested by Petersen et al. [62].

For topic-specific classification, the goal is to categorise studies according to their primary focus areas or themes. This is achieved through a systematic keywording of abstracts method, as outlined by Petersen et al. [87].

More specifically, the topic-specific classification process consists of two phases. In the first phase, the authors read the abstract of each study to identify keywords and concepts reflecting the research problem and focus area. If the abstracts lack clarity, the introduction and conclusion sections are reviewed for additional context to make the decisions. In the second phase, the keywords from different studies are consolidated into broader categories to provide a high-level understanding of the research topics. For example, studies with keywords like “*automated functional testing*, *requirements engineering*, *test generation*” and “*agent-based testing*, *automated testing*, *interactive systems*” could be merged into the broader category of “*automated testing*”.

To ensure reliability and reduce bias, the classification process involves a collaborative approach with three authors. One author performs the classification for all primary studies, and two additional authors independently classify half of the primary studies each. Disagreements are resolved through a consensus meeting, following the guidelines of Petersen et al. [62].

This approach enhances the accuracy and reliability of the classification, ensuring a robust understanding of the research domain.

Data Extraction

To address the research questions outlined in Section 3.2.1, data were systematically extracted from each primary study, following the guidelines by Kitchenham and Charters [78]. We collect general publication information, such as venues and years, alongside specific data fields relevant to answering the research questions. To standardise this process, we compose a data extraction form, as shown in Table 3.3. Each data extraction field specifies: (1) Data item: the type of information to be extracted; (2) Description: a brief explanation of the data item with its possible values to guide extraction; (3) RQ: the specific research question the data item aims to address. The outcomes of this process consist of data items and their corresponding extracted values for each primary study.

To summarise the contribution of the primary studies, three data fields related to the datasets and tools utilised or proposed by the studies are included. These contributions are discussed in Section 3.4.

For clarity and consistency, the data extraction form was pilot-tested using the initial set of studies (as described in Section 3.2.1), as recommended by Kitchenham and Charters [78]. To mitigate bias, all three authors were involved in the data extraction process. The first author performed the initial data extraction for all primary studies, and the other two authors collaboratively reviewed the extracted data. Any disagreements were discussed and resolved in consensus meetings.

After the classification and data extraction processes, we analysed and formulated the retrieved data to address the RQs. The analysis results are presented in Section 3.3.

Data Analysis

The purpose of this task is to analyse the classification and extracted data to address the research questions (RQs) outlined in Section 3.2.1 [78]. For instance, to answer RQ1, which investigates the current status of XR software testing research, we analyse the topics and research types derived from the classification, with relevant extracted data to provide a comprehensive overview of the XR software testing research landscape. Each RQ is mapped to specific data fields in the data extraction form (see Table 3.3), ensuring all RQs are adequately addressed.

The analysis results and findings are presented in Section 3.3. The extracted data are systematically tabulated to align with the RQs, highlighting key similarities and differences across the studies.

3.2.3 Reporting the Mapping

The report contains two parts. The first part comprises this chapter, which outlines the study's methodology and findings, and the second part is the data extraction form, which details the raw data collected and the basis for the study's conclusions. The complete results of the classification and data extraction are publicly accessible at: <https://sites.google.com/view/xr-testing>.

3.2.4 Threats to Validity

This section addresses potential threats to the completeness of the literature search and selection process. This threat is influenced by the search string choice, the bibliographic database limitations, and the robustness of the literature selection process.

To mitigate the risk of excluding relevant studies during the selection process, three authors independently screened all search results, following the most inclusive approach. Conflicts were resolved

Table 3.3: Data Extraction Form

Data Item	Description (and possible values)	RQ
Title	Title of the study	-
Authors	Names of the study's authors	-
Year	Publication year of the study	<i>RQ1</i>
Venue	Name of the publication venue	-
Venue Type	Type of the venue (e.g., conference, workshop, journal)	<i>RQ1</i>
Topic	Primary focus area of the study (e.g., usability testing, automated testing)	<i>RQ1</i>
Research Type	Research type of the study (e.g., solution proposal, validation research)	<i>RQ1</i>
Technology	Immersive technology specified in the study (e.g., XR, VR, AR)	<i>RQ1</i>
Test Activity	Specific Test activity involved (e.g., test data generation, test tool development, test execution)	<i>RQ2.1</i>
Test Objective	Primary objective of the testing approach (e.g., functionality, usability, security)	<i>RQ2.2</i>
Test Target	Focus of the testing approaches (e.g., general, GUI, XR-specific requirements)	<i>RQ2.2</i>
Test Level	Scope of the testing activities (i.e., unit testing, integration testing, system testing)	<i>RQ2.3</i>
Test Type	Type of testing performed (e.g., black box, white box)	<i>RQ2.3</i>
Test Technique	Core methodologies used for testing (e.g., search-based testing, mutation testing)	<i>RQ2.3</i>
Evaluation Environment	Environment for evaluating the testing approaches (e.g., Unity Editor, HMD, mobile device)	<i>RQ3</i>
Metrics	Metrics used to evaluate testing techniques (e.g., coverage, mutation score)	<i>RQ3</i>
Dataset_{train}	Details of datasets used for training machine learning-based approaches, including content types (e.g., video, image) and dataset size	<i>discuss.</i>
Dataset_{eval}	Details of datasets for evaluating testing techniques, including content types and size	<i>discuss.</i>
Tool	Details of software tools proposed or used by the study	<i>discuss.</i>

through consensus discussions. While the inter-rater agreement was not formally measured, our approach prioritised achieving absolute consensus to maximise the inclusion of relevant studies.

We acknowledge potential limitations in our search strategy—solely using OpenAlex as the digital library—may affect the thoroughness of our study. While OpenAlex indexes publications from major digital libraries such as IEEE, ACM, and Scopus, we recognise that it does not index some relevant studies (e.g., those that may exist exclusively in specialised venues or databases). Additionally, very recently published studies might not have been indexed in OpenAlex at the time of our search, creating a temporal bias against the latest research.

The classification of immersive technologies relies on the terminology and stated scope of the primary studies. Because authors may use terms such as XR, VR, AR, and MR inconsistently, the resulting categories may not represent a uniform technology taxonomy. We retained the authors' terminology to avoid inferring unsupported scopes and reviewed classifications to mitigate inconsistent interpretation.

To mitigate these risks, we adopted an exhaustive iterative snowballing strategy. We systematically identified relevant studies from the reference lists of each included study and repeated this process until no new relevant studies were discovered. Our snowballing process involved three iterations, which significantly reduced the likelihood of missing important contributions to the field.

3.3 Results

In this section, we present the results of this mapping study, including information about the search and selection results of the primary studies, and answer the research questions based on the information from the primary studies.

The initial search returned 1167 studies from OpenAlex. The selection process, as outlined in Section 3.2.2, reduced this to 135 studies after applying the selection criteria. In parallel with deep filtering, backward snowballing identified 53 additional studies, resulting in a final set of 34 primary studies for this mapping study (see Appendix A.1 for the complete list of primary studies).

Figure 3.3 shows the publication trend of the primary studies from January 2000 to July 2024. The data reveal that the XR software testing research field was relatively inactive before 2017, with only two studies published. However, starting in 2017, the number of studies began to increase gradually, reaching a peak in 2023, with ten studies published that year. As the literature search for this study was conducted in July 2024, the number of studies published in 2024 was not completely recorded.

3.3.1 RQ1: Research Status

To address RQ1, which explores the current status of research in XR software testing, we present the classification results of the primary studies. Additionally, we analyse the immersive technologies (e.g., AR, VR) featured as testing subjects in these studies, providing insights into the technologies most frequently explored in this domain.

The studies are classified based on the following criteria:

- Venue types (e.g., conferences, journals),
- Study topics (e.g., automated testing, usability testing), and
- Research types (e.g., solution proposal, evaluation research).

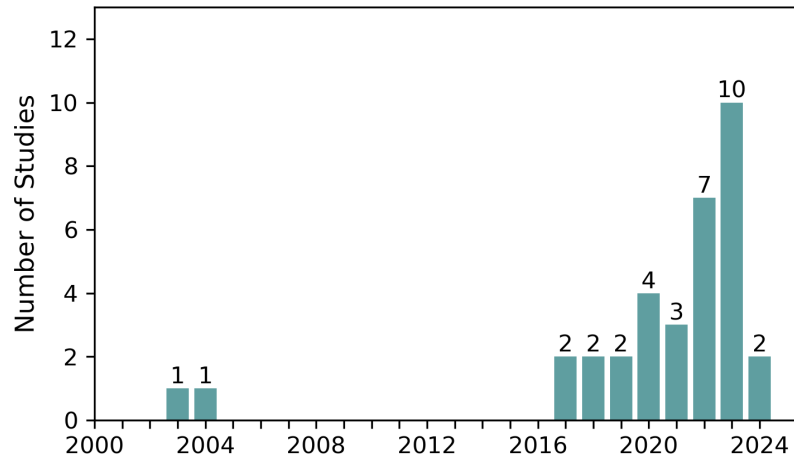


Figure 3.3: Publication Years of Primary Studies

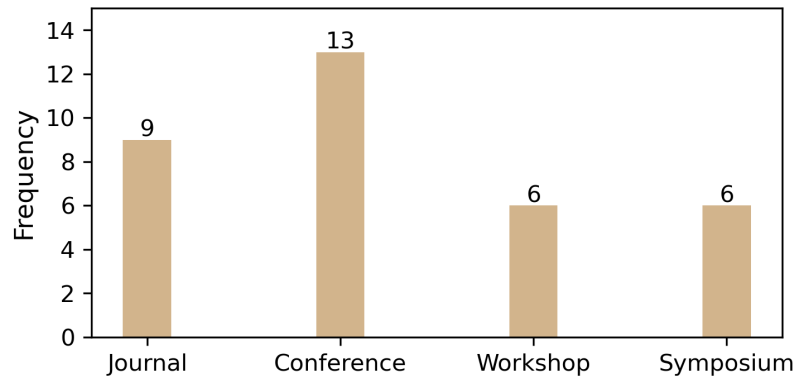


Figure 3.4: Distribution of publication venue types for the primary studies

Venues

The distribution of primary studies by venue type is presented in Figure 3.4. Conferences emerge as the dominant venue type, accounting for about 38% of the primary studies. Journals are the second most common venue, publishing approximately 26% of the studies. Workshops and symposiums each represent about 18% of the total primary studies⁶. The complete list of all publication venues for the primary studies can be found in Appendix A.2 and is available at: <https://sites.google.com/view/xr-testing>.

To categorise the primary studies based on the domains of computer science conferences, we define the following categories based on the primary studies' venue names:

- **SE:** Software engineering, including software testing
- **HCI:** Human-computer interaction

⁶The distinction between symposiums and conferences is not always clear. For classification purposes, venues with "conference" in their title are categorised as conferences and those with "symposium" are classified as symposiums

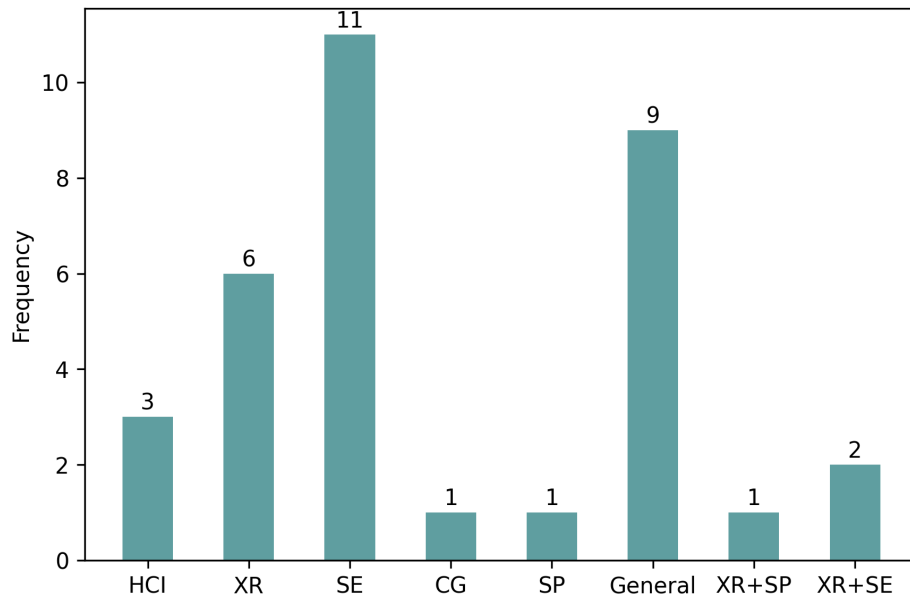


Figure 3.5: Distribution of publication venue domains for the primary studies

- **SP:** Security and privacy
- **CG:** Computer graphics
- **XR:** Extended reality
- **General:** Venues without a specific focus on the above domains

Figure 3.5 illustrates the distribution of primary studies across these domains. Half of the studies are distributed between *SE* and *XR* venues, highlighting these areas as key sources of XR software testing research. Around a quarter of the studies fall under the *General* category, encompassing venues focused on diverse or broad computer science and engineering fields, such as dependable computing and applied sciences. A smaller number of studies are published in *SP* and *HCI* venues, indicating the relevance of testing for addressing security and usability challenges in XR applications.

Notably, a few venues span multiple domains. For instance, *International Workshop on Virtual and Augmented Reality Software Engineering* covers both *SE* and *XR*. Although the studies published in such cross-domain ($XR + X$) venues are relatively few (3 out of 44), they demonstrate the growing research interest that applies findings from other computer science fields to advance XR research. This distribution underscores the interdisciplinary nature of XR software testing and the potential for future collaborations across diverse fields.

Topics

As explained in Section 3.2.2, we carefully categorise the primary studies into eight topics. Table 3.4 presents the resulting classification. Overall, *automated testing* emerges as the most prominent topic, with seven studies (21% of the total). *Usability testing* and *security testing* each account for six studies

Table 3.4: Research topics in the primary studies

Topic	Primary studies	Number
XR-specific testing	PS16, PS17, PS20, PS22	4
Scene testing	PS4, PS12, PS32, PS33	4
Security testing	PS2, PS13, PS14, PS15, PS18, PS30	6
Usability testing	PS1, PS7, PS10, PS11, PS19, PS21	6
Automated testing	PS3, PS5, PS8, PS23, PS24, PS25, PS27	7
Test automation	PS9, PS26, PS34	3
Open-source projects	PS6, PS28, PS31	3
Stakeholder survey	PS29	1

(18%). *XR-specific testing* and *scene testing*, each contribute four studies (12%), while *test automation* and *open-source projects* each comprise three studies (9%). Finally, *stakeholder survey* represents the least explored topic, with only one study (3%).

Below we provide details for each topic, including:

- **Description:** a general overview of the topic,
- **Examples:** summaries of representative studies within the topic, and
- **Implications:** insights into the potential consequences, proposed solutions, or general guidelines for addressing the challenges associated with the topic.

XR-specific Testing

- **Description:** Testing techniques designed for XR-specific requirements, such as collision, occlusion, registration, and tracking [14].
- **Examples:** Collision and occlusion are critical real-time requirements in XR systems. Collision refers to the interaction between objects (virtual or physical) when they come into contact, and occlusion occurs when objects (virtual or physical) block each other from view [14, 29]. Testing these aspects ensures realistic interactions between virtual and real objects. PS17 [18] proposes an approach that uses automatic test data generation to explore virtual scenes and detect collision and camera occlusion faults in VR apps.
- **Implications:** Testing XR-specific requirements demands a deep understanding of their impact on software behaviour, which calls for novel testing techniques tailored to these unique challenges.

Scene Testing

- **Description:** Focuses on validating XR functionality by *exploring XR scenes* through interacting with virtual objects and navigating within virtual scenes.
- **Examples:** PS32 [26] and PS33 [80] introduced testing techniques for VR scene testing, focusing on exploring virtual environments, triggering interactable virtual objects, and optimising interaction routes to enhance exploration efficiency.

- **Implications:** Scene testing extends principles of 2D GUI testing into 3D environments, targeting interactable components. As discussed in Section 2.2, unlike 2D apps, XR apps rely on 6DoF interactions, adding complexity to their testing.

Usability Testing

- **Description:** Focuses on identifying usability issues in XR software, rather than addressing functional defects.
- **Examples:** PS1 [89], PS19 [12], and PS21 [90] proposed approaches for detecting *cybersickness*, a prevalent side effect in VR systems. These methods utilise the visual content, such as screenshots from applications, to analyse and identify potential usability concerns.
- **Implications:** While most existing research employs user-centric methods, such as user studies, as discussed in Section 3.1.1, these studies also analyse the root causes of usability issues. This understanding provides a foundation for developing software-centric approaches to systematically identify and mitigate usability problems in XR systems.

Security Testing

- **Description:** Focuses on identifying and mitigating *security* and *privacy* issues in XR software. Security testing aims to detect system intrusion and address vulnerabilities, while privacy testing protects users' sensitive information.
- **Examples:** PS18 [91] introduces a framework to address privacy issues in mobile AR apps, specifically focusing on hidden operations that compromise user data. On the other hand, PS15 [92] targets attacks designed to disrupt user experiences in VR environments.
- **Implications:** Security and privacy testing in XR systems should account for XR-specific features, such as interaction with physical environments and differentiation between real and world objects [93].

Automated Testing

- **Description:** Proposes testing techniques that automate both test *generation* and *execution* for functional testing, without specifically targeting XR-specific or non-functional requirements.
- **Examples:** PS23 [82] and PS27 [11] tackle the oracle problem of object misplacement in AR apps. These studies analyse screenshots depicting various object misplacement scenarios and use crowdsourcing to generate test data. Neural networks are then employed to automatically detect misplacement errors in the screenshots, achieving full automation for both test data generation and execution.
- **Implications:** Effective test automation often requires a thorough understanding of the SUT, particularly its requirements or test oracles. This understanding is crucial for systematically generating reliable test data and ensuring accurate testing outcomes.

Test Automation

- **Description:** Automating test execution while relying on manually created test data, distinguishing it from *automated testing*, which automates both test generation and execution.
- **Examples:** PS34 [94] presents a unit testing framework for Unity-based VR/AR apps. It enables interaction with UI elements through manually created test scripts.
- **Implications:** Test automation often involves script-based testing frameworks. For instance, tools like *Espresso*⁷, a script-based testing tool for Android apps, exemplify this approach by enabling automated test execution driven by predefined scripts.

Open-source Projects

- **Description:** Empirical studies that analyse open-source XR projects to gain insights into current testing practices and challenges.
- **Examples:** PS6 [84] examines bugs in open-source WebXR projects, building a taxonomy based on symptoms and root causes. Similarly, PS31 [81] investigates open-source VR projects, revealing a lack of automated tests and insufficient quality in existing tests.
- **Implications:** Studying open-source projects provides valuable empirical data for understanding XR testing practices, particularly in this emerging field. These analyses offer practical recommendations for researchers and practitioners to improve strategies for XR software testing.

Stakeholder Survey

- **Description:** Surveys and interviews conducted with real-world stakeholders, such as users, developers, and researchers of XR software, aim to gather diverse perspectives on practices and challenges in XR software testing.
- **Examples:** PS29 [10] surveys XR stakeholders to understand software testing practices in VR apps, highlighting key concerns and common faults, such as interaction issues and crashes, within these systems.
- **Implications:** Stakeholder surveys provide empirical insights grounded in real-world experiences, offering perspectives distinct from those derived from analysing open-source projects. These insights contribute to understanding current challenges and can guide the improvement of XR testing.

Research Types

For research types, we adopt the classification categories proposed by Wieringa et al. [95]: *solution proposal*, *validation research*, *evaluation research*, *philosophical paper*, *opinion paper*, and *experience paper*. These categories have been carefully reviewed and adapted to align with the scope of XR software testing, ensuring relevance to the primary studies. Notably, a study can span multiple categories, such as studies that propose solutions and include initial validation.

⁷<https://developer.android.com/training/testing/espresso>

Table 3.5: Research types in the primary studies

Research type	Number of studies
Validation research	2
Evaluation research	5
Solution proposal	6
Solution proposal <i>and</i> validation research	15
Solution proposal <i>and</i> evaluation research	4
Philosophical papers	2

Table 3.5 summarises the results. *Solution proposal and validation research* emerges as the most prevalent research type, encompassing 15 studies (44% of the total). *Solution proposal* accounts for 6 studies (18%), *evaluation research* includes 5 studies (15%), and *solution proposal and evaluation research* covers 4 (12%). On the other hand, *validation research* and *philosophical papers* each account for 2 studies (6%). Notably, no *opinion paper* and *experience paper* were identified in the primary studies.

Similar to the approach in Section 3.3.1, we present the *description* and *examples* of each research type based on the primary studies.

Validation Research

- **Description:** Providing initial validations of solutions or problems, typically involving limited experiments in controlled, simplified settings, such as toy applications (e.g., research prototypes or low-popularity open-source apps) and datasets of minimal complexity.
- **Examples:** PS12 [96] introduces a black-box testing approach for a VR musical instrument game, using equivalence partition to design the test cases. Validation was limited to manual assessment of test results without systematic methodologies.

Evaluation Research

- **Description:** Conducts rigorous testing in real-world settings, addressing meaningful research questions. These studies engage real users or practitioners or evaluate with practical applications, such as industrial software or widely used open-source projects, and use datasets derived from real-world scenarios.
- **Examples:** PS31 [81] conducted an empirical study on VR automated testing in open-source VR projects, revealing gaps in current practices.

Solution Proposal

- **Description:** Proposes innovative approaches to XR testing challenges, focusing on theoretical benefits with minimal empirical evidence. These studies typically use basic examples and lack experimental validation with real-world applications.
- **Examples:** PS3 [97] presents an autonomous agent-based testing framework for XR systems. The study details the architecture and potential applications, but without experimental assessment.

Solution Proposal and Validation Research

- **Description:** Combines proposing novel solutions with preliminary validation, typically in simplified experimental settings.
- **Examples:** PS23 [82] presents a technique for detecting object misplacement issues in AR apps. It is validated using Unity-provided examples rather than real-world apps.

Solution Proposal and Evaluation Research

- **Description:** Proposes novel solutions and rigorously evaluates them in real-world contexts.
- **Examples:** PS19 [12] introduces a technique to detect stereoscopic visual inconsistencies in VR apps, validated using screenshots from real-world VR apps available on the Steam store.

Philosophical Papers

- **Description:** Focuses on theoretical perspectives or conceptual frameworks rather than implementing technical solutions. These studies aim to propose new ways of thinking about challenges, without presenting concrete implementations.
- **Examples:** PS13 [98] outlines general guidelines for detecting and preventing cybersecurity attacks in MR environments, identifying threats and countermeasures but offering no implementations.

Overall, 25 out of 34 studies (74%) include a solution proposal component. This count combines studies classified as pure solution proposals, solution proposals with validation, and solution proposals with evaluation. Among these, 15 studies, representing 60% of the solution-proposal studies, include only preliminary validation rather than real-world evaluation.

The first empirical evaluation study was published in 2019 (PS26). More recently, 2023 saw the introduction of two novel testing solutions evaluated in real-world contexts (PS16 and PS32). This trend suggests increasing potential for applying XR testing techniques in practical, real-world environments in the near future.

Figure 3.6 presents a bubble plot illustrating the distribution of studies across research topics and types. The majority of studies in each topic propose novel solutions and validate them. For example, in the automated testing topic, 3 of the 7 studies are classified as solution proposal + validation research. In usability testing, 4 of the 6 studies fall into the same category. These examples show how the bubble plot encodes both topic frequency and research maturity. These findings highlight the positive contributions of the XR software testing community, showcasing high-quality research that introduces innovative techniques with empirical validation.

Immersive Technology

We analyse the primary studies by examining the specific immersive technologies targeted for testing in these studies. Figure 3.7 summarises the technologies examined in the primary studies. About 59% of the studies focus on testing VR apps, while approximately 26% target AR apps. Only a small number of

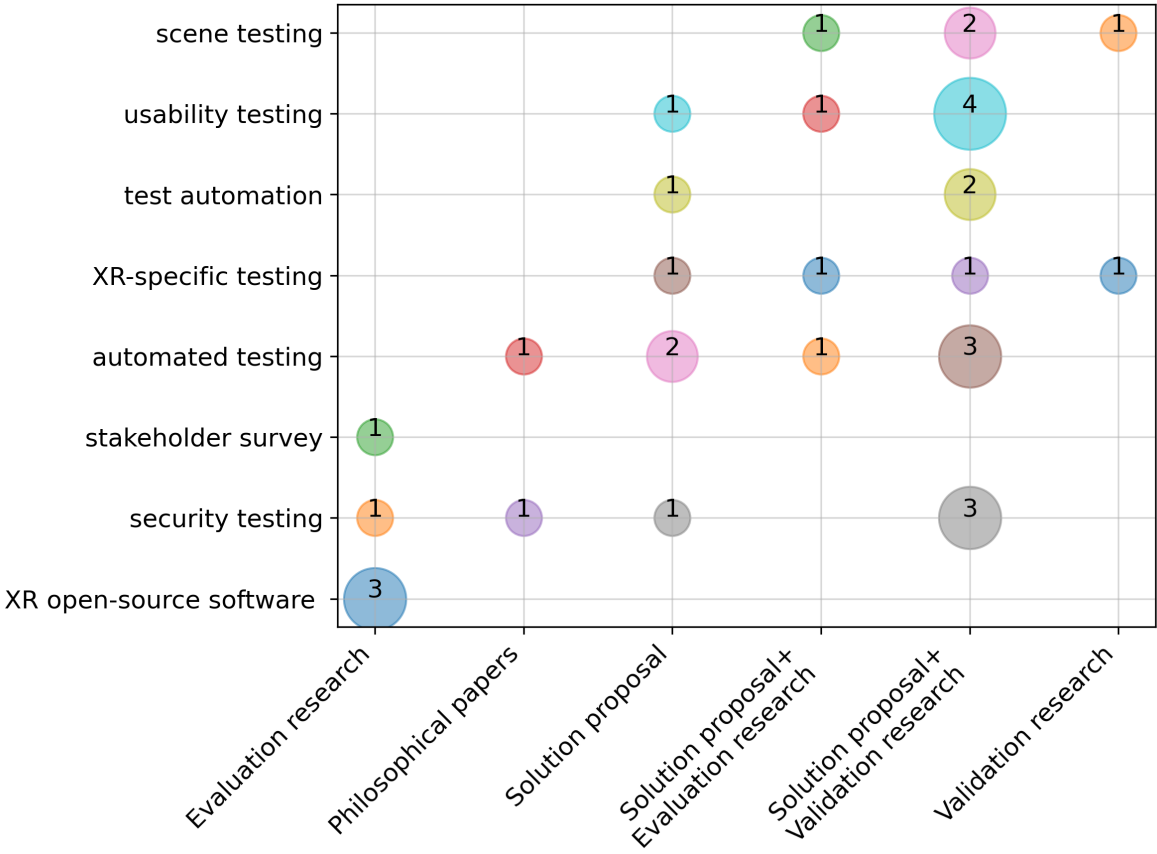


Figure 3.6: Bubble plot for research topics against research type

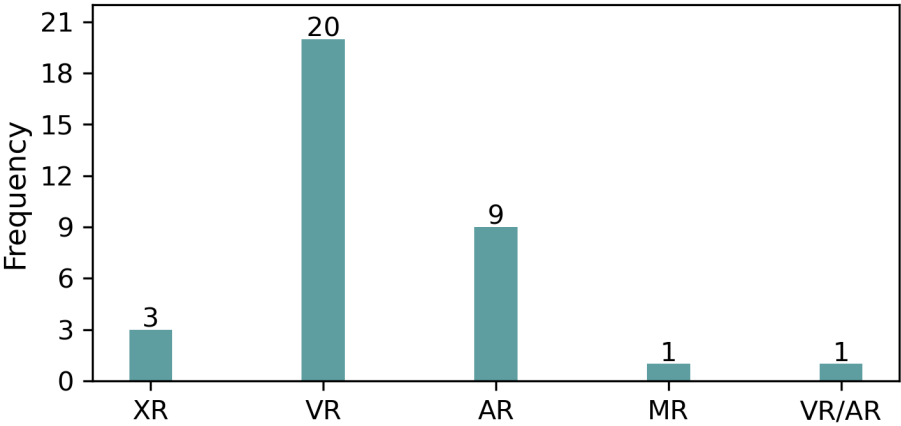


Figure 3.7: Immersive Technologies of Primary Studies

studies explore broader or integrated scopes: three use XR without limiting their scope to one specific technology, one investigates MR testing, and one explicitly addresses both VR and AR testing. We preserve this distinction because the categories reflect the terminology and stated scope of the primary studies. This also means that some category boundaries may reflect inconsistent author terminology rather than strict technical differences between XR, AR, MR, and VR.

The findings indicate that while VR and AR testing have received significant research attention, studies addressing broader scopes may still in the early stages of development within the research landscape.

The answer to RQ1.1, i.e., the current status of XR application testing research, is as follows:

Publication trends: Research on XR software testing has grown steadily, increasing from 2 publications in 2017 to 10 in 2023.

Publication venues: Conferences and journals are the main publication venues, representing 38% and 26% of the studies, respectively.

Research topics: Automated testing is the most prevalent research topic, accounting for 21% of studies, followed by usability testing and security testing, each contributing 18%.

Research types: 74% of studies propose novel XR testing solutions, with 60% relying on preliminary validations in controlled settings.

Immersive technologies: VR dominates with 59% of studies and AR represents 26%. XR, MR, and cross-technology research contribute 15%.

In summary, XR software testing is an emerging field, steadily gaining momentum.

3.3.2 RQ2: Testing Facets

We classify the primary studies based on three key test facets: *test activities* (e.g., test generation, test execution), *test concerns* (including objectives like functionality and security, and targets such as user interaction and collision), and *test techniques* (e.g., random testing, model-based testing). To ensure the meaningfulness of the extracted information, we exclude the studies that do not directly yield testing facets, which are five studies identified as empirical studies (PS2, PS6, PS28, PS29, PS31) and two classified as philosophical papers (PS13 and PS24). The remaining 27 studies are analysed to address this research question.

RQ2.1: Test Activities

Figure 3.8 visualises the distribution of test activities in the primary studies using a word cloud. The size of each keyword in the word cloud corresponds to its frequency, with larger keywords appearing more frequently in the test activities.

For instance, “test” is the most prominent keyword, reflecting its centrality across various activities. Among these, “generation” and “automation” are the next most prominent keywords, indicating that activities such as test generation and test automation are the most frequently addressed activities in the studies.

11 studies include the keywords “test” and “automation”, and all are associated with the test activity



Figure 3.8: Word cloud based on the test activities of the primary studies

test automation. On the other hand, eight studies include the keywords “test” and “generation”, among them, three involve the activity *test generation*, and five involve *test input generation*. The distinctions between the three activities are:

- **Test automation** only automates the execution of tests and does not include generating test data or oracles;
- **Test input generation** involves creating test input data, which can be done either manually or automatically, but it does not include test oracle generation;
- **Test generation** automates the creation of both test inputs and test oracles.

Among these activities, *test generation* is the least explored, appearing in only three studies, indicating its higher technical challenges compared to the other test activities.

Other recurring activities include *oracle prediction*, *test execution*, and *attack detection*. These findings highlight a clear focus on minimising manual effort and enabling scalable testing for XR software.

RQ2.2 Test Concerns

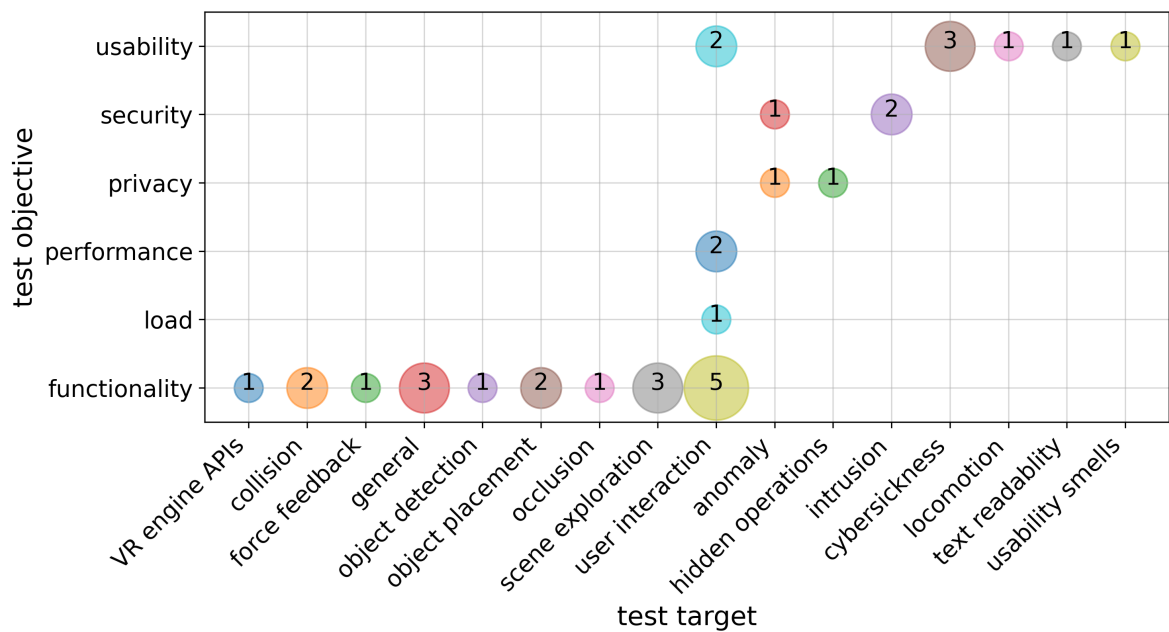
Test concerns cover both *test objectives* (e.g., functionality, usability, security) and *test targets* (e.g., cybersickness, collision) of testing.

Test Objective We categorise the primary studies into six groups based on their test objectives:

- **Functionality:** testing whether the functional specifications are correctly implemented,
- **Usability:** assessing whether SUT negatively impact user experience,
- **Security:** ensuring the SUT is protected from external attacks,
- **Privacy:** verifying that the user’s personal data is safeguarded against local threats,

Table 3.6: Test objectives in the primary studies

Test objective	Primary Studies	Number
Functionality	PS3, PS4, PS5, PS7, PS8, PS9, PS12, PS16, PS17, PS22, PS23, PS25, PS26, PS27, PS32, PS33, PS34	17
Usability	PS1, PS7, PS8, PS10, PS11, PS19, PS20, PS21	8
Security	PS14, PS15, PS30	3
Privacy	PS15, PS18	2
Performance	PS7, PS8	2
Load	PS8	1

**Figure 3.9:** Bubble chart of test objectives against test targets

- **Performance:** checking whether the SUT meets specific performance requirements (e.g., response time), and
- **Load:** evaluating the SUT's behaviour, reliability, or stability under stress.

Table 3.6 provides an overview of the test objectives. Among these, *functionality* is the most common objective in the primary studies. It is worth noting that individual studies can cover multiple test objectives. For instance, PS7 [99] addresses *functionality*, *usability*, and *performance*. It proposes a testing framework for system testing of mobile AR apps. The framework provides features such as collecting usability information, including the quality of the user experience in AR scenes; monitoring performance metrics, such as **Frames Per Second (FPS)** traces, to identify performance dips; and detecting functional edge cases through long-term monitoring.

Table 3.7: Test levels in the primary studies

Test Level	Primary Studies	Number
System Testing	PS1, PS3, PS4, PS5, PS7, PS8, PS9, PS10, PS11, PS12, PS17, PS18, PS19, PS20, PS21, PS22, PS23, PS25, PS26, PS27, PS32, PS33	22
Integration Testing	PS26	1
Unit Testing	PS9, PS26, PS34	3
Non-functional Testing	PS14, PS15, PS30	3
Not Applicable	PS16	1

Test Target To illustrate the relationship between test objectives and their associated test targets, we present a bubble chart highlighting how specific test targets align with certain test objectives. Importantly, a single study can address multiple test targets under a single test objective. For example, PS17 focuses on the test objective *functionality* and includes the test targets *collision* and *occlusion*.

Figure 3.9 shows the bubble chart. Among the test objectives, *functionality* is the most comprehensive, covering nine distinct test targets. The most studied target under this objective is *user interaction*, which focuses on testing interaction features in XR systems and has been examined in five studies. Other notable targets include *general*, which are the general guidelines for testing XR systems, and *scene exploration*, where testing focuses on exploring the XR scenes, each represented by three studies.

The second most prevalent test objective, *usability*, encompasses five test targets. Among these, *cybersickness* is the most studied, appearing in three studies, while *user interaction* is addressed in two studies. These findings align with the critical importance of user experience and the operational and interactional aspects of XR systems, emphasising the primary focus on XR software testing efforts.

The other test objectives, i.e., *security*, *privacy*, *performance*, and *load*, cover fewer test targets. Both *security* and *privacy* are linked to two test targets each, while *performance* and *load* are associated with only one test target each.

This analysis underscores the diversity of test targets within each objective and highlights relatively well-explored areas versus those requiring further investigation.

RQ2.3 Test Techniques

We address this sub-question by analysing the test level (e.g., system testing, unit testing), test type (i.e., black-box or white-box testing) and the specific techniques employed (e.g., random testing, search-based testing).

Test Level A single study can address multiple test levels. For instance, PS9 introduces preliminary solutions for both unit testing and system-level interaction in VR systems.

Table 3.7 provides an overview of the test levels explored in the primary studies. *System testing* is the most prominent test level, featured in 22 studies. In contrast, *unit testing* and *non-functional testing* each appear in only three studies, while *integration testing* is represented by a single study.

Test levels do not apply to PS16. The study proposes a technique for recommending potential test types (i.e., animation, colliding, and general) for VR systems but does not explicitly perform any testing

Table 3.8: Test types in the primary studies

Test Type	Primary Studies	Number
Black-box Testing	PS1, PS11, PS12, PS17, PS18, PS19, PS20, PS21, PS22, PS23, PS25, PS26	13
White-box Testing	PS3, PS4, PS7, PS9, PS10, PS26, PS27, PS32, PS33, PS34	10
Unclear	PS5, PS8	2
Not Applicable	PS14, PS15, PS16, PS30	4

Table 3.9: Test techniques in the primary studies

Test Technique	Primary Studies	Number
Static Analysis	PS4	1
Dynamic Analysis	PS4	1
Statistical Analysis	PS15	1
Image Processing	PS1	1
Machine Learning	PS11, PS14, PS15, PS17, PS19, PS21, PS23, PS27, PS30	9
Model-based Testing	PS3, PS4, PS5, PS10	4
Search-based Testing	PS32	1
Mutation Testing	PS25	1
Metamorphic Testing	PS17	1
Random Search	PS33	1
Greedy Search	PS33	1
Record and Playback	PS25	1
Runtime Monitoring	PS18	1
Equivalence Partition	PS12	1

actions. Therefore, PS16 is excluded from the data extraction for both *test level* and *test type*.

Test Type We define *white-box testing* as testing that requires access to the source code of the SUT, such as instrumentation or static analysis. In contrast, *black-box testing* does not involve internal information about the SUT. Instead, it relies on analysing the system’s input and output behaviour, such as evaluating screenshots or video recordings of specific actions in the XR systems. Notably, a single study may include both black-box and white-box testing methods.

In our analysis, we excluded three studies categorised under the *non-functional testing* level (discussed in the previous paragraph) from the extraction of test types. This is because black-box and white-box testing typically focus on verifying the *functionality* of the SUT.

Table 3.8 presents a summary of the findings. The distribution studies utilising *black-box testing* and *white-box testing* are relatively balanced, with 13 and 10 studies, respectively. In addition to the four excluded studies, two studies lack sufficient evidence to determine their test type and are therefore categorised as *unclear*.

Table 3.10: Evaluation metrics in the primary studies

Metrics	Primary Studies	Number
Standard ML Metrics	PS11, PS14, PS17, PS18, PS19, PS23, PS27, PS30	8
Classification Results	PS15	1
Manual Validation	PS10, PS12, PS16, PS21	4
Method Coverage	PS32	1
Model Coverage	PS4	1
Object Coverage	PS32, PS33	2
Requirement Coverage	PS5	1
Mutation Score	PS5, PS25	2
SSQ Score	PS21	1
Detected Bugs	PS32	1
Object Detection Success	PS22	1
Suspiciousness Score	PS15	1

Test Techniques Table 3.9 summarises the test techniques utilised in the primary studies. A few studies that do not employ specific testing techniques but instead provide general testing guidelines are excluded from the table. Additionally, a single study may apply multiple test techniques. In total, 14 distinct techniques are identified, with two being adopted by multiple studies: *machine learning-based testing*, used in nine studies, and *model-based testing*, applied in four studies. These findings align with the characteristics of XR systems. Machine learning techniques are particularly well-suited for handling the rich graphical interfaces of XR systems, such as identifying faults using app screenshots. Meanwhile, model-based testing simplifies the inherent complexity of XR systems by abstracting them into models, facilitating systematic testing.

To further investigate the most predominant test technique, machine learning-based testing, we examine the dataset used in these studies to train the ML models for testing in Section 3.4.2.

Although the remaining test techniques are each represented by only one study, this diversity highlights the successful exploration and adoption of various innovative approaches in the emerging field of XR software testing. Future research could explore the underrepresented areas, like integration testing and expand the application of emerging test techniques.

The answer to RQ1.2, i.e., the test facets involved in XR applications, is as follows:

Test activities: The most common test activities involve test automation (11 studies) and test generation (8 studies), reflecting a focus on reducing manual effort in testing XR software.

Test concerns: The primary test objectives are functionality (17 studies) and usability (8 studies). The most frequent test targets for functionality are user interaction (5 studies), while for usability, the key focus is cybersickness (3 studies). These align with critical user experience and interactional aspects of XR systems.

Test techniques: System testing is the dominant test level (22 studies), with black-box and white-box testing nearly balanced. Machine learning is the most prevalent technique (9 studies), followed by model-based testing (4 studies).

Table 3.11: Evaluation environments in the primary studies

Evaluation Environment	Primary Studies	Number
HMD	PS10, PS12, PS14, PS21, PS26, PS30	6
Unity Editor	PS16, PS17, PS23, PS26, PS32, PS33	6
Mobile Device	PS18, PS22, PS26, PS27	4
Haptic Device	PS25	1
Cloud	PS15	1
Unclear	PS4, PS5, PS7, PS11, PS34	5

3.3.3 RQ3: Evaluation

We address RQ3 by presenting the evaluation metrics reported in the studies and the evaluation environment used to assess the testing techniques.

Several studies lack concrete evaluation, as they do not fall under the research types of *evaluation research* or *validation research* (Section 3.3.1). Specifically, six solution proposals, five empirical studies and two philosophical papers are excluded from the extraction of evaluation metrics and environments.

Metrics

Table 3.10 summarises the evaluation metrics used in the primary studies. As discussed in Section 3.3.2, ML-based testing is the most prevalent technique, represented by nine studies. Among these, seven studies used *standard ML metrics*, such as *precision*, *recall*, and *F1-score*, to evaluate ML performance. One study relies on *classification results*, which provide general information on the number of correctly classified cases.

The studies identify four kinds of coverage metrics:

- **Method coverage:** measures the percentage of methods exercised by tests out of the total number of methods;
- **Model coverage:** calculates the proportion of states covered by tests in a model as used in model-based testing, such as a finite state machine;
- **Requirement coverage:** computes the percentage of nodes covered in a requirement flow graph, which is derived from an XR app’s scene graph;
- **Object coverage:** assesses the percentage of interactable objects triggered by tests.

Coverage metrics are utilised in four studies, with one covering both method and object coverage. Similarly, *manual validation*, where results are manually verified, is also used in four studies.

However, the reliance on manual validation suggests the need for more systematic and automated evaluation frameworks, especially as XR systems grow in complexity.

Evaluation Environment

To explore what types of environments are involved in the evaluation, such as simulation, mobile devices, and HMDs, we investigate the evaluation environments within the studies. Table 3.11 provides the results of evaluation environments. The most common evaluation environments are *HMD* (head-mounted

display) and *Unity Editor*, each represented in six studies. For clarity, we define the environment *Unity Editor* as the simulation performed within the Unity’s *Scene* or *Game* view⁸. *Mobile device* is the second most common environment, used in four studies, while *haptic device* and *cloud* are the least common, each appearing in one study.

Besides the studies excluded from the extraction of evaluation metrics and environments due to their research types or empirical nature, five studies lack enough details to determine the evaluation environment and are therefore labelled as *unclear*.

The focus on HMDs, Unity Editor, and mobile devices underscores their critical role in real-world testing, while the limited diversity in environments such as cloud-based or haptic devices suggests opportunities for further exploration and innovation.

Furthermore, we notice some studies utilise existing datasets for evaluation. The details of these datasets (content types, sizes and availability) are discussed in Section 3.4.2.

The answer to RQ1.3, i.e., the extent of the testing approaches validated, is as follows:

Out of 27 studies involving valid test activities, six studies do not provide any evidence on evaluation, leaving 78% of the studies validated through some form of evaluation.

Metric: The most common evaluation metrics are standard machine learning metrics for evaluating machine learning-based techniques. Additionally, manual validation and different types of coverage metrics are equally prevalent.

Environment: The most frequently used evaluation environments are HMD, Unity Editor, and mobile device, reflecting the typical platforms for XR application development and testing.

3.4 Discussion

In this section, we discuss the findings and implications of this mapping study. Specifically, we address

- The key insights and lessons learned from our methodology;
- The datasets and tools utilised or proposed in the primary studies;
- The implications for practitioners; and
- The remaining challenges and future research directions identified through our analysis.

3.4.1 Mapping Study Methodology

While conducting this mapping study, we carefully considered methodological choices that could influence our findings. Our approach embraces the diverse nature of XR testing research while acknowledging its potential impacts on interpretation.

While differences between research types or publication venues may yield varying depths of evidence, this diversity enhances the value of our mapping study. By capturing the full spectrum of XR testing research, we provide a more accurate representation of the field’s current state.

⁸<https://docs.unity3d.com/Manual/UsingTheEditor.html>

Table 3.12: Training datasets used for machine learning-based testing approaches

Study	Content	Size	Source	Avail.
PS11	Image	600	experiments	F
PS19	Image	20 000	Steam	F
PS21	Video	61	UCSD Ped1 & Ped2, Avenue, KITTI	T
PS23	Image	720	Unity Mars	T
PS27	Image	~ 2740	Google Play & GitHub	F
PS30	Traffic & attacks	~ 848 000	CIC-IDS2017	T

Following the guidelines by Petersen et al. [62], we deliberately chose an inclusive approach without applying quality assessments during selection. We acknowledge this introduces certain limitations as analysing heterogeneous studies collectively may obscure category-specific characteristics. Despite potential influences on the interpretation of trends, we believe the benefits of comprehensive coverage outweigh these limitations for mapping the emerging research area of XR testing.

3.4.2 Datasets and Tools

To facilitate future research and practices in XR testing, we present an in-depth investigation of the datasets and tools identified in our primary studies.

This subsection examines the following:

- Datasets used for training ML models in ML-based testing techniques,
- Datasets for evaluating testing techniques,
- Industrial tools employed or referenced in the studies, and
- Research tools used or proposed within the studies.

The availability of the datasets and tools is determined as of the submission date of this mapping study (December 2024). The resources are organised and can be accessed at <https://sites.google.com/view/xr-testing>.

Datasets for Training

As discussed in Section 3.3.2, nine primary studies (PS11, PS14, PS15, PS17, PS19, PS21, PS23, PS27, PS30) utilised ML-based techniques for testing XR apps. To better understand their capabilities and provide valuable resources for future research and practice, we analyse the datasets used for training the ML-based techniques. Among the nine studies, six provide detailed dataset information. We examine their content type, training set size (excluding test sets), data source, and availability, and summarise our findings in Table 3.12.

We provide a brief overview of each dataset and its specific task for testing XR applications, referring to each dataset as “Dataset PSx” based on the corresponding primary study.

Dataset PS11 consists of 600 images of XR scenes, containing some texts in their background. Each image is labelled whether the text is readable or not by human participants and features various configurations of font styles and background textures. The dataset is not publicly available.

Dataset PS19 is a subset of 20 000 stereoscopic screenshots, randomly sampled from an original training set of 154 566 screenshots, collected from 288 VR apps on Steam⁹. Steam is one of the largest platforms for video games, including VR apps.

Dataset PS21 is based on multiple datasets, comprising a total of 61 video clips, each containing 200 frames, to train a model for measuring exceptional motion in VR video content that contributes to cybersickness. The original datasets, UCSD Ped1 and Ped2 [100], Avenue datasets [101], and KITTI benchmark datasets [102], are all publicly available.

Dataset PS23 consists of 720 screenshots from a basic AR scene provided by Unity Mars¹⁰, a Unity extension for AR/MR content development. The dataset is labelled via crowdsourcing to identify realistic object placement. It is used to train a model to identify object misplacement issues in AR systems, capturing variations in placement gaps, distance, and viewing angles.

Dataset PS27 includes 3043 screenshots from 21 AR apps sourced from the Google Play Store and GitHub. With 90% (approximately 2740 screenshots) allocated for training a model to detect object misplacement issues in AR systems. The dataset is labelled via crowdsourcing to provide placement information. However, the exact numbers of screenshots in the training and testing subsets are not specified in the paper, and the dataset is currently not publicly accessible.

Dataset PS30 utilised the Intrusion Detection Evaluation Dataset (CIC-IDS2017)¹¹, containing over 2.8 million network traffic instances, including normal traffic and attacks like DoS and DDoS. Reformatted for binary classification (attack vs benign), it comprises 1 211 327 instances, and 70% of them are used for training. Because the paper reports inconsistent sample sizes across subsets, we recalculated the training set size for consistency as 70% of 1 211 327, which is approximately 848 000.

PS14 and PS15 do not provide details about the datasets used. PS17 employs a deep reinforcement learning technique to generate test data, in which data are generated through the exploration and perception of an autonomous agent within a virtual environment.

As ML-based testing techniques often evaluate their performance using the same dataset from which their training subset is derived, these test sets are not discussed in Section 3.4.2.

Overall, the prevalence of image-based training datasets highlights the potential of image-based techniques to address a wide range of software testing tasks for XR applications effectively.

Datasets for Evaluation

This section focuses on evaluation datasets, potentially encompassing diverse data points or scenarios, offering broader applicability for testing methodologies, empirical studies, and potential reuse in future research. Isolated research prototypes or limited open-source applications are not considered comprehensive datasets.

As discussed in Section 3.4.2, ML-based techniques often evaluate their performance using test datasets, i.e., subsets derived from the same datasets as their training data. Detailed information about these evaluation sets is omitted to avoid redundancy, as they may only differ from the training sets in size. Apart from these, most studies utilised research prototypes or basic open-source applications.

Two empirical studies, PS2 and PS6, present independent datasets for evaluation. **Dataset PS2** consists of 390 mobile AR apps from the Google Play Store to conduct an empirical study on user

⁹<https://store.steampowered.com/>

¹⁰<https://unity.com/products/unity-mars>

¹¹<https://www.unb.ca/cic/datasets/ids-2017.html>

Table 3.13: Industrial tools in the primary studies. OSS indicates if the tool is open-source or not.

Name	Platform	Input	Test Type	OSS
UTF	Unity	Test scripts	Unit	T
XRI	Unity	Interaction designs	N/A	T
Airtest	Unity, Cocos ¹³	Test scripts	Scene	T
AltUnity Tester	Unity, Unreal	Test scripts	Scene	F
ML-Agents	Unity	Training env.	Scene	T
clumsy	Windows	N/A	Network	T
Wireshark	Windows, Linux, macOS	N/A	Network	T

privacy concerns in mobile AR apps. However, this dataset is not available. **Dataset PS6** collects 368 real bugs from open-source WebXR projects, labelled with their bug symptoms and root causes and is publicly accessible.

We note that multiple studies (PS16, PS17, PS31, PS32, PS33) utilised a dataset called **Unity List**, which is no longer accessible¹².

Industrial Tools

Table 3.13 highlights industrial tools used or referenced in the primary studies. These tools address various testing needs, including GUI, unit, and network testing, as well as one tool for XR interaction development. For each tool, we outline key details such as supported platforms and engines, input formats, test types, and whether the tool is open-source. This information is intended to guide researchers and practitioners in selecting tools suitable for their testing requirements.

Unity Test Framework (UTF)¹⁴ is an official testing tool provided by Unity for unit testing Unity-based projects. It integrates with NUnit¹⁵, a unit testing library for .NET languages.

XR Interaction Toolkit (XRI)¹⁶ is an official Unity package for creating 3D and UI interactions in VR/AR experiences. While it does not directly facilitate XR app testing, it is useful for prototyping research apps that can serve as experimental platforms for testing methodologies.

Airtest¹⁷ is a visual-based UI test automation framework commonly used for video game testing. It uses screenshot-based locators in test scripts to simulate user actions, making it suitable for dynamic and visually complex interfaces.

AltUnity Tester¹⁸ is a test automation framework designed for games and 3D apps, supporting UI and functional testing. Test scripts interact with Unity elements using identifiers such as object names and tags, simulating user actions.

ML-Agents¹⁹ is an open-source toolkit by Unity for training intelligent agents in Unity-based 2D, 3D, and VR/AR environments using various AI methods. It provides Python APIs for training and Unity C# scripts for environment simulation. With over 17 example Unity environments, it is well-suited for evaluating XR testing approaches, including agent-based testing [18].

¹²According to Unity List's X homepage <https://x.com/unitylist>, it is no longer available.

¹⁴<https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/index.html>

¹⁵<https://nunit.org/>

¹⁶<https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0>

¹⁷<https://airtest.netease.com/>

¹⁸<https://alttester.com/tools/>

¹⁹<https://github.com/Unity-Technologies/ml-agents>

Table 3.14: Research tools in the primary studies

Name	Source	Function	Platform	Avail.
iv4xr	PS3	Agent-based testing	N/A	T
ARCHIE	PS7	Usability testing	Unity	T
MAR-Security	PS18	Hidden operation detection	Android	T
StereoID	PS19	Cybersickness detection	N/A	F
PredART	PS23	Object misplacement prediction	Unity	T
VOPA	PS27	Object misplacement assessment	N/A	F
VRGuide	PS32	VR scene exploration	Unity	T
VRTest	PS33	VR scene exploration	Unity	T
AutoQuest	Herbold and Harms [103]	Usability smell detection	N/A	F
TESTAR	Vos et al. [104]	Scriptless GUI testing	mobile, desktop, web	T

Clumsy²⁰ and **Wireshark**²¹ are tools for network simulation and analysis. Both were used in PS15 to simulate network- and application-based attacks. These tools are applicable to networked applications, including XR clients and servers, enabling the evaluation of resilience and performance under adverse network conditions.

Research Tools

This section examines research tools specifically designed for XR testing, excluding general tools for tasks like data analysis. We assess each tool's source (primary studies or external references), key functionalities, supported platforms, and availability. This analysis is based on publicly available versions, focusing on implementations rather than techniques reported in the papers. While we did not run the tools, we thoroughly reviewed their documentation and repositories. Table 3.14 lists the tools analysed.

iv4XR²² is a suite of tools for automated testing for XR applications. It includes frameworks for agent-, model-, and reinforcement learning-based testing, as well as user experience testing.

ARCHIE²³ is a Unity Editor plugin for usability testing in mobile and wearable AR apps. The repository includes Unity-based examples and supports cloud functions.

MAR-Security²⁴ is a framework for preventing hidden operations in mobile AR apps. Its repository includes an Android project implementing the detection mechanism and scripts for collecting runtime data from Android devices.

StereoID²⁵ is a tool for detecting stereoscopic visual inconsistencies linked to cybersickness. However, the tool is not currently accessible.

PredART²⁶ includes two types of scripts: a C# camera control script for Unity projects, and scripts for machine learning model implementation and training.

VOPA²⁷, is a tool designed to assess virtual object misplacement. However, it is not currently

²⁰<https://jagt.github.io/clumsy/>

²¹<https://www.wireshark.org/>

²²<https://github.com/iv4xr-project>

²³<https://github.com/lehmansarahm/ARCHIE>

²⁴<https://github.com/lehmansarahm/MAR-Security>

²⁵<https://sites.google.com/view/stereoid>

²⁶<https://sites.google.com/view/predart2022>

²⁷<https://sites.google.com/view/vopa-for-artesting/home>

publicly accessible.

VRGuide²⁸ and **VRTest**²⁹ are automated VR testing tools for scene exploration. While each tool employs different exploration strategies, both provide Unity scripts for their implementations.

AutoQUEST [103] detects usability smells, i.e., recurring interaction or interface patterns that may indicate usability problems, by analysing recorded user data. While its website³⁰ is accessible, only compiled Java (.jar) files are available rather than source code.

TESTAR³¹ [104] is an open-source tool for scriptless automated testing of desktop, web and mobile apps at the GUI level. The repository includes documentation for setup and execution. PS24 [105] references it as a tool that can be extended for scriptless testing in XR environments.

3.4.3 Implications for Practitioners

Based on our analysis of datasets and tools referenced in the primary studies, our mapping study reveals two useful insights for XR practitioners.

First, regarding tool selection guidance, Table 3.13 provides a curated selection of industrial tools organised by platform and testing task. While our findings are based on the primary studies selected for this mapping study, we acknowledge that additional options like Meta XR Simulator (discussed in Section 2.4) may also be valuable for certain testing scenarios.

Second, concerning research-to-practice opportunities, Table 3.14 highlights the research tools that address gaps in current industrial offerings. Though these may require additional implementation effort, they provide cutting-edge capabilities for organisations with specialised testing needs or those seeking competitive advantages in XR quality assurance.

3.4.4 Challenges and Future Research Directions

This section explores the open issues and potential future research directions based on the findings of this mapping study.

During the study selection process (Section 3.2.2), some studies were excluded because they did not directly align with the focus on testing-related research. However, these studies address challenges that could inspire novel testing approaches by being adapted to specific XR testing needs. By integrating insights from these excluded studies with the findings from our mapping study, we aim to present meaningful and actionable future research directions to advance XR software testing.

Interaction Formalisation

As discussed in Section 3.3.2, *user interaction* is the most common testing objective for functional testing, indicating the importance of interaction testing in XR apps. In Section 3.3.1, we classified *scene testing* studies that validate XR functionality through interactions with virtual objects and scene navigation. However, these approaches provide limited context on the specific interactions required to trigger objects (e.g., touching) or complete navigation tasks (e.g., reaching a destination).

²⁸<https://sites.google.com/view/vrguide2023>

²⁹<https://sites.google.com/view/vrtest2021>

³⁰<https://autoquest.informatik.uni-goettingen.de/trac/wiki>

³¹<https://testar.org/>, https://github.com/TESTARtool/TESTAR_dev

Unlike 2D GUI apps, where interaction types are relatively straightforward, XR apps' 6DoF nature demands more diverse interaction types. Moreover, XR interaction methods may vary based on the deployment platform and device capabilities.

Drawing from prior research, formal gesture descriptions have proven effective in automating UI testing for mobile apps [106] and could similarly benefit XR apps. However, this requires a predefined set of XR-specific interaction types, which remains an open challenge [63]. Leveraging these predefined interactions could support cross-device compatibility testing and facilitate the development of reusable testing frameworks for diverse XR platforms.

We recommend systematic empirical studies to categorise XR-specific interactions (e.g., gestures, haptic feedback) by thoroughly analysing documentation from XR development platforms and open-source projects. This methodological approach supports the development of standardised taxonomies for XR interaction, facilitating better understanding and comparison across systems.

In the following chapters (Chapters 4 and 5), we adopt empirical techniques to investigate XR interactions, yielding a taxonomy of XR interactions and an open-source project analysis. These results form a foundational basis for the testing methodologies introduced later in the thesis, ensuring that the testing approaches are grounded in a structured understanding of real-world XR interaction patterns.

Test Oracle Automation

In Section 3.3.2, we identified *test automation*, *test input generation*, and *test generation* as the most frequent test activities for XR apps. Among these, *test generation*, which involves generating both test inputs and oracles, remains the least explored. Non-crashing functional bugs often require manual validation, with current approaches focusing primarily on crash bugs due to the lack of automated oracles [107]. Automating oracles is crucial for overcoming this bottleneck and advancing automated testing [32].

While some research has addressed the oracle problem for XR apps, the specific oracles needed to validate functionality remain unclear and vary by system [105]. For example, detecting collision and object misplacement may require distinct oracles, each demanding tailored techniques. Addressing this gap necessitates a deeper understanding of the problem and the development of novel solutions.

To advance the state of knowledge, we suggest conducting the following studies:

- Investigating which characteristics of XR software can serve as reliable test oracles, and
- Determining the most effective oracle types (e.g., assertions, contracts, or metamorphic relationships [37]) for different XR testing scenarios.

In Chapter 5, this thesis contributes to this line of work by designing an oracle tailored to detecting non-exceptional issues in XR scenes. The oracle is based on contracts (i.e., postconditions), leveraging the properties of objects defined within those scenes to identify subtle but meaningful behavioural problems.

XR-specific Testing

XR-specific requirements encompass a wide range of test targets, including real-time collision and occlusion, as well as key AR features such as tracking and registration [14]. Section 3.3.1 identifies *XR-specific testing* as a primary research focus in XR software testing.

Additionally, studies identified during the selection process provide insights into testing these requirements. For example, several studies [108–112] propose effective collision testing techniques. However, these studies mainly focus on experimental simulations and have not been applied to specific XR apps. Their methodologies could be adapted to enable systematic collision testing in XR apps, such as instrumenting specific objectives in an XR app to yield collision information.

The unique nature of XR-specific requirements calls for novel testing methodologies not present in other software domains, underscoring the need for tailored approaches and further research. For effective testing, we recommend first conducting systematic studies to analyse the software manifestation of XR-specific features. This analysis should identify observable behaviours in XR apps and determine which software testing techniques would be most effective for validating these unique characteristics. Such foundational work is essential before developing specialised XR testing methodologies.

Software-centric Usability Testing

Cybersickness is the most common usability issue in XR apps, with several studies proposing software-centric techniques for automated detection, as detailed in Section 3.3.2. Additionally, many user-centric studies explore the nature of cybersickness (Section 3.1.1), providing a foundation for developing software-centric detection methods.

Beyond cybersickness, we identified usability-focused user studies during the study selection process that could inform automated testing techniques from a software-centric perspective. For example, Kia et al. [113] highlighted factors affecting physical effort during AR app interactions, such as target size and interaction error rates. Such factors could be translated into software-level indicators, for example by detecting small targets or interaction designs likely to cause repeated failed attempts.

To bridge the gap between software- and user-centric approaches, we recommend integrating findings from user studies into automated testing frameworks. This integration would enable the detection of common usability issues without requiring human evaluation, making usability testing more scalable and consistent across XR apps.

AI for XR Testing

Advancements in AI, particularly large language models (LLMs) and reinforcement learning techniques, present opportunities to enhance XR app testing.

As discussed in Section 3.4.4, test oracle automation remains a significant challenge in XR testing. While crowdsourcing has been shown to effectively address oracle-related tasks [82, 114], recent progress in LLMs offers a potential alternative for automating text-based tasks [115], which benefits the generation of human-readable assertions, validating expected outputs, and synthesising test expectations from natural language specifications.

The oracle problem in XR systems is complex due to their reliance on 3D graphics. However, multimodal LLMs, which process both textual and visual information, have demonstrated capabilities in understanding graphical content, ranging from 2D screenshots to 3D assets [116, 117]. These advancements could enable more robust testing of intricate graphics systems, including XR apps.

Furthermore, LLMs have been effectively used to generate unit tests for Unity-based game development [118]. Given the shared Unity platform, these techniques could potentially be adapted for XR app unit testing, further advancing automation in this domain.

In addition, deep reinforcement learning and imitation learning techniques have demonstrated capabilities to both play (complete specific tasks) and test (explore unknown scenarios) video games [59]. We suggest leveraging these techniques to tackle the interactive challenges of XR app testing.

3.5 Answer to ORQ1

This chapter provides an answer to ORQ1 from Section 1.1:

ORQ1: *How has XR testing been approached in existing research, and what key focuses, limitations, and research gaps have been identified?*

XR applications are currently tested through a small but growing body of research that focuses primarily on automated and system-level techniques, with an emphasis on functionality and user experience. Studies increasingly rely on test automation, test generation, and machine learning-based methods to reduce manual effort and manage the complexity of immersive systems.

3.5.1 How XR Applications are Currently Tested

XR testing research is still emerging but steadily growing, with publications increasing from 2 in 2017 to 10 in 2023, indicating rising interest but a relatively small body of work overall. Most studies focus on automated testing and system-level testing, often using black-box or white-box techniques and machine learning-based methods to generate or execute tests. Test activities emphasise test automation and test generation, aiming to reduce manual effort. In terms of development platforms for testing approaches, many are implemented and evaluated on common XR platforms such as HMDs, the Unity Editor, and mobile devices.

3.5.2 Focuses of Existing Approaches

There is a strong emphasis on automation: test automation and automated test generation are the most common activities, reflecting a clear drive to make XR testing more scalable and less labour-intensive. Functionality and usability are the primary test objectives, and user interaction and cybersickness appear as common test targets, showing that research is aligned with core user experience and interaction challenges in XR. Machine learning and model-based techniques are increasingly used, suggesting a move toward more sophisticated, data-driven and model-driven testing strategies.

3.5.3 Limitations and Weaknesses

The research community is still relatively small, with modest publication counts and concentration in a limited number of venues, which implies that many aspects of XR testing remain underexplored. Evaluation and validation are often preliminary. Although many studies include some form of assessment, they frequently rely on controlled, small-scale settings rather than large, diverse, or real-world deployments, limiting confidence in generalisability. Coverage of test concerns is narrow: functionality and usability dominate, while other important aspects such as security, robustness, accessibility, and long-term user comfort receive comparatively little attention.

3.5.4 Key Research Gaps

Technology coverage is uneven, where VR accounts for the majority of studies, with fewer works on AR, MR, and cross-technology XR systems, leaving gaps in understanding how to test heterogeneous or blended XR experiences. Test facets remain incomplete in the sense that there is limited existing work on richer oracles, realistic interaction modelling, and comprehensive coverage metrics that capture both code and complex spatial configurations. Evaluation practices need strengthening. More systematic and large-scale studies are needed, especially studies conducted with realistic XR applications, representative users or tasks, and practical deployment settings. Shared benchmarks, datasets, and tool comparisons are also needed to better assess effectiveness and practical impact.

3.6 Conclusion

This chapter presents the methodologies, results, and findings of a systematic mapping study on software testing for XR applications [13]. From an initial pool of 1167 studies retrieved from a digital library, we selected 34 relevant studies for in-depth analysis.

We classified these studies and extracted meaningful information to address key research questions regarding the current research status, test facets (including test activities, concerns, and techniques), and evaluation methodologies employed in XR testing. Additionally, we catalogued datasets and tools referenced in these studies, offering a valuable resource for researchers and practitioners to build upon and advance their work.

The mapping study identifies several open issues and outlines promising future research directions. Our findings highlight the growing importance of XR testing and provide a foundation for advancing methodologies to address its unique challenges. As XR technology rapidly evolves with new platforms, devices and applications, testing methodologies must not only adapt to support these innovations but also leverage the emerging capabilities they offer. Advanced features and hardware capabilities present both challenges and opportunities for testing. Future testing approaches will need to accommodate the increasing complexity of XR environments and the integration of AI-driven behaviours that characterise next-generation XR systems.

To address the future work identified by this mapping study, Chapter 4 investigates interaction formalisation for XR testing, and Chapter 5 presents an empirical analysis of open-source XR projects to categorise common XR interactions. By systematically mapping XR interactions to specific user actions, these chapters develop tools capable of generating user action sequences for executing XR testing tasks.

Chapter 4

Test Automation for XR User Interactions

The primary testing challenges for XR applications, identified in the previous chapter’s systematic mapping study, are centred around validating both functionality and user interactions. To address user interaction testing, we introduce INTENXION, a novel test automation framework designed for validating user interactions in Unity XR applications. Drawing on principles from established GUI testing frameworks like UI AUTOMATOR [119] and APPIUM [120], INTENXION adapts action simulation and outcome verification specifically to the spatial context of XR. At its core is a comprehensive taxonomy, which is developed from HCI research and industry guidelines. The taxonomy enables systematic mapping of high-level interaction goals to testable actions within automated scripts, and organises XR interactions into three layers: (1) **Core Tasks** representing user intents; (2) **Atomic Actions** that serve as fundamental building blocks for interactions; (3) **Compositional Patterns** that govern the combination of atomic actions into complex, meaningful interactions.

INTENXION enables hardware-free, fine-grained validation of diverse XR interactions beyond simple point-and-click UI elements. As discussed in Chapter 3.4, test oracles for XR applications remain an open challenge; to mitigate this, INTENXION incorporates a dedicated assertion mechanism to validate test outcomes. INTENXION features three main capabilities. First, it provides intuitive APIs for simulating complex interaction behaviours and spatial gestures. Second, it supplies convenient assertion primitives for checking expected interaction outcomes. Third, it natively supports the specification and execution of intricate compositional interaction patterns. Combined together, INTENXION allows developers to express rich XR scenarios as structured sequences of test actions and validations.

To demonstrate the practical value of INTENXION, we present a case study covering eight representative interactable object types, selected according to industrial XR guidelines and our taxonomy. For each, we developed targeted test scripts encompassing both activation of expected behaviours and outcome assertions. Our results demonstrate that INTENXION can cover the intended functionalities of all subject types.

In summary, this chapter contributes:

- **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 that INTENXION supports the construction of executable XR interaction sequences and corresponding outcome assertions for representative interaction patterns.

This chapter is structured as follows. Section 4.1 introduces the motivation for this and the subsequent chapter. Section 4.2 presents our taxonomy of XR interaction. Section 4.3 describes the design and implementation of INTENXION. Section 4.4 presents our case study and results. Section 4.5 discusses the implications of our work, while Section 4.6 revisits ORQ2 and ORQ3 from Section 1.1 and provides their answers. Finally, Section 4.7 summarises this chapter, highlights its main contributions, and explains how they connect to and motivate the work presented in the next chapter.

Further details, including source code, assets, and test scripts, are available at: <https://github.com/ruizhengu/IntenXion>. The content of this chapter is based on “Ruizhen Gu and José Miguel Rojas. A Test Automation Framework for User Interaction in Extended Reality Applications” published at *The Third International Workshop on Virtual and Augmented Reality Software Engineering (VARSE 2025)* [121].

4.1 Motivation

This section motivates the need for a new test automation approach (i.e., automated test execution based on predefined test scripts), and also the automated testing approach (i.e., automated test generation and execution) in Chapter 5. The discussion begins with existing research on XR scene testing, reviewing the current state-of-the-art and clarifying why these techniques are insufficient for modern XR software, which increasingly relies on realistic, spatial user interactions.

4.1.1 XR Scene Testing

As identified in the systematic mapping study of XR software testing in Chapter 3, *XR scene testing* is one of the most actively studied topics in the field. Scene testing focuses on exploring XR scenes through two core tasks: scene navigation and interaction event triggering, treating scenes as the primary abstraction for exercising application behaviour. Wang et al. pioneered automated VR scene testing with VRTEST [54] and VRGUIDE [55], which introduced the first dedicated tools for systematically exploring VR scenes. Despite their novel contributions, these approaches rely on and were evaluated using relatively simplified VR projects that *lack comprehensive 3D interaction mechanisms*. Specifically, they exclusively adopted basic UI-style event triggering, instead of the more realistic 3D interaction mechanisms enabled by VR input devices such as hand-held controllers.

Therefore, most prior work tests VR applications primarily through conventional UI interactions, leaving the validation of complex, spatial 3D user inputs largely unexplored. In this thesis, the focus is on enabling XR software testing that exercises such 3D user interactions via simulated controller actions, rather than limiting testing to traditional UI elements. Motivated by these gaps, this chapter

advances the state of the art by introducing a novel test automation approach for XR applications that supports realistic, controller-based interaction patterns.

4.1.2 From UI Events to Spatial 3D User Interactions

This subsection explains the technical details of VRTEST and VRGUIDE, and why they cannot address the testing needs of modern 3D XR apps. In addition, we clarify why they are not used as evaluation baselines in Chapter 5.

The key limitation lies in the user interaction paradigm each approach targets. VRTEST and VRGUIDE are implemented based on Unity’s `Event Trigger` mechanism¹. It is a component designed for handling events within the Unity UI framework. As a result, these tools only support interaction with game objects that function as traditional UI elements. To elaborate, these tools were designed for an earlier generation of VR apps where user interactions were limited to performing “point-and-click” style interaction. The 3D objects in the environments are fundamentally the same as UI buttons, and interaction reduces to dispatching discrete events rather than simulating continuous physical manipulation. The evaluation environments used in the VRTEST and VRGUIDE studies reflect this limited scope, comprising scenes with objects that function in simple on/off states that are exclusively triggered by such “point-and-click” interactions.

In contrast, the approaches developed in Chapters 4 and 5 target modern XR apps built around rich, spatial 3D interactions. These subjects are developed with comprehensive XR interaction frameworks like Unity’s XR Interaction Toolkit, where users interact via 6DoF controllers tracked in 3D space. Here, interaction involves grasping, manipulating, and moving virtual objects in a physically plausible manner, rather than merely “clicking” them. The XR apps in our case study subjects (Chapter 4) and all projects in the XRBENCH3D benchmark dataset (Chapter 5) all adopt this modern paradigm. Because VRTEST and VRGUIDE are inherently tied to UI-style event triggering and do not support these spatial, controller-based interactions, they are not directly comparable to the proposed methods and are therefore not included as evaluation baselines in this thesis.

4.2 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. Figure 4.1 presents the overview of the taxonomy. It structures interactions from abstract objectives to implementable actions, providing a foundation for systematic XR testing and directly informing INTENXION’s design (Section 4.3).

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”.

Although the taxonomy is established using Unity and 6DoF controller inputs in this thesis, its three-layer structure is not specific to a particular platform or input modality. The high-level tasks describe user intentions instead of their implementation, while atomic actions can be defined for other input

¹<https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/script-EventTrigger.html>

modality. For example, eye-tracking interactions can be decomposed into gaze movements, fixation, and dwell actions [122]. Multimodal interactions can then be expressed as compositional patterns that coordinate actions across modalities, such as gazing at an object before issuing a controller action. Consequently, supporting another XR framework or modality requires defining its atomic actions and composition rules rather than changing the taxonomy’s underlying structure.

4.2.1 Core Task Categories

XR interaction fundamentally revolves around three core spatial tasks [14]:

- **Selection:** identifying, acquiring, and activating interaction targets;
- **Manipulation:** transforming objects through translation, rotation, or scaling;
- **Navigation:** repositioning the user’s viewpoint or locomotion within the environment.

These categories represent high-level user intents, forming the key objectives for XR interaction testing. It encompasses both direct object engagement and spatial exploration.

4.2.2 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 shown in Figure 2.4). These controllers enable interaction through two essential input modalities: the *trigger* for discrete activation (e.g., firing, confirming selections) and the *grab* (or grip) for continuous object possession.

When combined with *controller movement*, these modalities enable four basic interaction tasks [123]. These tasks include: 1. *Pointing*: spatial targeting of objects, e.g., directing a ray or laser from the controller to target distant objects; 2. *Selection*: initiating/confirming interaction intents, e.g., pointing or hovering targets; 3. *Translation*: linearly repositioning grasped objects through controller movement; 4. *Rotation*: reorienting objects by rotating the controller while maintaining grip.

4.2.3 Compositional Patterns

Realistic XR behaviours emerge from composing atomic actions according to established design principles [124–127]. These principles include:

- **Constrained Physics:** restricting movement to specific axes, planes, or paths (e.g., pulling a lever along a single axis or sliding a drawer);
- **State Persistence:** maintaining interaction context across actions (e.g., carrying a grabbed object while navigating);
- **Sequential Patterns:** ordered atomic action sequences, e.g., ⟨grab and hold → spatial translation → release⟩.

This three-layered taxonomy covers from high-level intents to atomic primitives to governed compositions. It provides the conceptual framework for informing the design of INTENXION (Section 4.3). It bridges high-level interaction tasks to implementable low-level XR test actions.

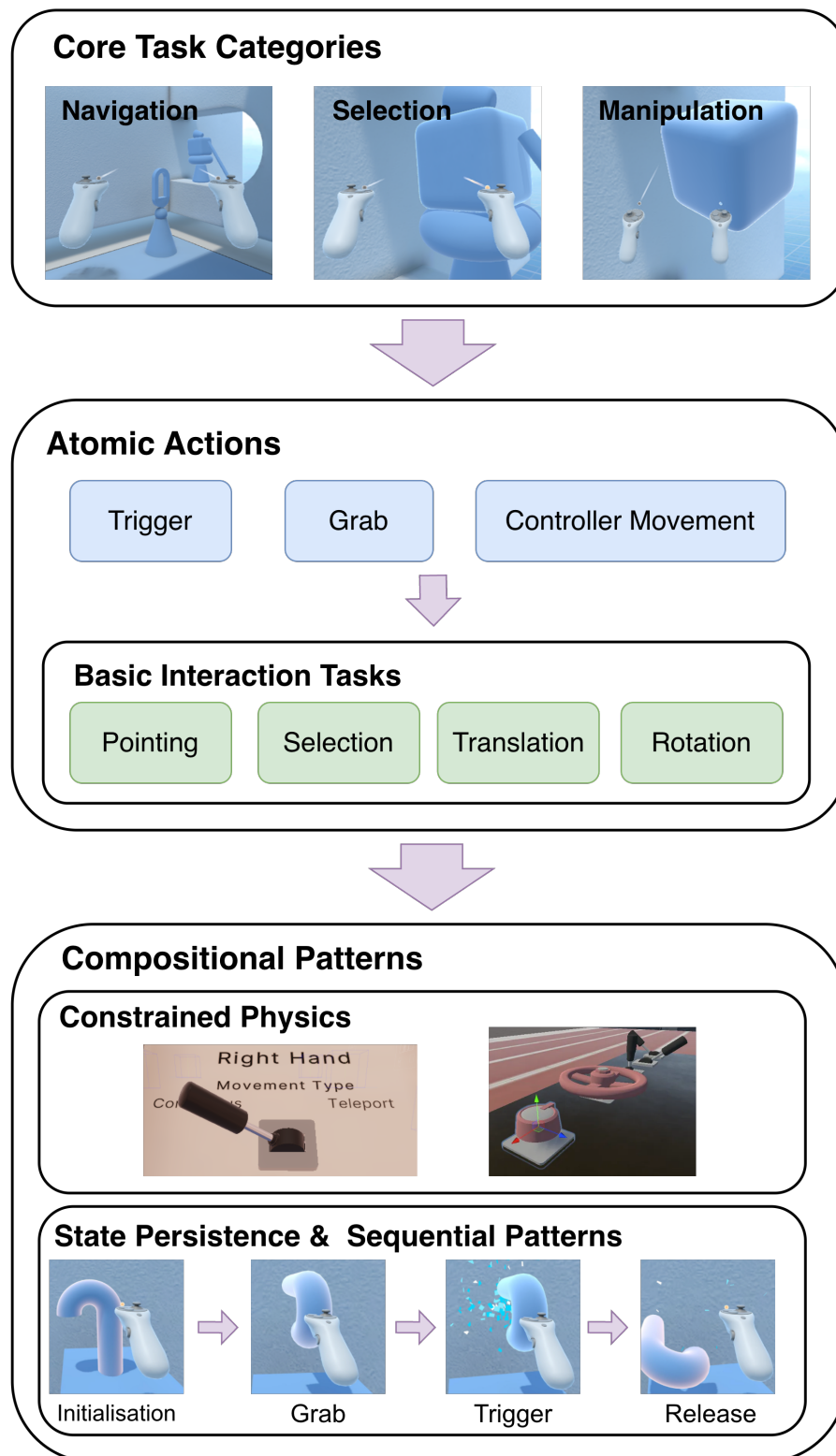


Figure 4.1: Overview of the XR Interaction Taxonomy

4.3 INTENXION Test Library

Building upon the XR interaction taxonomy (Section 4.2), 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 4.2 illustrates the overview of developing XR interaction tests with INTENXION.

4.3.1 Testable Interaction Mapping

Building on our interaction taxonomy (Section 4.2), 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 4.1 summarises the mapping of interaction tasks and test actions.

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.

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 Section 4.2.

Manipulation

Manipulation enables spatial transformation of objects through sustained controller engagement, utilising the continuous grab actions (Section 4.2). This task encompasses two fundamental operations: translation and rotation (Section 4.2.2).

4.3.2 Test Library Implementation

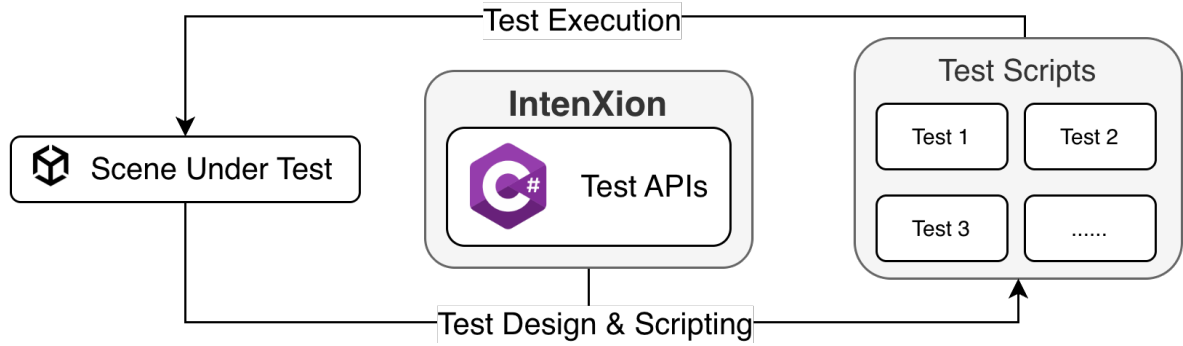
INTENXION is designed as a test automation framework for XR user interactions, compatible with the Unity Test Framework (UTF) [128]. The relationship resembles combining unit testing frameworks like JUnit² and unittest³ with APPIUM for GUI testing. UTF offers a similar test infrastructure, including

²<https://junit.org/>

³<https://docs.python.org/3/library/unittest.html>

Table 4.1: 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, Grab (instant/continuous), Trigger (instant/continuous)	Controller movement, Grab, Trigger
Manipulation	Spatial transformation, Continuous grab	Controller movement, Grab

**Figure 4.2:** Overview of INTENXION

test case management, execution control, and reporting [40], while INTENXION delivers XR-specific capabilities.

We use Unity’s XR Interaction Simulator [129] 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 4.2 lists all the APIs provided by INTENXION, covering the tasks of *Navigation*, *Selection*, and *Manipulation*, aligning with our mapping (Section 4.3.1). 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.

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 (Section 4.2), INTENXION adopts APPIUM’s *action chains* mechanism [130]. It facilitates 1. programmatic construction of ordered action sequences, 2. intuitive authoring of complex interaction patterns, and 3. human-readable test scripting.

Figure 4.3 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

Table 4.2: Test APIs Provided by INTENXION

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

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 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.

Assertion

Assertions serve as code-based test oracles, validating expected behaviours in specific scenarios [37] and are essential for systematic test automation [131]. INTENXION provides specialised assertion APIs (Table 4.2) that implement two complementary validation approaches:

- **Interaction state validation** that leverages Unity XRI’s interaction event system [132], 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

```

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");

```

Figure 4.3: A test case developed with INTENXION: Navigating to an object, grab holding, rotating, triggering, and releasing (action chains highlighted)

extensibility for developing custom assertions tailored to unique interaction events and object properties. Consider Unity’s *VR Template* scene [133], 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.

4.4 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 [124, 127]. These object types require distinct interactions to fully exercise functionality. Together, these object types exercise the main elements of the taxonomy used in this chapter, including instant activation, sustained grabbing, constrained translation, constrained rotation, object placement, and combined grab-trigger interactions. They are therefore suitable for demonstrating the framework, although they are not intended to exhaust all possible XR interaction designs. 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 [134], 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>.

4.4.1 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, checking that each test exercised the intended behaviour of its corresponding object type. Figure 4.3 demonstrates the test case for the *Weapon* object type. Across the eight test cases, the average and median sizes of test cases are both approximately 18 lines of code.

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 Section 4.5.1).

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 Section 4.5.2).

The case study provides coverage of the intended behaviours implemented by the eight test subjects. It does not claim that a single test case is sufficient for every real-world variant of these interaction types. In larger applications, additional tests may be needed to cover edge cases like alternative paths or invalid inputs. 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.

4.5 Discussion

This section reflects on the quality and generality of the test cases developed with INTENXION, and briefly outlines potential directions for strengthening XR-specific testing methodologies.

4.5.1 Test Case Quality

Test case quality is assessed by examining the extent to which tests created with INTENXION exercise the intended functionalities of the interactable objects and verify their observable behaviours. For the eight representative interactable object types in our case study, tests developed with INTENXION achieve full functional coverage in this sense: each object’s designed behaviours are triggered through explicit test actions, and corresponding assertions are provided to validate the expected outcomes.

The current tests focus on expected interaction sequences, such as grabbing before moving or triggering while holding an object. Real users may perform actions in different orders, interrupt an interaction, or resume after a pause. Such partial or reordered behaviours should be treated as additional test scenarios rather than being covered by the representative success-path tests used in this case study.

Conventional quantitative metrics, such as code coverage, while informative, offer only a partial view of test adequacy for Unity-based XR applications. Unity apps are realised not only through source code but also through rich scene configurations such as hierarchies, component properties, and prefabs (i.e., prefabricated assets), that drive runtime behaviour. A focus solely on executed code can therefore overlook configuration-dependent functionality, leading to an overly optimistic view of test quality. In XR contexts, this limitation is amplified by spatial and interactional complexity that may be encoded primarily in scene assets rather than in control logic.

Mutation testing has shown promise as a more discriminating indicator of test effectiveness, including in GUI domains where both source code and interface artefacts can be systematically mutated [135]. For Unity and XR applications, *text-based serialised scene files* [136] provide natural mutation targets. These files capture the structural and behavioural definitions of a scene, including object hierarchies, object configurations, and component properties, enabling controlled perturbations.

Future work should investigate mutation operators tailored to XR environments, such as modifying collider shapes, interaction ranges, or physics constraints, to evaluate how well tests developed with INTENXION detect realistic faults. Such XR-oriented mutation testing could complement functional coverage and provide a more nuanced assessment of test suite quality.

Specifically, Unity’s text-based serialised files describe the configuration of GOs, prefabs, and scenes in a readable format that reflects their internal properties. Figure 4.4 illustrates an excerpt from such a file, showing a *GameObject* definition with multiple component references and a property `m_IsActive` (line 18) indicating whether the object is active. Scene files can also reference multiple prefab or object files, creating a nested and hierarchical structure that reflects the composition of runtime scenes.

This static representation enables precise mutation operations without requiring dynamic execution or recompilation, such as altering object activation states, modifying component parameters, or injecting inconsistencies into scene hierarchies. For example, researchers could define XR-specific mutation operators that adjust collider trigger enablements (discussed in Section 5.4.4 as test oracles), object visibility, or physics constraints to simulate realistic design or configuration faults. Evaluating how tests developed with INTENXION detect these faults would provide a more nuanced assessment of test suite quality than traditional coverage metrics alone. We did not implement or evaluate these mutation operators in this thesis. They are discussed here as a future direction for assessing XR test-suite strength beyond the representative functional coverage used in the case study.


```

1  --- !u!1 &6
2  GameObject:
3      m_ObjectHideFlags: 0
4      m_PrefabParentObject: {fileID: 0}
5      m_PrefabInternal: {fileID: 0}
6      importerVersion: 3
7      m_Component:
8      - 4: {fileID: 8}
9      - 33: {fileID: 12}
10     - 65: {fileID: 13}
11     - 23: {fileID: 11}
12     m_Layer: 0
13     m_Name: Cube
14     m_TagString: Untagged
15     m_Icon: {fileID: 0}
16     m_NavMeshLayer: 0
17     m_StaticEditorFlags: 0
18     m_IsActive: 1

```

Figure 4.4: Format of Unity Text Serialised files

4.5.2 Extensibility

INTENXION is designed to be extensible, allowing testers to define custom interaction routines and assertion logic beyond the core 3D object interactions presented in this chapter (Section 4.3.2). In principle, this enables the framework to support additional scenarios such as XR locomotion patterns, complex tool use, or application-specific interaction schemes, as long as they can be decomposed into controller-level actions and observable outcomes.

Effectively exploiting this extensibility, however, requires testers to possess a solid understanding of the XR scenes and objects under test. Such domain knowledge is essential for identifying meaningful interaction goals, designing representative action sequences, and implementing validators that capture subtle or application-specific correctness properties. Without this understanding, extensibility risks being underused or misapplied.

The current implementation of INTENXION, as well as the objects in our case study applications, primarily targets interactions that can be performed with a single controller. Nonetheless, the framework can be extended to support coordinated multi-controller interactions, such as dual-hand scaling via symmetric translation and rotation, or two-handed weapon handling where one controller stabilises a rifle while the other controls firing. These patterns can be expressed by synchronising multiple controller action streams within the existing abstraction.

For example, a two-handed scaling test could hold a virtual object with both controllers, move the controllers apart along opposite directions, and assert that the object’s scale increases. A two-handed weapon test could keep one controller fixed on the support handle while the other presses the trigger, then assert that the firing behaviour occurs only when both interaction states are satisfied.

Given the practical and exploratory nature of our contribution, the present framework and test suite design should be viewed as an initial step rather than definitive best practice. Future empirical studies with industry practitioners will be important to assess how well INTENXION’s extensibility aligns with real-world XR development workflows, to identify missing abstractions, and to refine the framework so that it better supports large-scale, heterogeneous industrial XR projects.

4.6 Answers to ORQ2 and ORQ3

This provides the answers to ORQ2 and ORQ3 introduced in Section 1.1.

4.6.1 ORQ2: XR Interaction Synthesis

This chapter provides an answer to ORQ2 from Section 1.1:

ORQ2: *How can individual XR interactions be systematically decomposed and synthesised as executable test actions?*

The answer lies in the combination of the proposed three-layered taxonomy of XR interactions and its concrete instantiation in INTENXION. The taxonomy organises XR interactions from high-level user intents (core tasks) through atomic input primitives to compositional patterns, thereby providing a structured conceptual basis for interaction synthesis (Section 4.2). This structure is then operationalised in INTENXION (Section 4.3), where each element of the taxonomy is mapped to programmable test actions.

Concretely, high-level tasks in the categories of *navigation*, *selection*, and *manipulation* are decomposed into low-level controller operations such as trigger presses, grab actions, and 6DoF controller movements. For each category, we define explicit test responsibilities (e.g., reaching a target location, acquiring an object, transforming an object) and the corresponding action sequences required to exercise them. INTENXION exposes these sequences through an intuitive API, allowing tests to be written directly in terms of meaningful XR interactions while being executed as precise, low-level controller commands.

By bridging the taxonomy and the API design, this chapter demonstrates that meaningful XR interactions can be systematically synthesised by: (1) classifying interaction intents; (2) decomposing them into device-level primitives; and (3) encapsulating these primitives as reusable, executable test actions. This synthesis process turns abstract interaction designs into concrete, programmable scenarios that can be used consistently across XR testing approaches and tools.

4.6.2 ORQ3: Effectiveness of Synthesised XR Interaction

ORQ3: *How effective are the synthesised XR interactions at evaluating XR applications?*

Effectiveness, in this context, is measured by the extent to which synthesised interactions can exercise intended behaviours and support precise validation of outcomes. INTENXION exposes test APIs grouped into *navigation*, *selection*, *manipulation*, and *assertion*, directly instantiating the three-layered interaction taxonomy as concrete, low-level executable actions. This alignment ensures that the synthesised interactions are both semantically meaningful (reflecting user intents) and operationally precise (mapped to controller-level commands).

To empirically assess effectiveness, a case study was conducted on eight representative interactable object types, selected from industrial XR interaction design guidelines [124–127]. These objects embody diverse interaction requirements (e.g., constrained movements, object grabbing and releasing, and spatial repositioning), thus providing a challenging testbed. For each object type, tests written with INTENXION successfully exercised all intended interaction functionalities and incorporated assertions to verify the

corresponding expected behaviours. This demonstrates that the synthesised interactions can evaluate the functional behaviours represented by the case-study objects, while broader effectiveness across larger and more varied XR applications remains a topic for future empirical evaluation.

Moreover, the extensible API design of INTENXION enables testers to introduce custom interaction routines and domain-specific assertions, enabling coverage of a broader spectrum of XR interaction patterns and application-specific behaviours. Taken together, the taxonomy-grounded APIs, full functional coverage in the case study, and extensibility for additional patterns indicate that the synthesised XR interactions are effective for evaluating XR applications in terms of both interaction coverage and validation capability.

4.7 Conclusion

This chapter introduced a three-layered taxonomy of XR user interactions that organises high-level intents, atomic input primitives, and compositional patterns into a coherent framework for testing [121]. The taxonomy captures core XR tasks and systematically relates them to low-level controller actions and design principles for combining these actions into realistic interactions. This conceptual structure provides a foundation for reasoning about what it means to exercise XR applications in a behaviourally meaningful way.

The empirical analysis presented in Chapter 5 provides further evidence for the applicability of this taxonomy. All 456 interactions identified across ten open-source XR scenes can be expressed using its three layers. Specifically, these interactions: 1. at the task level, involve selecting or manipulating objects; 2. at the atomic-action level, are composed by grab, trigger and controller movement actions; 3. at the compositional-pattern level, incorporate persistence, sequencing, or movement constraints.

Building on this taxonomy, the chapter presented INTENXION, a test automation framework for validating user interactions in XR applications. INTENXION instantiates the taxonomy as programmable test APIs that simulate spatial interactions via 6DoF controllers and provide assertion mechanisms for checking expected outcomes. A case study using eight representative interactable object types from industrial XR design guidelines showed that test cases written with INTENXION can fully cover the intended interaction functionalities of these objects, with explicit action sequences and corresponding assertions for each behaviour.

Although INTENXION currently targets *test automation*, i.e., scripted execution of manually authored tests, it establishes the necessary abstractions for moving towards *automated testing* with automatic generation of interaction sequences and oracles. This transition is the focus of the next chapter, which builds on the interaction taxonomy and the execution model of INTENXION to synthesise XR tests automatically. Beyond this, future work includes extending platform support beyond Unity to increase compatibility across the XR ecosystem and conducting empirical studies with industrial practitioners to evaluate and refine INTENXION in real-world XR development settings.

Chapter 5

Automated Testing for XR User Interactions

This chapter advances the thesis from test automation to fully automated testing for XR user interactions. Unlike conventional 2D mobile apps, XR applications require validating rich 3D interactions and exploring spatial scenarios, which creates a vast input space of possible user actions that current techniques struggle to cover systematically [13, 137]. Most existing VR testing approaches, such as VRTEST [54] and VRGUIDE [55], remain centred on legacy 2D-style UI elements rather than the spatial, controller-based interactions that characterise modern XR experiences (see Chapter 4.1). Chapter 4 introduced INTENXION [121], a test automation library that simulates realistic 3D XR inputs via scripted controller actions, but its scope is limited to automated execution of manually authored tests and does not address automated test generation.

Building on the insights from Chapter 3 (the essentials of user interaction in XR software testing and the lack of robust automated oracles), and Chapter 4 (the formalised taxonomy of XR interactions), this chapter moves from executing predefined tests to automatically creating them. The goal is to develop an automated testing approach that systematically models user interactions in XR apps and automatically synthesises realistic controller-based interaction sequences. Because no prior work has empirically characterised how XR user interactions are implemented in existing open-source XR projects, we first conduct a study of such projects to identify prevalent 3D interaction types. Based on results from ten representative XR scenes, we identify four prevalent interaction categories: *fire*, *manipulate*, *socket*, and *custom*. Each category captures distinct interaction mechanics, collectively encompassing both simple and complex composite patterns.

To capture these interactions in a form amenable to automated testing, we propose a novel static model called **Interaction Flow Graph (IFG)**. It emphasises the *interaction flows* between interactive objects (e.g., controllers, buttons, grabbable items), representing them as nodes and the directed edges between them as possible interaction pathways. Edges are further annotated with labels that specify required interactions to traverse them (e.g., firing, grabbing), thereby encoding the conditions needed to exercise object functionalities. This graph-based abstraction provides the structural foundation for systematic test generation.

On top of this model, we develop XRINTTEST, a novel automated testing tool that implements a

hybrid approach to statically construct IFGs and dynamically traverse them. It concentrates on the fundamental interaction types of *fire*, *manipulate*, and offers extensibility to support others. Guided by the IFG, XRINTTEST automatically explores scenes to *activate* interactions. Within this scope, XRINTTEST is, to our knowledge, the first automated framework specifically targeting 3D interactions in XR applications. It automatically derives interaction flows from the IFG, drives virtual hand-held controllers to realise the required actions, and executes resulting tests end-to-end. In doing so, it enables developers to validate complex spatial user interactions in their XR projects without relying solely on manual testing with physical HMDs and controllers, thereby extending the automation capabilities introduced in Chapter 4 to full automated testing of XR user interactions.

XRINTTEST necessarily involves essential XR functionalities such as locomotion for scene navigation (see Section 5.3), but the systematic testing of locomotion mechanisms themselves (e.g., teleportation schemes) is beyond the scope of this work. Likewise, we deliberately exclude traditional UI interaction testing, as such elements do not fully exploit the spatial capabilities that characterise immersive XR experiences. Instead, the focus is on inherently *spatial* interaction patterns, which pose distinct testing challenges unique to realistic 3D environments.

To evaluate our approach, we first identified the lack of a suitable public benchmark for automated XR interaction testing techniques. Although some prior studies have released portions of their evaluation subjects, these datasets lack the modern 3D spatial user interactions that are central to our work, focusing instead on simplified graphical user UI elements rather than immersive, spatial interactions [54, 55]. To fill this gap, we constructed a novel benchmark dataset XRBENCH3D, which comprises ten VR scenes with 456 3D interactions from nine open-source projects. The evaluation shows that XRINTTEST substantially improves interaction coverage and efficiency over a random baseline, while also detecting configuration-related interaction issues. Detailed results are reported in Section 5.4. In addition, we propose a novel automated oracle that enables XRINTTEST to detect non-exceptional functional issues automatically. Furthermore, the IFG helps identify potential interaction smells, providing deeper insights into overall interaction design quality. The implementations of XRINTTEST and the XRBENCH3D benchmark are publicly available at <https://github.com/ruizhengu/XRintTest> and <https://github.com/ruizhengu/XRBench3D>, respectively.

In summary, this chapter makes the following contributions:

- **XR Interaction Categorisation:** We present, to our knowledge, the first empirical categorisation of XR user interactions, derived from analysis of open-source XR projects.
- **Interaction Model for XR Testing:** We introduce the Interaction Flow Graph (IFG), a novel abstraction that systematically represents interactions in XR environments and serves as a foundation for XR testing strategies.
- **Dataset:** We construct XRBENCH3D, a novel benchmark comprising **ten** XR scenes with **456** spatial user interactions. It facilitates standardised evaluation of XR testing approaches, addressing the absence of dedicated XR testing resources.
- **Technique:** We develop a model-based testing technique that leverages IFGs to explore XR scenes and activate interactions automatically. Our tool, XRINTTEST, achieves 93% interaction coverage on XRBENCH3D, outperforming a random baseline nine times in effectiveness and five times in efficiency.

This chapter is structured as follows. Section 5.1 presents our empirical analysis of prevalent XR interaction types. Section 5.2 formalises the interaction flow graph, and Section 5.3 details our proposed technique. Section 5.4 presents our empirical evaluation and results. Section 5.5 provides the answers to ORQ4, ORQ5 and ORQ6 from Section 1.1. Section 5.6 discusses the implications and findings derived from our study, and Section 5.7 concludes the chapter and outlines future work directions.

The content of this chapter is based on “Ruizhen Gu, José Miguel Rojas and Donghwan Shin. XRINTTEST: An Automated Framework for User Interaction Testing in Extended Reality Applications, in *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering*, 2025.” [138], and on “Ruizhen Gu, José Miguel Rojas and Donghwan Shin. Automated Testing of Prevalent 3D User Interactions in Virtual Reality Applications” published in the special issue of *Virtual and Augmented Reality Software Engineering* in the journal *Automated Software Engineering*, 2025 [139].

5.1 Prevalence of XR Interactions

Prior work has emphasised the lack of standardised categories for XR interactions [13, 63]. While Chapter 2 introduced the main XR input modalities (in particular, *grab* and *trigger*) and Chapter 4 proposed a taxonomy of XR user interactions grounded in interaction design guidelines, it is still unclear how these interaction types are actually adopted in real-world XR applications. The extent to which different interaction patterns appear in practice, and how frequently they occur, has not been systematically examined.

To address this gap and support automated XR testing, we adopted an empirical methodology to investigate the prevalent interactions in open-source XR apps. These insights are critical for prioritising testable interactions and ensuring comprehensive XR functionality coverage.

5.1.1 Data Collection

To collect subjects to analyse prevalent VR interactions, we focus on open-source VR projects built with XRI, a widely adopted framework for implementing 3D VR interactions in Unity-based apps. We collected candidate projects from two representative sources, GitHub¹, by searching for the XRI package identifier “`com.unity.xr.interaction.toolkit`” and Unity Asset Store², using the keywords “XR” and “VR”. The initial search yielded 46 GitHub repositories and 315 free Unity Asset Store packages (253 VR + 62 XR, with potential duplicates).

To evaluate our approach for testing 3D user interactions in VR apps, our selection criteria focused on identifying *self-contained* projects with sufficient interaction mechanics, such as object manipulation via controllers. We manually screened all candidate projects, first excluding partial components, SDKs, and non-standalone features. We then downloaded, configured, and executed the remaining candidates to verify they contained a minimum of five distinct interactable objects, ensuring a meaningful basis for evaluation. This selection process yielded ten distinct VR scenes across nine qualifying projects (seven from GitHub and two from Unity Asset Store).

¹<https://github.com/>

²<https://assetstore.unity.com/>

Table 5.1: Prevalence of 3D Interactions in VR Projects (Manip.: Manipulate)

ID	Project	Scene	Fire	Manip.	Socket	Custom	Tot.
Scene 1	VR Template	Sample Scene	1	8	0	2	11
Scene 2	XRI Starter Assets	Demo Scene	2	6	0	3	11
Scene 3	XRI Starter Kit	XRI Starter Kit	10	28	0	30	68
Scene 4	XRI Examples	Examples_Main	8	44	5	27	84
Scene 5	VR-Game-Jam	Game Scene	3	21	2	8	34
Scene 6	The Escape Room	Prototype Scene	6	11	1	2	20
Scene 7	The Escape Room	Escape Room	19	112	2	7	140
Scene 8	XRToolkitEssentials	XR.Sockets	0	5	5	0	10
Scene 9	EscapeTheRoomVR	SampleScene	0	14	0	1	15
Scene 10	VR-Room	My VR Room	6	22	32	3	63
Total			55	271	47	83	456

5.1.2 Results

To identify and categorise these interactions, we manually examined each GO’s attributes (e.g., attached interaction scripts) using the Unity Editor. We further validated our findings through hands-on exploration of the scenes using a Meta Quest 3 HMD. Our analysis identified a total of 456 interactions. During this process, we observed a complexity where interactables were generated or destroyed at runtime. To ensure consistency and reproducibility, we focus only on the interactions that exist in the scene definition.

Interactables created or destroyed at runtime can affect fair comparisons between scenes. They can change which interactions are available during each execution, changing the total number of interactions against which coverage is measured. For instance, destroyed interactables become untestable, reducing both measured coverage and opportunities for bug detection. Consequently, a lower-quality scene may appear to contain fewer bugs simply because fewer of its interactions were exercised. To ensure consistent and reproducible evaluation, we therefore consider only interactions present in the static scene definition.

Table 5.1 summarises the categorisation of 456 3D interactions across the ten subject scenes into four groups: *fire*, *manipulate*, *socket* and *custom*, each representing various VR interaction mechanisms. *Manipulate* interactions are the most straightforward, involving object grabbing via the controller’s grab button (Section 2.2.1), and may also be combined with controller translations for more complex manipulation [121]. In contrast, *fire* interactions require a composite sequence: the user must first grab and hold a virtual object (e.g., a gun) persistently and then press the trigger button to fire. This persistent grab distinguishes continuous from instant actions [121]. Notably, all trigger actions in XRBENCH3D occur while an object is actively grabbed, underscoring the prevalence of this composite pattern.

The *socket* and *custom* interactions also adopt composite patterns similar to the *fire* interactions. *Socket* interactions involve a **XR Socket Interactor** component, which serves as a dedicated target for a specific interactable, such as the keyhole for a key. To complete a socket interaction, the user must continuously grab an object and move it precisely to the socket’s position. While socket interactions do not directly map to explicit user inputs, they represent intentional responses to user actions, distinguishing them from autonomous behaviours like physics-driven collisions. The *custom* category includes various interactions tailored to specific scenarios. For example, a *knob* interaction permits rotational movement along a fixed axis during a continuous grab. Other examples include sliders constrained to a linear track, levers constrained to a hinge-like movement, and door handles that must be rotated or

pulled before the door can open. These interactions still rely on grab, trigger, or controller movement, but add object-specific rules that determine whether the interaction succeeds. While users typically initiate these interactions through standard grab and trigger actions, successful activation often depends on following additional rules and constraints [121].

Among the 456 interactions, *manipulate* interactions dominate with 271 instances (59%), followed by *fire* (55, 12%), *socket* (47, 10%), and *custom* (83, 18%) interactions. These scenes form our benchmark XRBENCH3D. We carefully adapted these projects with minimal changes to ensure compatibility with the latest Unity and XRI versions (Section 5.4). This standardisation enables consistent evaluation across all test subjects while preserving their original interactions.

5.1.3 Discussions

During the empirical analysis of user interactions in the subject VR scenes, we observed several potential issues that can negatively impact the overall testing process and performance.

Unresponsive Interaction Issues

During manual exploration of the subject VR scenes, we repeatedly encountered cases where users correctly positioned their controllers and *initiated* the expected interactions, but failed to *activate* them. Here, activation is defined as successfully issuing the required input on an interactable object so that the associated interaction logic and the object’s intended behaviour are fully exercised. By contrast, initiation refers to the input sequence that only starts, but does not guarantee that all preconditions for activation are satisfied. For example, consider a user who positions a controller in contact with a grabbable object and performs the grab action, yet the object fails to respond or be grasped successfully. This case illustrates an interaction that has been initiated but not activated.

We refer to cases where interactions are initiated but fail to activate as *unresponsive interaction issues*. In our subject scenes, we identified two concrete instances of such issues, where the target interactables had interaction scripts attached but did not respond to valid interaction inputs.

Socket Interactions

Socket interactions are more complex than *fire* and *manipulate* interactions, as they involve one interactable object and two interactors (e.g., an input interactor and a target socket interactor). Notably, the Interaction Layer Mask (discussed in Section 2.2.1) of socket interactors is set to “Everything” by default, likely allowing unintended cross-interactions. For example, while a “Key” interactable should only pair with its matching “Keyhole” socket interactor, the “Everything” layer setting permits any socket interactor to engage with any interactable (e.g., the “Keyhole” can be unlocked with a “Mug”), potentially violating the intended design. In this thesis, a design smell refers to an implementation or configuration pattern that is not necessarily a fault in every context, but may indicate a likely violation of intended interaction behaviour or maintainability. We consider this misconfiguration a potential design smell, which we discuss further in Section 5.6.1.

Additionally, one interactable can pair with multiple socket interactors, and vice versa. For example, Figure 5.1 shows socket interactions from Scene 10, featuring two “Painting” interactables and four “Picture Hook” socket interactors. When these objects share the same Interaction Layer Mask,

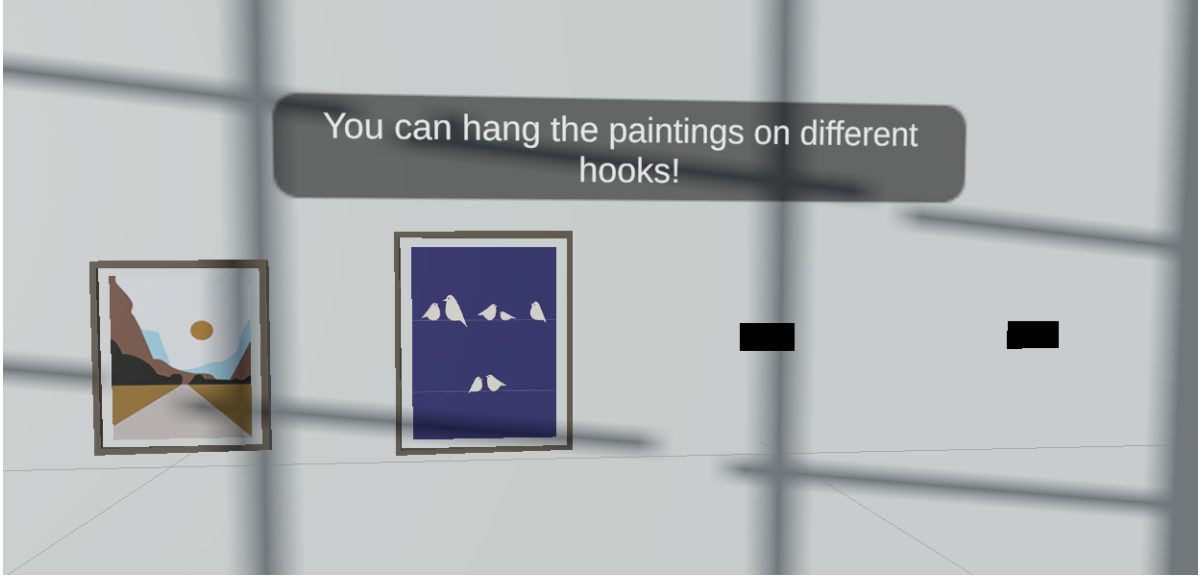


Figure 5.1: Permutations of Socket Interactions (an example from Scene 10)

fully exercising the socket interaction functionality requires testing all possible pairings between interactables and socket interactors. Since multiple interactables cannot occupy the same socket interactor simultaneously, we calculate the total number of valid socket interactions within one layer mask using the permutation formula $P(n, r) = {}_nP_r = \frac{n!}{(n-r)!}$, where n is the number of interactables and r is the number of socket interactors. In this case, $P(4, 2) = \frac{4!}{(4-2)!} = \frac{24}{2} = 12$, yielding 12 distinct socket interactions.

While socket interactors' Interaction Layer Masks default to "Everything," valid socket interactions should restrict each interactor to specific interactables via targeted layers. To ensure meaningful socket interactions in our subject scenes, we manually screened all socket interactors, inferred their intended target interactables, and assigned corresponding custom layers. For instance, the "Painting" interactables and "Picture Hook" socket interactors in Scene 10 were grouped within a dedicated "Painting" layer, eliminating any "Everything" layer defaults.

To compute the total number of socket interactions across a scene, we apply the permutation formula within each Interaction Layer Mask. The total is given by $\sum S = s_1 + s_2 + \dots + s_n$, where $s_i = P(b_i, s_i) = \frac{b_i!}{(b_i-s_i)!}$ represents the number of valid interactions in layer i , with b_i interactables and s_i socket interactors.

5.2 Interaction Flow Graph

Modelling 3D user interactions in XR apps is challenging due to their complex, spatial, and highly interactive nature (Section 2.3.4). Models such as [Extended Finite State Machine \(EFSM\)](#) [46] can capture rich state and transition information, but they easily suffer from state-space explosion when every fine-grained action (e.g., a discrete controller movement) is represented explicitly [140]. This limits their practicality for testing large or complex XR scenes.

To address these challenges, we propose the [Interaction Flow Graph \(IFG\)](#), a variation of EFSMs that

abstracts away low-level state transitions while retaining the information needed for systematic testing. It is represented as a scalable model that can cover all interaction categories in Section 5.1. In the IFG, nodes represent the user and interactable objects, while directed edges capture valid interactions flowing from source to target. Each edge is annotated with an intended interaction (e.g., a continuous controller movement) as well as any preconditions required to perform it.

Crucially, an annotated interaction corresponds to a single logical transition, regardless of the underlying sequence of controller inputs. The interaction denotes what is required to *activate* an interactable’s functionality. Activation is defined as successfully issuing the necessary inputs on an interactable object (for example, pressing the grab button on the controller), thereby triggering the associated interaction logic and fully exercising the object’s intended behaviour. By contrast, an *initiated* interaction may start the input sequence but does not guarantee that all conditions for activation are satisfied.

Furthermore, the complexity of modern spatial VR interactions necessitates models like the IFG. Figure 5.2 illustrates both the scene graph [50] and IFG of an example scene. The scene graph (Figure 5.2a) lists all objects in the scene, but not all are interactables, and many support multiple interaction types beyond simple point-and-click. Different interactables require specific actions to exercise the intended interactions. For instance, the interaction of the “Handle” object in the IFG (Figure 5.2b) can only be activated via an instant grab, not other actions like trigger presses. Existing formalisms, such as EFSMs [46], are in principle expressive enough to represent such behaviour by encoding fine-grained states and transitions. However, directly modelling modern spatial VR interactions at this level of detail could lead to state-space explosion [140]: each small variation in actions like controller position or timing becomes a separate state or transition, making the resulting models difficult to construct, interpret, and use for scalable test generation. This limits the practical applicability of EFSM-based approaches for testing large-scale, interaction-rich VR apps. In contrast, the IFG abstracts low-level continuous actions into a set of valid and meaningful interactions (e.g., *manipulate*, *fire*, *socket*) and their dependencies, based on the empirical insights gained in Section 5.1.

This interaction-centric abstraction provides an intuitive representation of the **Scene Under Test (ScUT)**, enabling systematic test generation and analysis without exhaustively modelling every intermediate state. The IFG thus serves as a repository of testable interactions and their possible flows, allowing test generation approaches to strategically select and combine interactions to construct meaningful test cases. Intermediate states can still be important for checking detailed correctness, such as whether an interaction is successfully activated. The IFG abstracts these states to keep test generation scalable, while detailed state validation is handled by runtime assertions and oracles during execution. Note that the term “flow” in IFG refers to paths of interactions between objects, and should not be confused with the experiential notion of “flow” in domains like video game design, which relates playfulness and engagement [141].

We define user interaction flows as **user-driven exchanges** between interactors and interactables, initiated by intentional user inputs. Unintentional physical consequences, such as an object being destroyed by a previous action, are not represented as IFG edges unless they correspond to an explicit interaction mechanism defined by the application. Instead, these effects are treated as environmental dynamics that may influence later test execution, as discussed in Section 5.6.3. This approach explicitly excludes autonomous object interactions (discussed in Section 5.1), which are not user-initiated and thus beyond the scope of this work. Formally, an IFG is defined as a directed acyclic graph (DAG) [142] $G = (N, E, L, I, A, C)$, where:

Nodes (N) represent objects capable of interaction, including both interactors and interactables.

Edges (E) denote valid interactions between nodes. An edge $e \in E$ is defined as $e = n_i \rightarrow n_j$, connecting a source $n_i \in N$ to a target $n_j \in N$. The direction of e reflects the flow of interaction from the source to the target node.

Labels (L) annotate edges with corresponding interactions that exercise the functionalities of target interactables. Each label $l \in L$ denoting an edge e consists of an unordered list of interactions such that $l = [I_1, I_2, \dots, I_n]$.

Interactions (I) are the action sequences to form meaningful interactions, possibly conditioned on prerequisites. Each interaction I is defined as a tuple $I = (A, C)$, where A represents the action sequence and C is a (possibly empty) set of conditions required to perform the interaction.

Actions (A) refer to sequences of atomic VR input actions [121], such as controller movements or grabs. The action sequence A is expressed as $A = [a_1, a_2, \dots, a_n]$, where each $a \in A$ represents a single atomic action. For example, a *manipulate* interaction may be represented as $A = [grab_{cont}, mov_{fwd}]$, meaning that the user continuously grabs an object and moves the controller forward.

Conditions (C) specify prerequisite *interactions* that must persist to enable the activation of the current interaction. Each condition $c \in C$ references a specific interaction from a particular edge label, which may belong to the same or a different edge. For example, an interaction I_i within label l_q could be the sole condition of another interaction I_j , expressed as $(I_j = A_j, C_j = [I_i^{l_q}])$. A practical illustration is that firing a *Gun* requires the user to have first continuously grabbed the *Gun*.

Figure 5.2 illustrates both the scene graph [50] and IFG of an example scene, clarifying their similarities and distinctions. Although our approach does not directly employ scene graphs, we include the scene graph representation in Figure 5.2a to better demonstrate the IFG's distinctive characteristics. The scene graph (Figure 5.2a) organises objects hierarchically under the entry node *Scene*, with seven children spanning three hierarchical layers. In contrast, while the IFG (Figure 5.2b) also represents objects within the same scene, it emphasises *interaction flows* rather than hierarchical structure. The IFG replaces *Scene* with *User* as the entry node and distils the scene into five nodes connected by five edges with six labels. These differences highlight three defining properties of an IFG: *interaction prioritisation*, *interaction dependency*, and *interaction brokerage*.

5.2.1 Interaction Prioritisation

In an IFG, each node represents a distinct object capable of interaction, and each edge represents possible interaction flows. This design decouples interaction behaviours from the overall scene hierarchy. To enhance precision and reduce complexity, we exclude objects without interaction capabilities, focusing exclusively on those with interaction definitions (i.e., objects with attached scripts that define interactions). This approach avoids hierarchical complexity and potential inconsistencies (e.g., misaligned positions between parent and child objects), while enabling more accurate information collection to guide automated testing. Here, misaligned positions refer to cases where a parent object's transform

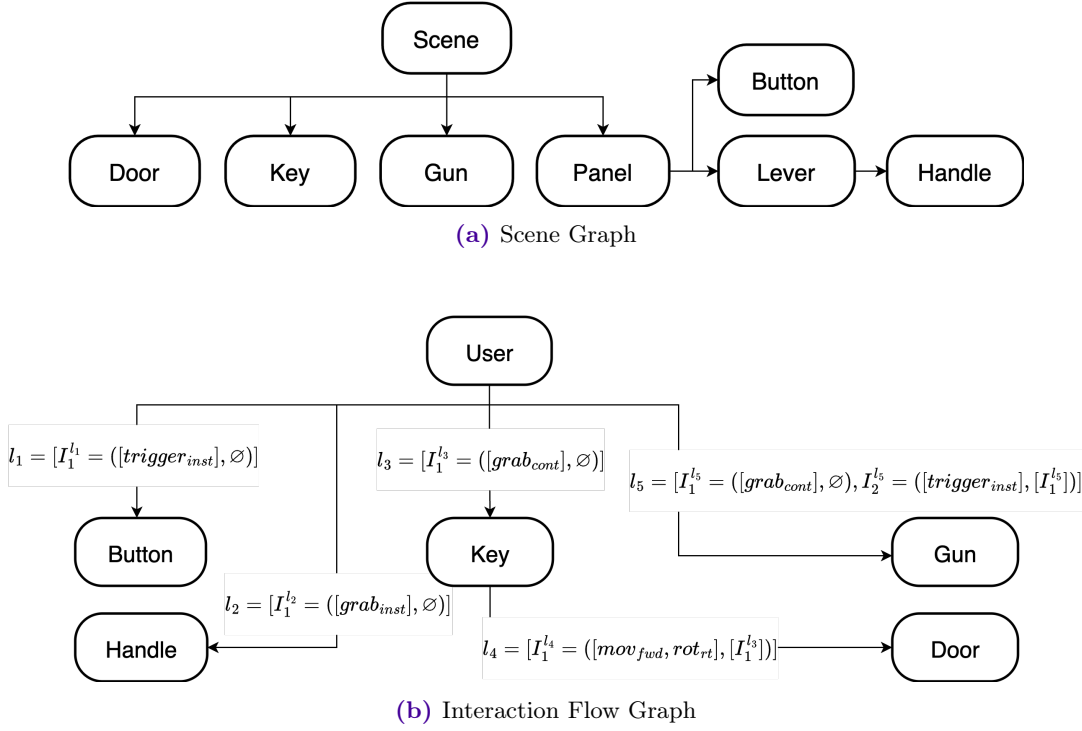


Figure 5.2: Hierarchical Structure and Interaction Flows in VR Scene Modelling

(Actions: $trigger_{inst}/grab_{inst}$ denote instant actions; $trigger_{const}/grab_{const}$ denote continuous actions; mov_{fwd} indicates forward controller movement; rot_{rt} indicate rightward controller rotation.)

does not directly indicate the position of its interactable child. For example, a parent may act only as a container located at the centre of a panel, while the actual interactable button or handle is offset within that panel. Using the parent position for testing could therefore move the controller to the wrong location.

For example, in Figure 5.2a, the *Panel* itself is not interactable, but has two children: an interactable *Button*, and a *Lever*, which is not interactable but contains an interactable child, *Handle*. In the IFG shown in Figure 5.2b, only the interactables *Button* and *Handle* are represented as nodes. Non-interactable objects such as *Panel* and *Lever*, although part of the hierarchy, are deliberately omitted. The IFG does not assume hierarchical event propagation from non-interactable parents to interactable children. If a parent object only organises children spatially or structurally, it is omitted. If a parent object implements interaction logic itself, it would be included as an interactable node. This selective representation ensures the IFG captures essential interaction information while filtering out non-interactive objects (e.g., for scene management purposes), thereby prioritising meaningful user interactions for testing.

5.2.2 Interaction Dependency

Interactions may depend on other prerequisite interactions, specified as conditions in edge labels. For example, in the IFG in Figure 5.2b, the *Gun* is connected to the *User* with the edge labelled by two interactions. The first interaction, $I_1^{l_5} = ([grab_{cont}], \emptyset)$, represents continuously grabbing the gun with

no prerequisites. The second interaction, $I_2^{l_5} = ([trigger_{inst}], [I_1^{l_5}])$, corresponds to firing the gun, which requires the persistence of the first interaction. This dependency ensures $I_2^{l_5}$ could only be activated if the state of $I_1^{l_5}$ persists. The IFG explicitly captures such constraints, ensuring correct interaction ordering.

5.2.3 Interaction Brokerage

While the *User* typically serves as the default interactor and the starting node of edges in the IFG, our analysis revealed cases where inter-object interactions are initiated by the user. In such scenarios, an object manipulated by the user acts as a *broker*, mediating interaction with another target object and forming a chained interaction flow (Section 5.1). For instance, the *Door* and *Key* in Figure 5.2b illustrate this: although the *Door* technically contains a socket interactor [121], the interaction flow is more accurately modelled with an edge from *Key* to *Door*. This represents how the user manipulates the key to unlock the door (rather than the other way around), making the key a broker between the user and the door. Moreover, interaction $I_1^{l_4}$ (use the key to unlock the door) required the prerequisite interaction $I_1^{l_3}$ (continuous grabbing the key), aligning with our defined interaction dependency property (Section 5.2.2).

This brokerage mechanism ensures the IFG accurately captures both direct user interactions and indirect, object-mediated interactions, thereby accommodating the multi-mode interaction patterns of complex XR apps.

5.3 Automated XR User Interaction Explorer

We propose XRINTTEST, an automated XR user interaction testing approach. It statically constructs the IFG from the XR scene definition, informing the exploration of the target scene and activation of interactions. Figure 5.3 illustrates the overall workflow of XRINTTEST, which is implemented as a set of scripts within the Unity XR project containing the target scene. XRINTTEST takes a Unity project containing the target scene as input. It first processes the assets required to construct the scene, performing a detailed static analysis to build an IFG representing all interactions within the scene, following the definitions described in Section 5.2. The graph then serves as a guide for the dynamic exploration process, directing the system to engage with the scene’s interactables using identified interactions. Let us now describe the main components of our approach in detail.

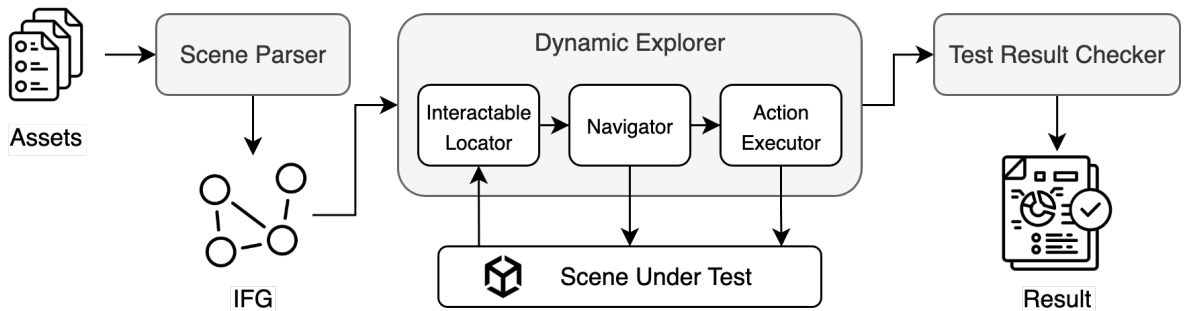


Figure 5.3: XR User Interaction Testing with XRINTTEST

5.3.1 Scene Parser

Unity XR projects comprise source code defining logic and behaviours along with scene definitions that specify GOs and their components. The objective of the scene parser is to construct a comprehensive IFG (c.f. Section 5.2) for the ScUT to facilitate the subsequent dynamic exploration phase.

The scene parser is implemented as a `EditorWindow` tool that runs inside the Unity Editor. When invoked, it traverses all the GOs within the ScUT and inspects their attached components. GOs that contain XR interaction scripts are recognised as interactables, and each script is mapped to one or more interaction types in the IFG. For example, a GO with the `XRGrabInteractable` component is mapped to the *fire* or *manipulation* interaction types (fire interactions require grabbing the object first). The resulting IFG encodes the user node, all detected interactables, and the outgoing edges that represent the interactions supported by each object. An example IFG instance is shown in Figure 5.2b, which includes one interactor *User* and three interactables: *Button*, *Handle*, *Key*, and *Gun*.

To accommodate developer-defined custom interactions (i.e., those identified in the *Custom* category in Section 5.1), the parser is extensible via a JSON configuration file. This configuration allows users to declare additional interaction types and associate them with specific scripts. For instance, a custom script `XRKnob` can be configured as supporting the *manipulation* interaction, so that any GO with `XRKnob` attached is automatically incorporated into the IFG as a grabbable object. This mechanism enables the framework to support a wide range of bespoke interaction behaviours without modifying the core analysis logic.

A practical challenge when analysing complex Unity scenes is ambiguity arising from multiple GOs sharing the same name, analogous to the “ambiguous GUT” problem in Android GUI testing [143, 144]. If objects are referenced only by name, test generation and execution may target the wrong instance. To avoid this, the parser assigns each GO a unique identifier while preserving all of its properties. These identifiers are used in the IFG and in the dynamic phase, ensuring that every interactable is unambiguously addressed during testing.

We note that the current parser operates purely on static scene definitions and therefore does not detect interactables that are created or destroyed at runtime (see Section 5.6.3). Handling such dynamic behaviour would require more extensive analysis of source code, prefabs, and runtime configuration logic (Section 5.1), which is beyond the scope of this thesis and left as an avenue for future work. The current parser focuses on interactables present in the static scene definition. Interactables introduced through prefabs, runtime instantiation, or destruction are therefore treated as part of the broader dynamic-scene limitation discussed in Section 5.6.3.

5.3.2 Dynamic Explorer

The dynamic explorer’s objective is to utilise information from the IFG (Section 5.3.1) to achieve the two key tasks of XR scene testing: scene navigation and interaction activation (Section 4.1.1). This component consists of three processes that communicate with the ScUT in real time: *interactable locator*, *navigator*, and *action executor*.

Interactable Locator

This component leverages the information from the serialised IFG (JSON file; Section 5.3.1) to precisely identify target GOs for interaction. We adopted a straightforward localisation mechanism that directly maps each interactable name identified in the IFG to its corresponding GO instance at runtime, with the measures to disambiguate interactables in the scene (Section 5.3.1).

Navigator

This navigator handles a core XR scene testing task: scene navigation. It explores the ScUT and positions the simulated user close enough to interactables to activate interactions.

The navigator component, along with the interaction executor component, is implemented as a finite-state machine with three sequential states: viewpoint navigation, controller movement, and interaction. State transitions occur when specific conditions are met. For instance, when the virtual user (i.e., a simulated agent representing the user in the XR environment) reaches sufficient proximity to the target interactable during navigation, it switches to the controller movement states.

Viewpoint Navigation This state moves the virtual user between locations, which is a key task of XR scene testing. XRINTTEST facilitates effective scene exploration by leveraging information from the serialised IFG (JSON file) to locate corresponding GOs based on their unique identifiers. Unlike methods focusing on exploration efficiency [55], XRINTTEST prioritises interaction testing effectiveness. We adopted a greedy-based navigation algorithm similar to the VRGreedy strategy provided by VRTest [54], directing the simulated user towards the closest available interactable (i.e., the closest that has not yet been interacted with) at each step. The current strategy orders interactions greedily by proximity, except where the IFG encodes explicit dependencies such as grabbing an object before firing it. It does not infer broader semantic ordering constraints between unrelated interactables, such as whether opening a door should occur before grabbing a gun. Capturing such task-level ordering would require additional goal or scenario models beyond the current IFG.

Controller Movement Beyond viewpoint navigation, the navigator is responsible for handling controller movement. This state is activated when the virtual user is close enough to a target interactable. It manipulates the position of simulated input devices to engage with interactables. It precisely controls the position of simulated input devices to engage interactables using the XR Interaction Simulator³. It is a runtime utility allowing the simulation of user interactions using standard input devices like keyboard and mouse to drive XR headsets, controllers, or hand movements in the scene. The simulator cannot directly set camera or controller positions, which only supports indirect movement via simulated inputs (e.g., pressing the “W” key to move forward) [121]. Our implementation automatically generates the corresponding inputs based on the controller’s relative position to the target, producing smooth, natural navigation while maintaining precise interaction control.

³<https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.1/manual/xr-interaction-simulator-overview.html>

Interaction Executor

The interaction executor performs the essential task of activating interactions through synthesised XR inputs, responsible for turning the abstract interactions encoded in the IFG into concrete XR inputs. The interaction executor is responsible for turning the abstract interactions encoded in the IFG into concrete XR inputs. Once the navigator has positioned the virtual user and controller in contact with a target interactable (Section 5.3.2), the executor engages that object using the interaction type (i.e., action sequence) associated with the corresponding edge in the IFG (Section 5.3.1). In this way, viewpoint navigation, controller movement, and interaction execution form a coherent pipeline from high-level model to low-level input.

Similar to controller positioning, we simulate specific key events using the XR Interaction Simulator to reproduce controller actions, ensuring consistent executions of various interaction functionalities. As with controller positioning, the executor synthesises precise input sequences that correspond to trigger and grab operations on a 6DoF controller. Compositional interaction patterns are expressed as ordered key events. For example, pressing and holding a key to initiate a grab, pressing an additional key while the grab is held to fire, and then releasing keys to complete the interaction. The implementation supports both instantaneous actions (e.g., a single trigger press) and continuous actions (e.g., sustained grabbing), thereby maintaining persistent action states consistent with our taxonomy of XR user interactions defined in Section 4.2.

Currently, XRINTTEST supports the fundamental interaction types *grab*, *trigger*, and the composite *grab-and-fire* pattern, reflecting the prevalent XR interactions we identified in Section 5.1. Figure 5.4a and 5.4b show how these patterns are implemented in code, using simulated keyboard input rather than controllers, in line with the simulator constraints described in Section 5.3.2. Figure 5.5 illustrates the example of the compositional interaction sequence corresponding to the *grab-and-fire* implementation in Figure 5.4b, highlighting how the executor orchestrates key presses, holds, and releases to activate the intended XR interaction successfully.

A key strength of the IFG and XRINTTEST is the ability to support all prevalent interaction types identified in Section 5.1, including those in the *socket*, and *custom* categories. Grounded in the XR interaction taxonomy defined in Section 4.2, the IFG follows the same principles of decomposing interactions into core tasks, atomic actions, and compositional patterns. As a result, interaction flows that realise socket-based or bespoke behaviours can also be expressed as graph edges and labels, and XRINTTEST can be extended with additional mechanisms to support them. Although the overall approach is designed for this level of generality, in this chapter we focus on the *fire* and *manipulate* interactions, reflecting their foundational role in XR user interaction and providing a clear starting point for future extensions to more specialised patterns.

5.3.3 Test Result Checker

To verify successful interactions during test sessions, XRINTTEST registers callback listeners on each interactable using XRI's built-in interactable event system⁴. After executing a corresponding interaction, XRINTTEST validates success by confirming that the callback data from the activated listeners matches the information of the target interactable. Additionally, throughout the testing process, our approach

⁴<https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.1/manual/interactable-events.html>


```

1  var kboard = InputSystem.GetDevice<Keyboard>();
2  InputSystem.QueueStateEvent(kboard, new KeyboardState(Key.G));

```

(a) Executing *grab* interaction

```

1  var kboard = InputSystem.GetDevice<Keyboard>();
2  // Hold the grab key
3  InputSystem.QueueStateEvent(kboard, new KeyboardState(Key.G));
4  // Wait time/hold duration
5  yield return new WaitForSeconds(0.1f);
6  // Press the trigger key while grabbing
7  InputSystem.QueueStateEvent(kboard, new KeyboardState(Key.T, Key.G));
8  yield return new WaitForSeconds(0.1f);
9  // Releasing both keys
10 InputSystem.QueueStateEvent(kboard, new KeyboardState());

```

(b) Executing *grab-and-fire* compositional interaction

Figure 5.4: Examples of direct *grab* and compositional *grab-and-fire* interactions

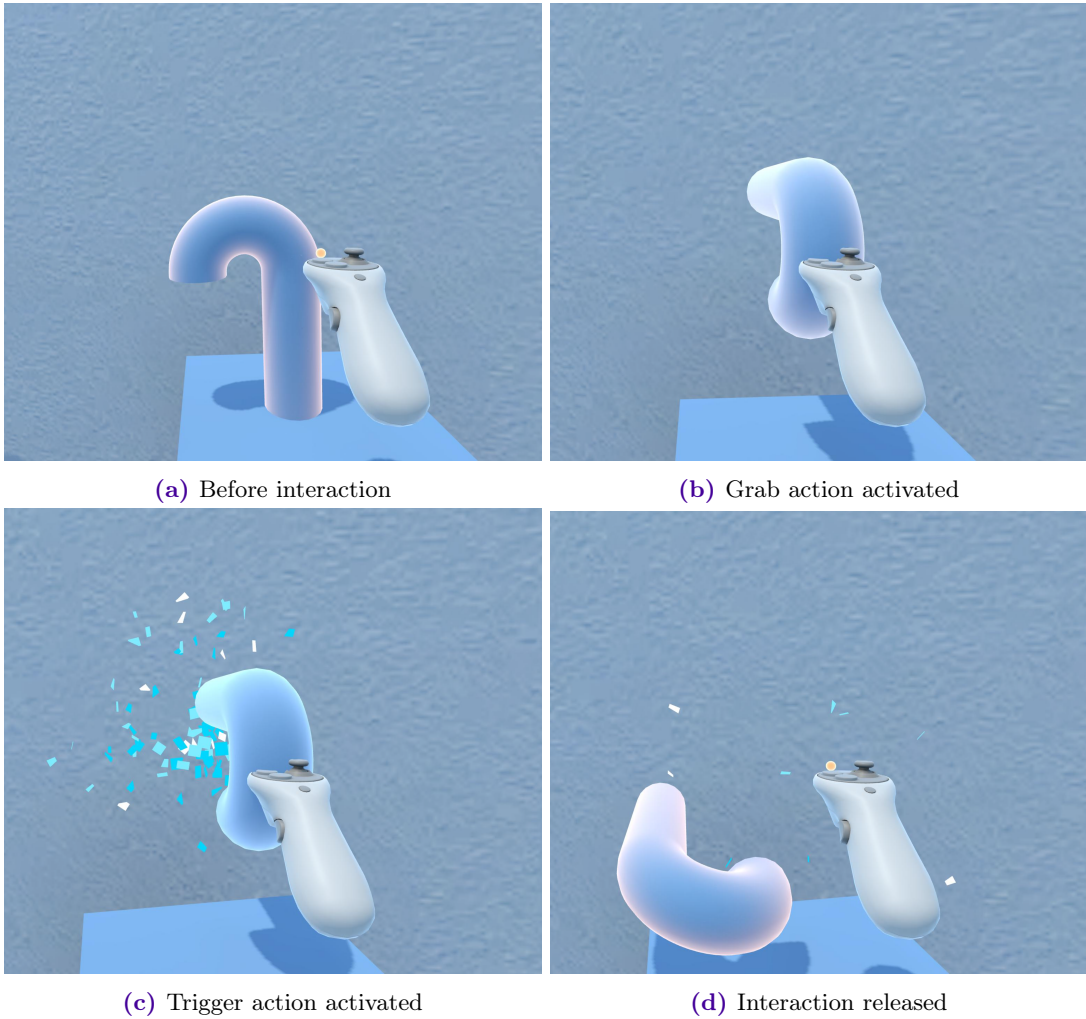


Figure 5.5: Examples of Compositional Interaction Sequences

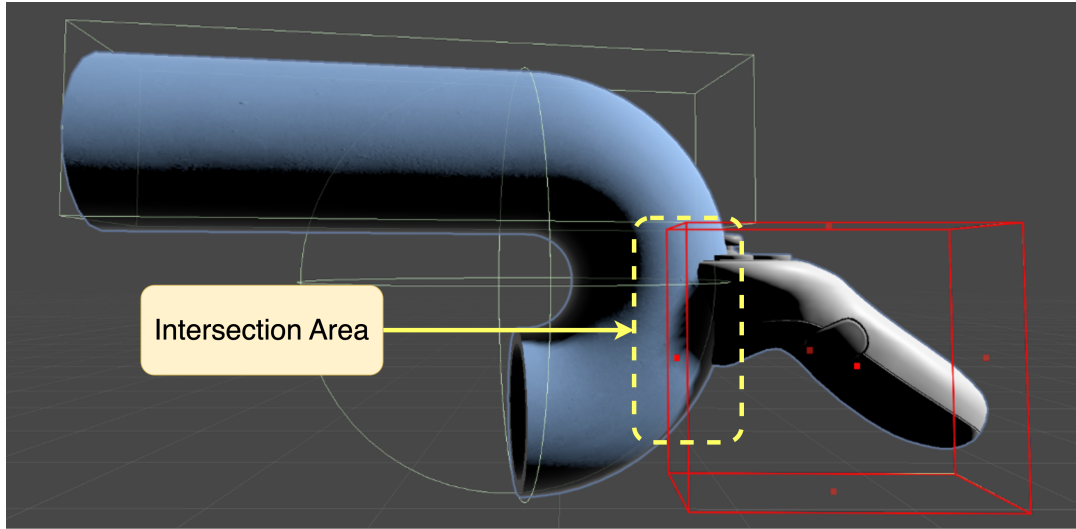


Figure 5.6: Intersection of Colliders of the Target Interactable (Green Bounds) and Controller (Red Bounds)

continuously monitors runtime exceptions, which are recorded along with contextual information about the target interaction being attempted at that moment. While detecting exceptions is valuable, comprehensive testing approaches should also identify non-exception bugs that compromise user experience without evoking errors [145].

To address such non-exception bugs, we developed a test oracle to detect subtle problems related to user interactions. During our manual exploration of the subject XR scenes (Section 5.1), we observed a recurring pattern where users correctly positioned controllers and *initiated* expected interactions but failed to *activate* them (discussed in Section 5.2). We refer to this problem as *unresponsive interaction issues*. This insight is leveraged to inform our test oracle, which specifically targets the detection of unresponsive interactions.

Our oracle leverages collision detection principles [137, 146], and is grounded with XRI’s requirement that each interactable must possess at least one collider component (Section 2.2.1). In this setting, we consider that interactions can be activated whenever the collider of the controller intersects with the collider of the target interactable. Figure 5.6 illustrates such a situation. The target object (the candy-cane-shaped item) is enclosed by green bounds representing its collider, while the controller is enclosed by a red bounding box. The yellow dashed region marks the volume where these colliders intersect, indicating that the controller is in contact with the interactable and that the interaction should be able to proceed. During test execution, the oracle continuously monitors the collider states of controllers and interactables. When it observes a valid intersection between a controller and a target interactable, it expects the corresponding interaction to activate and produce the intended behaviour.

We implement a simple yet effective rule: if a valid collision occurs but the interaction does not activate the expected behaviour (for example, the object is not grabbed), the oracle flags this as an unresponsive interaction issue for that interactable. This mechanism reliably detects objects that appear interactive in the scene but fail to respond to correct user input, ensuring such subtle faults are systematically uncovered.

To capture detailed testing results, upon the completion of a test session that attempts to interact

with all interactables in the scene, XRINTTEST generates a test report containing:

- The interaction flow coverage of the scene (detailed in Section 5.3.4);
- Interactables flagged as containing bugs by the collision-based oracle mechanism;
- Any exceptions encountered during testing.

Based on the current implementation of test oracle for non-responsive interactions, XRINTTEST can also be extended to integrate additional validation mechanisms for different testing aspects or test oracles, such as spatial UI interaction and locomotion.

5.3.4 Interaction Flow Coverage

Traditional code coverage metrics provide useful quantitative information but are limited in what they capture. For XR applications, focusing solely on code coverage risks overlooking configuration-dependent functionality, leading to a misleading assessment of test quality. On top of the discussion regarding XR test case quality in Section 4.5.1, we argue that Unity-based XR apps, in particular, derive their runtime behaviour from both source code and rich scene configurations, such as object hierarchies, colliders, and interaction components [121].

Unity’s native code coverage⁵ primarily reports which methods or lines in user scripts and engine libraries have been executed, including subsystems such as physics and rendering. However, executing code related to an interaction (e.g., an interaction handler method) does not guarantee that the corresponding interaction was correctly activated or functionally validated. As a result, even high code coverage cannot ensure that all intended XR interactions have been triggered and observed, especially when their activation depends on specific scene configurations and spatial conditions.

Video game testing metrics like interactable object coverage [147] quantify the number of activated interactable objects over the total available. However, since our IFG emphasises *interactions* rather than objects, these existing metrics are not sufficient for evaluating diverse VR behaviours, as one object can support multiple interactions (Section 5.1). Covering an object does not imply that all its interactions are covered. For example, if an object supports both *fire* and *manipulate* interactions, executing only one means the object is covered, but its interaction coverage remains incomplete. Conversely, if all interactions across a scene are covered, full object coverage is inherently achieved.

To systematically evaluate XRINTTEST’s effectiveness, we introduce **Interaction Flow Coverage (IFC)**, measuring the proportion of activated interactions within a scene. Unlike object-based metrics, *IFC* directly captures coverage of interaction flows. We define *IFC* as the number of interaction flows activated by tests divided by the total flows in the ScUT’s IFG. Given trigger and grab actions are foundational in VR interaction, our evaluation emphasises the *IFC* of interactions involving these actions, specifically *fire* and *manipulate* (Section 5.1).

5.4 Evaluation

To assess the performance of XRINTTEST, we conducted an evaluation based on the XRBENCH3D benchmark, guided by the following research questions:

⁵<https://docs.unity3d.com/Packages/com.unity.testtools.codecoverage@1.2/manual/index.html>

RQ1 (Coverage): How effectively does XRINTTEST achieve Interaction Flow Coverage on foundational VR actions?

RQ2 (Efficiency): How efficiently does XRINTTEST achieve Interaction Flow Coverage on foundational VR actions?

RQ3 (Bug Detection): How well can XRINTTEST detect bugs in VR apps?

5.4.1 Evaluation Setup

The evaluation focuses on *fire* and *manipulate* interactions (Section 5.3). This scope enables valid comparisons, as these standardised interactions are adopted consistently across XR apps. These serve as meaningful targets for automated testing, allowing for fair evaluation of testing performance. The implementation of XRINTTEST and the XRINTTEST dataset can be accessed at: <https://github.com/ruizhengu/XRintTest> and <https://github.com/ruizhengu/XRBench3D>, respectively.

Baseline

We implemented a baseline variant of XRINTTEST using a purely random testing strategy, conceptually similar to monkey testing in mobile apps [148]. This baseline randomly selects and executes atomic VR actions at 0.1-second intervals, ensuring sufficient action executions for interaction coverage while allowing previous actions to complete. For each test execution, the simulated user is positioned at a random spawn point to start navigating the scene. To enable composite interactions, we adopted a random key press duration (0.1 to 0.5 seconds). Additionally, to prevent testing inefficiencies caused by drifting beyond relevant testing areas, we implemented a position reset function with a weighted probability distribution that favours continuing interactions over position resets.

Following the motivation from Section 4.1, we deliberately excluded the prior VR scene testing tools available when this evaluation was designed, VRTEST [54] and VRGUIDE [55], from our evaluation to preserve methodological validity. These tools can navigate VR scenes and trigger certain events, but their support is largely restricted to conventional UI elements, and they do not handle 3D spatial interactions. Moreover, they simulate mouse-based clicks rather than spatial XR inputs (e.g., 6-DoF controller gestures), making them fundamentally incompatible with the type of 3D interactions targeted in this work (see Section 4.1).

In fact, the fundamental incompatibility of VRTEST and VRGUIDE to address modern XR environments was empirically confirmed during our experiments. When we applied these tools to our XRINTTEST dataset, they yielded 0% *IFC*, indicating that they were unable to exercise any of the 3D spatial interactions of interest. This outcome suggests not just limited effectiveness, but a technical mismatch with the interaction paradigm that our framework addresses.

A recent related work, VRExplorer [149], was published after the main work and evaluation of this thesis had been conducted. VRExplorer proposes a model-based, semi-automated approach for testing VR scenes and is relevant to this thesis because it also uses scene models to guide testing. However, it addresses a different evaluation setting: it requires instrumentation and triggers interactions through programmatic calls, whereas XRINTTEST evaluates end-to-end execution by synthesising controller-based XR inputs and observing runtime interaction outcomes. Including VRExplorer would therefore

Table 5.2: Effectiveness of XRINTTEST on Interaction Flow Coverage

ID	Fire	Random Baseline			Fire	XRINTTEST		
		Manip.	Socket	Total		Manip.	Socket	Total
1	0%	45%	N/A	40%	100%	100%	N/A	100%
2	0%	43%	N/A	33%	100%	100%	N/A	100%
3	0%	2%	N/A	2%	100%	96%	N/A	97%
4	0%	2%	0%	2%	100%	100%	100%	100%
5	0%	31%	0%	28%	100%	100%	100%	100%
6	0%	45%	0%	29%	100%	100%	100%	100%
7	0%	9%	0%	8%	93%	95%	100%	95%
8	N/A	40%	0%	20%	N/A	100%	100%	100%
9	N/A	0%	N/A	0%	N/A	100%	N/A	100%
10	0%	0%	0%	0%	100%	91%	75%	83%
Total	0%	12%	0%	8%	97%	94%	83%	93%

require a separate experimental setup and would answer a different comparison question. We consequently treat it as complementary late-breaking work and a useful direction for future comparative studies, rather than as a baseline for the evaluation reported here.

In light of these findings, it is important to clarify the rationale behind omitting VRTEST and VRGUIDE from our evaluation. While a comprehensive solution should ultimately support both traditional UI and 3D spatial interactions, our work is specifically motivated by the observation that 3D spatial interaction remains essentially unaddressed by existing automated testing tools. The primary contribution of this work is to close this gap. Excluding VRTEST and VRGUIDE from the evaluation is therefore not an attempt to sidestep UI-oriented baselines, but a recognition that they operate in a different problem space and are unsuitable for a meaningful comparison on the 3D interaction problem studied here.

Regarding the use of *IFC* instead of interactable object coverage, interacting with an object does not guarantee all its interactions are covered (Section 5.3.4). Specifically, our subject scenes contain 49 *fire* interactions (Section 5.1), and all of them are associated with *manipulate*. If we only measure interactable objects (e.g., by activating only the *manipulate* interactions), we would miss all 49 *fire* interactions, resulting in incomplete testing coverage despite full object coverage.

Running Environment

To account for the variability of Unity’s nondeterministic processes (particularly in physical simulations common to VR), we report the average results from five independent executions of XRINTTEST and the random baseline for each subject scene. This ensures more reliable measurements, despite identical input behaviours that may yield different outcomes. Notably, all research questions were addressed using this unified experimental design.

We run our experiments on a MacBook Pro with M3 Pro Chip, 36 GB of RAM, macOS 15.4.1, Unity 6000.0.61f1, and XRI v3.1.2. All comparisons used identical settings: Unity ran at 2x simulation speed, and each run was allocated ten simulated minutes, corresponding to five minutes of wall-clock time.

5.4.2 RQ1: Coverage

Table 5.2 demonstrates the effectiveness of the random baseline and XRINTTEST in covering VR user interactions. The random baseline failed to achieve any successful interaction in the *fire* (0/55, 0%) and *socket* (0/47, 0%) categories, showing that composite interaction patterns are particularly difficult

for random exploration. For *manipulate* interactions, the random baseline achieved only 12% coverage (31.6/271), resulting in an overall coverage of 8% (31.6/373). This gap is expected to some extent because random action selection is poorly suited to spatial interactions that require precise positioning and ordered input sequences. We nevertheless include it as a lower-bound baseline analogous to monkey testing. More informed baselines would require support for 3D controller-based interactions, which existing VR testing tools do not provide.

XRINTTEST significantly outperformed the random baseline across all subject scenes and interaction categories, achieving 97% (53.6/55) for *fire*, 94% (254.6/271) for *manipulate*, and 83% (39/47) for *socket* interactions, yielding an overall coverage of 93% (347.2/373). The substantial gap highlights the effectiveness of XRINTTEST compared to random exploration, indicating approximately 12 times greater overall effectiveness. The remaining uncovered interactions mainly arise from limitations in reaching or activating some objects under dynamic scene conditions, including objects affected by physics, runtime state changes, or socket configurations. Thus, 100% IFC refers to all interactions represented in the static IFG, but some of these may become difficult or impossible to exercise during a particular execution. The reasons for missed interactions and the relatively lower *socket* interaction coverage will both be discussed in Section 5.6.3.

These results align with and extend previous findings from VRTest [54]. Its greedy approach, VRGreed, achieved $\sim 80\%$ coverage for interactable objects across five VR scenes (with 100% in two scenes), while its pure random VRMonkey achieved only $\sim 10\%$ (0% in three scenes). Our work amplifies this effectiveness gap, as our subject VR apps involve more complex spatial interactions that require precise navigation and manipulation, making random approaches particularly ineffective at locating and engaging with target interactables.

Answer to RQ1: Across 373 *fire*, *manipulate*, and *socket* interactions, XRINTTEST demonstrated great effectiveness with 93% coverage, vastly outperforming the random baseline which achieved only 8% coverage overall.

5.4.3 RQ2: Efficiency

Figure 5.7 compares the efficiency of XRINTTEST and the random baseline in achieving *IFC* across ten subject scenes, averaged from five executions, and the last chart shows the averaged results for all scenes. The results show XRINTTEST's high efficiency across all subject scenes. Notably, in scenes 1, 2, 5, 6, 8, and 10, it rapidly achieved nearly 100% *IFC* within the first two minutes of the test sessions. For the rest of the scenes, we observe a more gradual coverage gain due to the significantly larger number of interactions they contain. However, XRINTTEST still completed the test sessions and achieved high coverage within the allocated time budget. We did not evaluate longer execution budgets because the aim was to compare approaches under a practical fixed testing budget. The plateauing curves in Figure 5.7 suggest that additional time alone would not necessarily resolve all missed interactions.

In contrast, the random baseline exhibited slower interaction coverage efficiency. While showing continuous improvement over time in several scenes, it struggles to achieve over 40% coverage in the allocated time. As shown in Figure 5.7, the last averaged chart illustrates this significant difference in efficiency, with XRINTTEST reaching approximately 95% coverage at the midpoint of the allocated time while the random baseline achieves less than 15% coverage in the same timeframe.

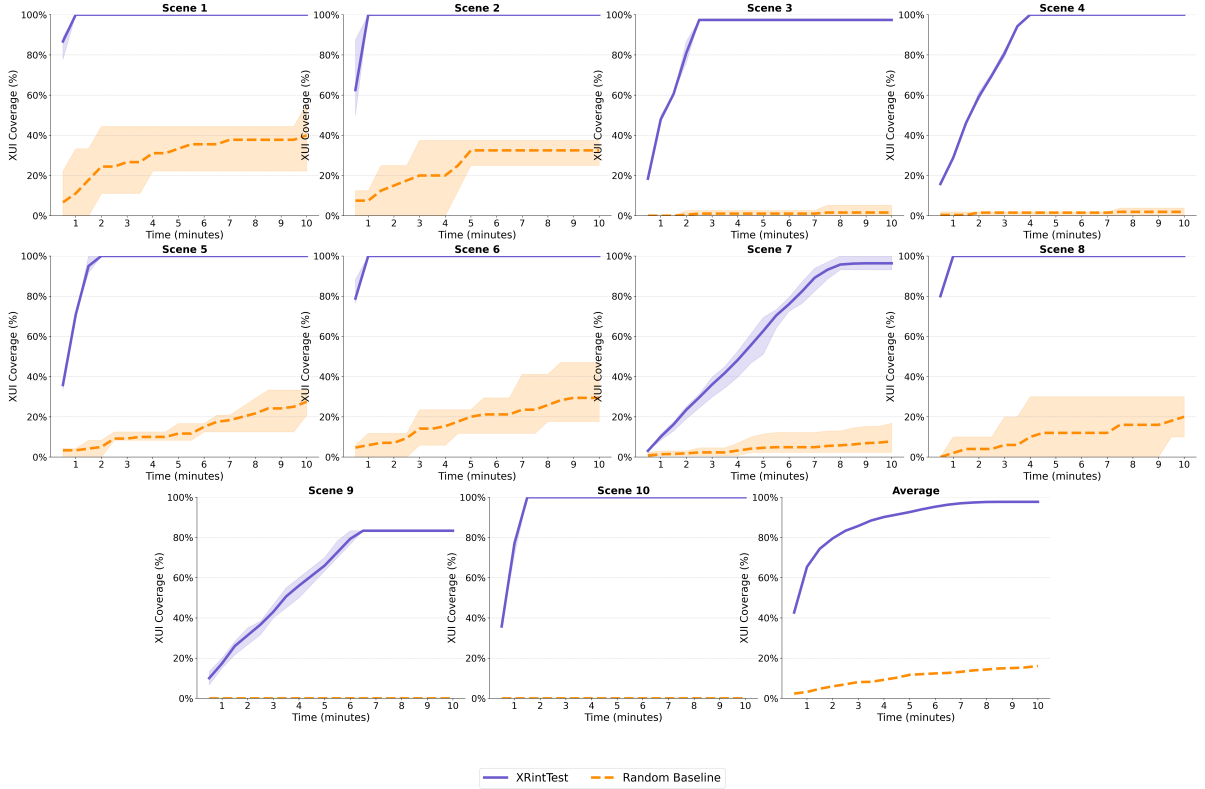


Figure 5.7: Efficiency of XRINTTEST in Achieving VR User Interaction Coverage

The steeper coverage curves of XRINTTEST across all scenes highlight its ability for rapid testing of the interactions in VR scenes. The efficiency advantage is particularly important for testing large, complex scenes (e.g., scene 7 in our dataset), with high spatial complexity and interaction density.

Answer to RQ2: XRINTTEST achieves high coverage within 2 minutes in most scenes, reaching near full coverage within the allocated time for all scenes, consistently outperforming the random baseline across subjects.

5.4.4 RQ3: Bug Detection

Both XRINTTEST and the random baseline successfully detected the same exception in scene 3, demonstrating comparable ability in identifying runtime errors. We traced the exception to an implementation issue and reported it to the project’s maintainers. Furthermore, XRINTTEST’s test result checker (Section 5.3.3) identified two unresponsive interaction issues among the failed *manipulate* interactions for scene 7. Notably, one of the unresponsive interactions *had not been identified during our manual exploration of subject scenes*. This finding highlights XRINTTEST’s bug detection capabilities, which in this instance surpassed human testing efforts.

Our root cause analysis revealed a key distinction between VR interaction issues. While the runtime exception in scene 3 stemmed from an implementation error in the source code, the two unresponsive interaction issues resulted from misconfigured object properties. Specifically, these issues were caused

by misconfiguration of the objects' *collider* properties (Section 2.2.1). These interactables had their `Collider.isTrigger` parameter enabled, which prevents physical collision and instead enables specific event triggering (e.g., function calls, typically for UI elements). Consequently, these objects could not physically engage with other objects (e.g., controllers), rendering them impossible to interact with despite having proper interaction definitions. To ensure validity, we are communicating with the respective VR project developers to confirm the identified issues⁶.

We did not seed artificial faults or scene mutations in this evaluation. The reported issues were detected in existing benchmark scenes. Seeded faults would provide a more controlled way to measure fault-detection capability and are a useful direction for future work, especially for configuration-level XR faults.

These findings underscore the importance of testing approaches that can identify configuration-related defects in addition to traditional code-level or runtime exception bugs. This is especially important for VR apps, where physics-based interactions and realism are fundamental to user experience, but effective exploratory testing is challenging for human testers.

Answer to RQ3: Both XRINTTEST and the random baseline detected the same runtime exception, but only XRINTTEST found two additional *unresponsive interaction* bugs attributable to misconfigured object properties, one of which had been missed during manual exploration.

5.5 Answers to ORQ4, ORQ5 and ORQ6

This chapter provides the answers to ORQ4, ORQ5 and ORQ6 from Section 1.1.

5.5.1 ORQ4: Prevalent Interaction Types

ORQ4: *What XR interaction types are prevalent in open-source XR projects?*

Most prevalent XR interaction types in open-source XR projects are built from grab and trigger actions, with grab-based *manipulation* dominating overall. In the analysed benchmark XRBENCH3D, composite patterns such as *fire* (grab-and-fire), *socket*, and bespoke *custom* interactions are also common.

Empirical Interaction Distribution

Across the ten subject scenes in XRBENCH3D, manual inspection and in-headset validation identified 456 distinct 3D interactions in total. These interactions were categorised into four groups: *manipulate*, *fire*, *socket*, and *custom*, each corresponding to a characteristic XR interaction mechanism grounded with Unity's XR Interaction Toolkit, a high-level interaction framework for developing XR experiences.

We detail the distributions and the descriptions of these interaction types as follows:

- **Manipulation** (271/456, 59%): direct object manipulation initiated via the controller's grab action, optionally combined with controller translation or rotation for more complex movement.

⁶GitHub issue: <https://github.com/mikeNspired/XRI-Starter-Kit/issues/36>

- **Fire** (55/456, 12%): composite “grab-and-fire” patterns, where the user must first maintain a persistent grab on an object (e.g., a gun) and then use the trigger to activate a firing action. Furthermore, all trigger actions in XRBENCH3D occur under an active grab, underscoring the dominance of this composite pattern.
- **Socket** (47/456, 10%): interactions involving **XR Socket Interactor** components, in which the user continuously grabs an object and positions it into a designated socket (e.g., a key into a keyhole), representing deliberate, spatially constrained placement rather than autonomous physics behaviour.
- **Custom** (83/456, 18%): scenario-specific interactions that layer additional rules or constraints on top of grab and trigger primitives, such as knobs that rotate along a fixed axis while grabbed.

Prevalent Primitives and Patterns

The *manipulate* and *fire* groups are both directly composed of the foundational grab and trigger actions. Taken together, they account for 71% of all observed interactions (326/456), highlighting their central role in contemporary XR projects. Socket and custom interactions frequently reuse the same primitives in more constrained or domain-specific patterns, suggesting that higher-level XR interaction designs are typically constructed as compositions over these core actions. This observation aligns with the taxonomy of XR user interactions defined in Section 4.2, where complex interactions are modelled as compositions of fundamental atomic actions.

5.5.2 ORQ5: XR Interaction Modelling

ORQ5: *How can interactions among objects in XR scenes be systematically modelled to support automated test generation?*

User interactions in XR environments can be systematically modelled for automated testing by abstracting them into a graph-based representation that captures users, interactables, and their possible interaction flows. In this work, this is realised through the **Interaction Flow Graph (IFG)**, which serves as a behavioural model for test generation and execution.

Graph-based Interaction Abstraction

The IFG models an XR scene as a directed graph whose nodes represent the interactor (user) and all interactable objects, while edges encode the interaction types that can occur between them (e.g., grab, fire). Each interactable node is identified through static analysis of the Unity scene (Section 5.3.1), where interaction scripts such as **XRGrabInteractable** or custom components are mapped to corresponding interaction types. This graph abstracts away raw implementation details and provides a concise, testable view of which interactions are possible and how they connect across the scene.

From Taxonomy to Testable Actions

The IFG is grounded in the three-layer taxonomy of XR user interactions defined in Chapter 4.2, which distinguishes core tasks, atomic actions (e.g., grab, trigger), and compositional patterns (e.g., grab-and-fire, socket placement). Each edge in the IFG corresponds to an interaction formed by a set of

atomic actions with specific patterns, which can be directly mapped to test actions in the automation framework. This mapping allows high-level user intents (core tasks) to be decomposed into sequences of low-level controller operations, enabling systematic, automatic construction of test scripts that exercise both primitive and composite interactions.

Informing Automated Exploration

Because the IFG is a structural model of all detectable interactions in the scene, it can be used to guide automated exploration strategies (e.g., path traversal over interaction edges) and to define coverage criteria over XR interactions rather than just code. In this way, the IFG offers a systematic, scene-aware foundation that connects interaction design and automated test generation within XR environments.

5.5.3 ORQ6: Design and Evaluation of Automated XR Testing

ORQ6: *How can automated testing strategies be designed and evaluated to effectively assess XR user interactions?*

The automated testing approach developed in this thesis is designed to systematically evaluate XR user interactions through model-based execution and interaction simulation. XRINTTEST, integrates the graph model of user interactions with automated test generation to ensure sufficient coverage of various scenarios.

Effectiveness (Coverage of Interactions)

Across 373 target interactions (*manipulate* and *fire*), the random baseline achieved only 8% **Interaction Flow Coverage (IFC)** overall (31.6/373), with 12% for *manipulate* (31.6/271) and 0% for *fire* (0/55) and *socket* (0/47), indicating that unguided exploration struggles particularly with composite interaction patterns such as grab-and-fire. In contrast, XRINTTEST achieved 93% coverage overall (347.2/373), with 94% for *manipulate* (254.6/271), 97% for *fire* (53.6/55) and 83% for *socket* (39/47), making it roughly 12 times more effective than the random baseline in exercising intended XR interactions. These results reinforce and extend prior evidence from VRTEST, where a guided strategy VRGREED similarly outperformed random exploration in simpler XR scenes.

Efficiency (Speed to Achieve Coverage)

Efficiency analysis reveals that XRINTTEST reaches a high IFC substantially faster than the random baseline. In six of the ten evaluated scenes, it achieved nearly 100% IFC within the first two minutes of execution. In the remaining, more complex scenes with larger interaction sets and spatial density, it continued to progress steadily to near-full coverage within the overall time budget. By contrast, the random baseline rarely exceeded 40% coverage within the same duration, and at the midpoint typically remained below 20%. The consistently steeper coverage curves of XRINTTEST demonstrate its ability to efficiently navigate dense and interaction-rich XR environments, supporting its use for rapid iterative testing cycles.

Bug Detection (Types of Issues Found)

In terms of fault detection, both XRINTTEST and the random baseline identified the same runtime exception in one test scene, indicating comparable efficacy in exposing overt code-level errors. However, XRINTTEST further detected two additional *unresponsive interaction* issues in Scene 7 that were missed by both the baseline and by manual exploratory testing. These defects stemmed from misconfigured object collider properties (specifically `Collider.isTrigger` enabled), which prevented physical interactions despite correct interaction definitions. This demonstrates XRINTTEST’s capability to identify subtle configuration-level faults that compromise interactive realism but might remain undetected through manual testing or conventional automation.

5.6 Discussion

5.6.1 Interaction Design Smells

Design smells indicate potential problems in software. For instance, code smells and test smells signal issues in source and test code [150, 151]. Similarly, UI design smells violate guidelines in complex systems, such as Android apps [152], often involving elements that can be dismissed unexpectedly or hinder access to key information.

In our findings, we consider setting socket interactors’ Interaction Layer Masks to “Everything” as a potential interaction design smell that violates intended functionalities. We are engaging with developers of affected VR projects to confirm this as a bad practice⁷.

Furthermore, we found that the IFG can be used to effectively detect this type of interaction design smell. The IFG identifies interaction brokerages (Section 5.2.3) by matching interactables and socket interactions that share the same Interaction Layer Masks. Normal interactables (i.e., those not paired with sockets) typically use the “Everything” layer. When socket interactors also default to “Everything”, the IFG generates an explosion of unintended edges, such as linking all nodes to a “Door” interactable in Figure 5.2b. This misconfiguration could render the graph excessively dense and uninformative. For testing, this increases the number of candidate interactions and can direct the explorer toward interactions that are permitted by configuration but unintended. It can therefore reduce testing efficiency and make coverage results harder to interpret, because covering these edges may not correspond to meaningful application behaviour.

We now discuss key insights from this chapter, highlighting the significant advancements and remaining challenges.

5.6.2 Oracle Automation

Test oracle automation represents a major challenge in VR automated testing [13]. The current implementation of XRINTTEST employs a contract-based oracle automation approach that validates object properties as postconditions [37]. Its generalised mechanism has been proven effective without requiring manual intervention or scene-specific customisation. While effective, this approach can only validate interaction activation, missing semantic interaction failures where interactions technically succeed but produce incorrect behaviours. For instance, a movable object may respond to grab actions but cannot

⁷GitHub issue: <https://github.com/Unity-Technologies/XR-Interaction-Toolkit-Examples/issues/140>

actually be moved *as intended*, representing an opportunity for improvement. For instance, a movable object may respond to grab actions but fail to move along its intended path or within its intended constraints. Detecting this kind of fault would require richer oracles that check post-interaction state changes, such as object position, rotation, constraint satisfaction, or task-specific outcomes, rather than only confirming that an interaction event was activated.

The complexity and variability of VR interactions make it particularly challenging to determine expected outcomes programmatically. While metamorphic testing has shown promising results in assessing VR runtime requirements [137], our evaluation revealed the difficulty of defining meaningful metamorphic relationships for diverse interaction patterns and their resulting behaviours. The challenge lies not only in identifying appropriate metamorphic relationships but also in efficiently validating them across the wide spectrum of possible VR app behaviours. Existing image-based techniques have been proposed for detecting visual or spatial inconsistencies in XR applications [153]. However, we argue that such approaches may be inefficient for automated testing purposes, as they rely heavily on image processing and computer vision analysis during test execution. These operations introduce substantial runtime overhead, making them less practical for large-scale or continuous testing workflows. Developing alternative verification strategies that balance generality, efficiency, and interpretability, therefore, remains an open research challenge for future work in this area.

5.6.3 Environment-Aware Testing

XRINTTEST currently focuses on intended user interactions, and our evaluation demonstrates effective detection of direct interaction failures (Section 5.4.4). However, an emerging challenge has been identified where complex environmental dynamics may indirectly affect testability. While testing scene 7, we observed instances where interactions became permanently inaccessible after being destroyed through physics-based interactions inadvertently caused by test actions.

The challenge lies in XRINTTEST’s current focus on isolated interaction validation without considering the cascading effects of actions within the VR environment. While our oracle successfully detects when individual interactions fail, it cannot predict or prevent scenarios where successful interactions lead to environmental changes that compromise subsequent testing. For instance, a physics simulation that correctly responds to user input may inadvertently destroy other interactable objects. This limitation suggests the need for more comprehensive environment modelling that accounts for the complex interplay between objects, physical systems, and scene-wide changes that characterise realistic XR environments.

The challenges stem from the lack of comprehensive characterisations of XR-specific properties that may affect testing, which require extensive domain knowledge. The existing literature reveals two distinct but incomplete perspectives on the XR domain knowledge: the human-computer interaction research focuses on interaction design while neglecting implementation-level test concerns [154], and software engineering approaches address technical issues but overlook the nature of spatial interactions [155]. Bridging these perspectives is essential for developing comprehensive testing solutions.

Regarding the issue with runtime destroyed objects, a more robust understanding would link the relationship among *object properties* (e.g., physical properties for destructivity), *interaction modalities* (e.g., avoiding destructive actions), and *system functionality* (e.g., verifying the functional correctness of both individual objects and the XR scene as a whole). By gaining such insights, XRINTTEST and

similar approaches would be able to achieve more thorough XR testing.

The limitation of dynamic scene is important to note because Unity projects commonly use prefabs and runtime instantiation to reuse, spawn, or remove interactable objects during execution. A more complete approach would need to resolve prefab references, inspect prefab variants, and analyse scripts or configuration files that instantiate or destroy objects at runtime. The current implementation therefore measures coverage relative to the static scene model rather than all possible runtime states.

5.7 Conclusion

This chapter tackled key challenges in automated XR app testing by systematically modelling interactions within XR scenes and automatically generating 3D user interactions [138, 139]. It began with an empirical analysis of ten diverse XR scenes in XRBENCH3D, identifying 456 interactions across four categories (*fire*, *manipulate*, *socket*, and *custom*) that characterise prevalent 3D interaction patterns.

Building on these insights, we introduced the **Interaction Flow Graph (IFG)**, a graph-based model of 3D user interactions in VR scenes that underpins XRINTTEST, our automated testing framework for systematically exploring scenes and activating the interactions identified in the model. Evaluation on XRBENCH3D shows that XRINTTEST achieves 93% interaction flow coverage, outperforming a random baseline by roughly a factor of 12 in effectiveness and about six in efficiency, while also detecting both code-level and configuration-related interaction issues.

While the current approach effectively covers foundational XR interactions, particularly the *manipulation* and *fire* categories, many bespoke, higher-level interaction designs remain outside the present scope. Further challenges, such as richer oracle automation and environment-aware testing, are discussed in Section 5.6, and additional research directions and potential solutions are outlined in detail in Section 7.2. The reported 93% IFC should therefore be interpreted relative to the statically constructed IFG used in this evaluation. Runtime object creation, object destruction, prefab-instantiated behaviours, and changing scene states may introduce additional interactions or alter which interactions are available during execution. Extending the IFG to capture these dynamic behaviours would further improve coverage interpretation and support fairer comparisons between scenes in future work.

Chapter 6

Envisioning the Future of XR Device Software and Testing

This chapter looks beyond the current generation of XR [Head-mounted Display \(HMD\)](#) and the studies in earlier chapters to consider where XR software and its testing are heading next. Building on the systematic mapping of XR testing ([Chapter 3](#)) and the concrete automation techniques for XR interaction testing ([Chapters 4 and 5](#)), the focus now shifts from today’s scene-centric XR apps to emerging, always-on XR devices integrated into everyday life. [Chapter 5](#) showed how XR interactions can be modelled and exercised automatically. This chapter extends that trajectory by asking how software engineering and testing practices may need to evolve when XR systems become persistent, AI-mediated, and embedded in open physical environments.

Academic and industry analyses indicate a convergence between spatial computing on XR HMDs and AI capabilities in smart glasses, giving rise to a new class of *AI-powered everyday XR glasses* (hereafter, XR glasses) [[2](#)]. These devices seamlessly superimpose digital content onto physical environments to generate realistic MR experiences. Some consider them as a plausible successor to smartphones, possibly leading to significant changes in human-digital interaction and social communication [[4](#)]. Industry prototypes such as Meta Orion¹ and XREAL’s Project Aura² exemplify the trajectory of fusing multimodal AI with XR capabilities for imminent commercial release. Crucially, their integration with large [FMs](#) enables sophisticated contextual understanding and adaptive content and behaviour generation that extends beyond traditional task-specific AI features [[156](#)].

Despite rapid industrial progress, academic work on XR glasses remains concentrated in AI and [Human-Computer Interaction \(HCI\)](#), with emphasis on topics such as multimodal, context-aware perception (e.g., gaze prediction [[157](#)] and social gaze understanding) and spatial interaction design frameworks [[158](#)]. In contrast, the [Software Engineering \(SE\)](#) facets of these systems—covering architecture, evolution, reliability, and testing of FM-enhanced, embodied spatial applications—are still in their infancy. Existing XR SE work on requirements, testing, and quality assurance provides important foundations but largely targets earlier XR platforms and does not yet fully address the complexities introduced by always-on, AI-driven XR glasses [[13](#), [159](#)].

¹<https://www.meta.com/emerging-tech/orion/>

²<https://www.xreal.com/aura>

Positioned against this backdrop, this chapter frames AI-centric software engineering for XR glasses as a distinct and timely research frontier and offers a first foundational roadmap for this space. By synthesising insights from spatial computing, intelligent systems, and prior chapters' findings on XR testing and interaction automation, the chapter:

1. Establishes a conceptual overview of XR glasses as a new class of intelligent, context-aware software systems.
2. Maps their unique computational and interaction capabilities into a unified AI-SE perspective, extending the interaction and testing paradigms introduced earlier in the thesis.
3. Identifies possible interdisciplinary research problems at the intersection of AI, XR, and SE, with particular attention to testing and assurance in FM-enabled, dynamic, interactive environments.
4. Highlights potential societal and ethical implications arising from pervasive, AI-mediated spatial experiences.

By proactively exploring these dimensions, the chapter aims to equip AI-SE researchers and practitioners with a forward-looking lens on how XR device software and its testing must evolve in the era of AI-powered XR glasses.

This chapter is structured as follows. Section 6.1 introduces a conceptual framework that characterises XR glasses as integrated software systems combining advanced spatial and AI capabilities. Section 6.2 examines key capabilities that distinguish XR glasses from other computing platforms. Section 6.3 outlines emerging research problems for XR glasses from both AI and SE perspectives. Section 6.4 discusses broader societal and ethical challenges beyond purely technical concerns. Section 6.5 answers ORQ7 from Section 1.1. Section 6.6 concludes this chapter.

The content of this chapter is based on “Ruizhen Gu, Jingqiong Zhang, José Miguel Rojas and Donghwan Shin. On the Promises and Challenges of AI-Powered XR Glasses as Embodied Software, in *Proceedings of the 2nd ACM/IEEE International Conference on AI-powered Software*, 2025.” [3].

6.1 Conceptual Framework

Figure 6.1 presents a conceptual framework that characterises XR glasses as integrated software systems combining advanced spatial computing with AI-driven intelligence. The framework is designed to capture the unique architectural and interaction paradigms of XR glasses, and to provide a basis for reasoning about their development, operation, and testing.

Drawing inspiration from embodied AI agents and wearable computing, our framework highlights two key distinctions. First, unlike autonomous robots that act largely independently in the world, XR glasses are worn directly by humans, forming continuous perception-action feedback loops that emphasise human-AI collaboration, in which the human and the system jointly perceive, decide, and act. Second, while they share form factors with conventional wearables (e.g., smartwatches or fitness trackers), XR glasses go significantly further through rich multimodal rendering (visual, audio, haptic) and persistent spatial awareness of the surrounding environment. For example, rather than only displaying a notification, XR glasses may anchor a visual cue to a real object, provide a spatial audio explanation, and use subtle haptic feedback to guide the user's attention.

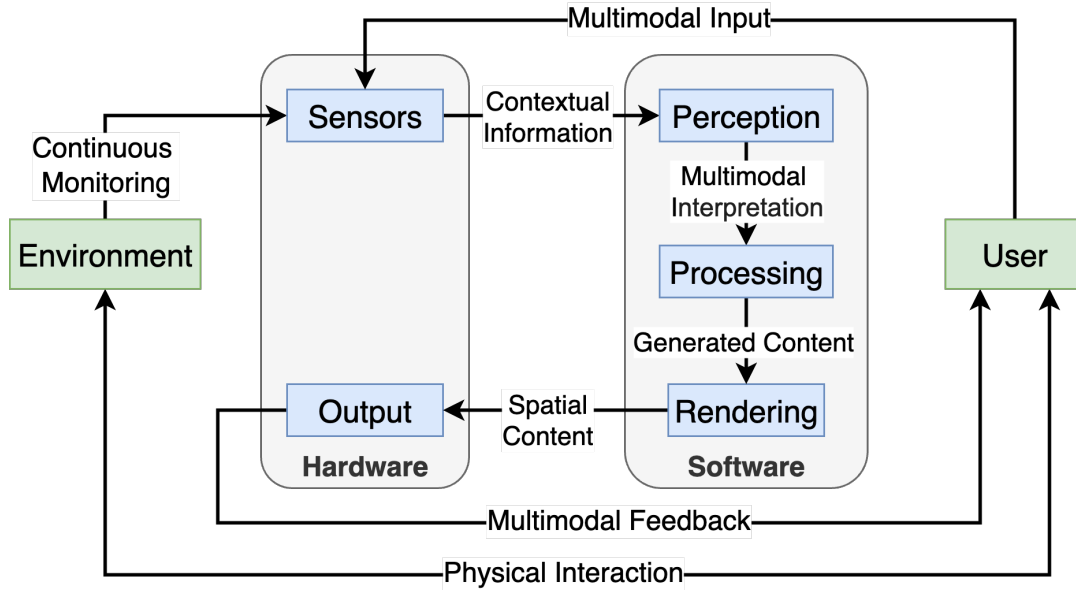


Figure 6.1: A conceptual overview of XR glasses systems

6.1.1 Perception–processing–rendering Cycle

Adapting the classic perception–planning–control paradigm from autonomous systems [160], the framework organises XR glasses into a human-centred loop with five main elements: *Sensors*, *Perception*, *Processing*, *Rendering*, and *Output*, all embedded within a continuous feedback cycle.

- The **Sensors** acquire multimodal environmental and user data, such as **Red, Green, and Blue (RGB)**/Depth camera streams, **Inertial Measurement Unit (IMU)** data, eye and head tracking, microphones, and biosignals. These streams capture both the physical environment and user state in real time.
- The **Perception** layer performs context-aware spatial interpretation over sensor data, including scene understanding (e.g., object and surface recognition), user pose and gaze estimation, activity recognition, and localisation. This layer builds and maintains an internal representation of the user’s physical surroundings and situational context.
- The **Processing** layer uses foundation models and other AI components to generate dynamic, context-aware behaviours rather than relying solely on predesigned, static content. Examples include personalised assistance, natural language understanding, summarisation of live scenes, predictive suggestions, and adaptive task support grounded in the perceived environment.
- The **Rendering** layer transforms these behaviours into concrete interactive media outputs. It selects appropriate modalities (e.g., visual overlays, audio cues, haptic feedback), arranges them spatially and temporally, and ensures coherence with the physical environment (e.g., correct anchoring of holograms, occlusion handling).
- The **Output** devices (e.g., optical see-through displays, speakers, haptic actuators) deliver the rendered content to the user, influencing their actions, attention, and decisions. For instance, a navigation assistant may render an arrow on the lens display, speak a warning, and provide a haptic cue when the user approaches an obstacle.

6.1.2 Human-in-the-loop Feedback and Collaboration

A central aspect of the framework is the explicit modelling of human-in-the-loop feedback. User responses (e.g., head and body movements, hand gestures, gaze shifts, voice commands, task decisions) immediately affect the environment and are captured by the *Sensors* as new input. This closes a continuous perception–action loop in which:

- The system’s behaviours shape the user’s perception and actions.
- The user’s actions, in turn, reshape the environment and sensor inputs.

In this loop, human cognition effectively becomes the primary *actuator*. The system (i.e., XR glasses) suggests or augments possible actions, but it is the human who ultimately interprets these suggestions, decides what to do, and carries out the action (or chooses to ignore it). This differs from traditional embodied AI, such as autonomous vehicles, where actuators like the motors directly execute the agent’s decisions. By putting human agency at the centre, the framework moves from “autonomous agents acting in the world” to “collaborative human–AI partnerships”, with XR glasses behaving more like an intelligent, context-aware co-pilot than a fully autonomous driver. Mobile phones can also provide AI assistance, but they usually require explicit attention to a handheld screen. XR glasses differ because they are hands-free, first-person, spatially registered, and continuously coupled to what the user is seeing and doing.

6.1.3 Implications for Software and Testing

By integrating AI-powered spatial intelligence with human-in-the-loop feedback, the framework is able to provide a structured lens for reasoning about XR glasses as software-intensive systems. It highlights:

- Multiple tight couplings (e.g., sensor processing, AI decision-making, spatial rendering) that must be engineered and tested in concert.
- The need to validate not only functional correctness, but also interaction quality and alignment between AI-generated behaviours, spatial context, and human expectations.

This conceptual view underpins later sections of the chapter, where XR glasses’ advanced capabilities, their AI–SE research problems, and their societal and ethical implications are examined in more depth.

6.2 Advanced Capabilities

This section examines key advanced capabilities of XR glasses that set them apart from conventional computing devices. Rather than providing an exhaustive catalogue, it focuses on a small number of representative capabilities that both unlock new interaction and application opportunities and introduce fundamentally new challenges for AI and software engineering. These capabilities form the technical basis for the research questions and problem spaces discussed in the remainder of the chapter.

6.2.1 Spatial Intelligence

Spatial intelligence is a defining capability of XR glasses that sets them apart from conventional mobile and wearable devices. While earlier AR/MR HMDs such as HoloLens³ and Varjo⁴ devices also support spatial sensing, XR glasses make this capability more persistent, lightweight, and tightly integrated with everyday AI assistance. It turns continuous streams of sensor data into a rich, actionable model of the surrounding physical environment and then uses that model to shape digital content and interactions in real time [161]. This allows devices not only to *interpret* spatial relationships but also to dynamically *generate* digital content with physical attributes (e.g., collision) [162]. In terms of our conceptual framework (Figure 6.1), these functions span the *Perception* and *Processing* layers, with foundation models (FMs) increasingly driving both spatial understanding and content generation.

Recent FM-based systems illustrate how far these capabilities can go. For example, Google DeepMind’s Genie 2 can generate fully playable virtual worlds with plausible object physics from a single image prompt [163]. For XR glasses, similar mechanisms could allow users to conjure context-aware, interactive virtual objects or scenes directly into their physical surroundings with minimal input. This moves XR glasses from merely recognising the world to actively *co-constructing* it with the user.

Spatial intelligence can be unpacked into several related aspects: streamlined environmental understanding, physics-compliant content generation, and affordance recognition. We detail them as follows.

Streamlined Environmental Understanding XR glasses can infer spatial structure and semantics directly from sensor streams, such as depth, surfaces, and object categories, without relying on extensive external tracking infrastructure [153]. This may include: (1) Reconstructing room layouts, identifying walls, floors, and furniture; (2) Estimating user pose, gaze direction, and proximity to key objects.

Such direct, FM-enhanced interpretation can reduce the need for handcrafted tracking pipelines and calibration procedures, improving portability and easing deployment across diverse environments. For software engineering, this changes how developers specify assumptions about the environment and challenges existing testing practices that assume more controlled sensing conditions.

Physics-compliant Content Generation Beyond recognising the scene, spatially intelligent XR glasses can generate digital content that respects physical laws and existing geometry [164]. For instance: (1) A virtual ball can be spawned on a real table, roll under gravity, collide with table edges, and bounce when knocked by a user’s hand gesture; (2) Virtual furniture can be placed in a room with correct scaling, occlusion, and contact with real floors and walls.

Here, FMs or specialised generative models infer both appearance and behaviour (e.g., mass, friction, elasticity), making virtual objects feel materially consistent with the surrounding world. This raises new SE questions about validating physics consistency and stability across heterogeneous hardware platforms, sensors, and environmental conditions (e.g., lighting, complexity).

Affordance Recognition Spatial intelligence also encompasses the ability to recognise *affordances*: the actionable possibilities that the environment offers [165]. XR glasses could:

- Detect that a flat horizontal surface (e.g., a table) affords placement or support.

³<https://learn.microsoft.com/hololens/>

⁴<https://varjo.com/products>

- Identify handles, buttons, or tools as graspable or operable elements.
- Infer social or task-relevant cues, such as a whiteboard being suitable for annotation.

By labelling regions with affordance roles (e.g., “place here”, “sit here”, “grab here”), the system can propose context-aware overlays and recommendations, such as suggesting where to pin virtual notes on a real wall or where to spawn a virtual control panel [166]. This enables more intuitive, expectation-aligned interactions but also requires robust, explainable models so that misidentified affordances do not lead to confusing or unsafe behaviour.

Taken together, these aspects show how spatial intelligence extends beyond static environmental *understanding* to active *generation* of content and interactions. As represented in the *Rendering* layer of our framework (Figure 6.1), XR glasses no longer just draw pre-authored 3D assets using fixed computer graphics pipelines. Instead, FMs and generative models can: (1) Synthesise new objects, interfaces, and layouts on demand, (2) Adapt content to current geometry, lighting, and user intent, and (3) Maintain physically and semantically coherent behaviour over time.

This seamless blending of real and virtual worlds opens powerful new design possibilities but also introduces substantial SE challenges. In particular, developers and testers must contend with effectively unbounded variation in real-world contexts, sensor inputs, and generated content. How to specify requirements, design test oracles, and systematically validate behaviour across such open-ended spatial scenarios is a central question that will be revisited in Sections 6.3.2.

6.2.2 Multimodal Interfaces

XR glasses introduce a new class of multimodal interfaces that tightly combine natural input and output channels in a single, always-on device. They integrate gesture, gaze, and voice input with spatial visual and auditory output, enabling users to act on virtual content as if it were part of the physical world [167]. These multimodal signals double as control mechanisms for applications and as rich context for FM-based agents, making the interface the main bridge between human intent and the system’s spatial intelligence.

Multimodal interaction becomes particularly powerful when modalities are combined rather than used in isolation. Consider a scenario where the user is looking at a flock of birds:

1. Gaze fixes the approximate region of interest in the **Field of View (FOV)** (i.e., the angular extent of the observable world that is seen at any given moment).
2. A hand gesture outlines or points to a specific group.
3. A voice query, such as “What species are these birds?” provides the task.

The XR glasses can fuse these signals, letting the FM agent understand *what* the user is referring to, *where* it is in space, and *what* the user wants to know. The system could then overlay labels or explanations directly above the birds on the lens display, while a voice assistant narrates additional details. This sequence exemplifies the interface loop described in Section 6.1: multimodal inputs (gaze, gesture, voice) initiate FM-mediated interpretation and decision-making, which then drive synchronised visual and audio outputs that complete the interaction cycle.

Beyond established interaction modes, emerging hardware innovations are redefining how input can be captured. A leading example is Meta’s *Neural Band* technology, introduced with the *Meta Ray-Ban Display*⁵ glasses. The Neural Band employs **Electromyography (EMG)** sensors [168] that detect minute electrical signals from wrist and finger muscles. Instead of large hand waves or arm movements, it allows users to achieve specific functionalities by performing subtle gestures, such as a slight pinch of the thumb and index finger to click, scroll, or select items.

Compared with traditional 2D interfaces (e.g., touchscreens or mouse-keyboard setups), spatial multimodal interfaces blur the boundaries between acting on the environment and interacting with an AI system. A single gaze shift or casual hand motion may simultaneously be a part of the user’s behaviour in the world, and a signal to the FM agents about what to interpret.

As FMs increasingly decide how to interpret and respond to these signals, questions of transparency, predictability, and user trust become central. Users need to understand when the system is monitoring, which signals it is using, and why particular responses or overlays appear. Designing interfaces that provide intuitive feedback (e.g., subtle highlights, confirmation cues) and explain system behaviour is therefore critical, and will be explored further in Section 6.3.3.

6.2.3 Distributed Software Architecture

XR glasses typically rely on a distributed software architecture that spans multiple hardware tiers, much like modern wearable and **Internet of Things (IoT)** systems [169]. This structure balances responsiveness, power constraints, and computational demands by distributing functionality across:

1. **On-device layer:** The Lightweight **Operating System (OS)** and runtime environments run directly on the glasses. They could handle (1) Real-time sensor fusion, (2) Immediate spatial processing needed for head/hand tracking, and (3) Local rendering and display management, ensuring low-latency visual and audio feedback under tight energy and thermal constraints.
2. **Edge layer:** Nearby companion devices, such as smartphones or dedicated compute packs, provide additional processing capacity. They can: (1) Offload heavier inference tasks (e.g., large FM calls for rich environmental understanding or complex language reasoning) when latency and connectivity permit, and (2) Manage cross-device coordination, notifications, and background tasks, effectively acting as services running in the background that complements the glasses’ limited hardware.
3. **Cloud layer:** Remote services offer virtually elastic resources for: (1) FM training, fine-tuning, and hosting, (2) Large-scale data storage, analytics, and user/profile modelling, and (3) Cross-user services such as collaborative experiences or global knowledge retrieval.

Together, these layers form a distributed pipeline in which data and control flow dynamically depending on network quality, privacy constraints, and performance requirements. For software engineering, this raises non-trivial questions around partitioning logic, handling partial connectivity, and testing cross-layer behaviours.

On top of this hardware stack, specialised OSs such as Android XR⁶ can serve as the primary software platform. They could expose multimodal interaction APIs (e.g., for gaze, gesture, and voice),

⁵<https://www.meta.com/ai-glasses/meta-ray-ban-display/>

⁶<https://www.android.com/xr/>

host third-party XR applications, and integrate deeply with FM-based assistants such as Gemini⁷, which may act as system-wide AI assistants for reasoning and coordination. A key architectural innovation is that Android XR provides functions that allow developers to convert traditional 2D mobile apps into spatial Android XR apps with minimal effort, accelerating ecosystem growth.

Within this distributed setting, XR glasses could adopt a transformative software paradigm: *agent-mediated interaction*. Instead of users explicitly launching and managing individual apps, intelligent agents interpret user intent, orchestrate services, and perform tasks across system boundaries. Conceptually, two main classes of FM-powered agents can coexist and collaborate:

- **OS-level agent:** Embedded deeply within the OS, this agent behaves as an enhanced, always-available assistant. It can interpret high-level, multimodal user intent (e.g., a spoken request combined with gaze context), and translate these intents into system actions such as opening spatial experiences, invoking tools, or composing cross-app workflows. For instance, it could handle a voice query that asks the system to launch Google Maps.
- **Domain-specific agents:** These agents live at the application or service layer and specialise in particular domains, such as travelling, education, or health. They are equipped with domain knowledge, tools, and workflows, and can be invoked directly by the user (e.g., a voice command of “Ask my travel agent to replan my trip”) or indirectly via the OS-level agent.

The interaction between these agents can make user experiences more fluid. A user no longer needs to install and launch a dedicated “bird watching” app. Instead, they can simply look at a bird and ask, “What species is this?”. The OS-level agent interprets the intent and context, dispatches a specialised bird-watching agent to recognise the species and retrieve relevant information, and then coordinates spatial overlays and spoken explanations back to the user. In this way, capabilities become modular and composable, and may be packaged and sold as individual agent features (e.g., £5 for the “Bird Watching Pro” feature) rather than as monolithic applications delivered through traditional app stores.

At the same time, autonomous agent systems such as Manus⁸ demonstrate how multistep real-world tasks can be executed without continuous human guidance. For XR glasses prioritising seamless interactions, these agents can minimise the barriers between app boundaries, delivering intuitive experiences to users. However, this agent-centric, distributed architecture also amplifies security, privacy, and governance concerns. Continuous sensing and cross-layer data exchange expand the potential attack surface, and users must trust agents not to misinterpret commands, leak sensitive context, or perform unwanted actions. Designing robust authentication, permissioning, and verification mechanisms for such agent ecosystems, as well as developing testing approaches that can validate these behaviours across architectural layers, becomes a central challenge, which will be revisited in Section 6.3.1.

6.3 Research Problems

This section identifies key research problems that arise specifically from the unique characteristics of XR glasses, viewed through both AI and SE lenses. While challenges such as security, validation, and explainability are well-studied in isolation within AI or XR contexts, their intersection in always-on,

⁷<https://blog.google/products/android/android-xr-gemini-glasses-headsets/>

⁸<https://manus.im/>

embodied, FM-powered nature creates novel complexities that demand tailored solutions. We focus on problems stemming from continuous environmental sensing, tight virtual–physical coupling, and opaque FM decision-making, discussing both the challenges themselves and potential mitigation strategies.

6.3.1 Security and Privacy

Security and privacy present critical concerns for XR glasses, given inherent risks in both AI and XR domains [170, 171] with the unique dangers of always-on, embodied devices that mediate perception and action in the physical world. While hardware vulnerabilities exist, we emphasise software risks.

Security Threats

Security threats can include severe safety risks from external attacks and internal faults. While less immersive than VR, XR glasses may face similar attacks, such as malicious overlays, cybersickness, or manipulating user behaviours [93]. These risks are amplified by XR glasses’ integration with the physical world: a compromised spatial intelligence layer could misdirect user attention toward hazards, while faulty FM processing might generate physically unsafe recommendations (e.g., suggesting to step onto an unstable surface). Embodied AI vulnerabilities, such as adversarial prompting that elicits harmful actions [172, 173], further compound these dangers when FMs control spatial behaviours.

Internal faults pose equally serious threats. Perceptual errors, such as misclassifying a pedestrian as an obstacle, could lead to physical accidents. In safety-critical scenarios (e.g., navigation assistance for the visually impaired), such defects become unacceptable. Moreover, distributed architectures (Section 6.2.3) introduce cross-layer failure modes where cloud-side compromises propagate to on-device rendering or sensor interpretation.

Security Mitigation

Mitigation strategies should involve multiple stakeholders. Effective security demands defence-in-depth across hardware, OS, and application layers. XR operating systems should embed non-bypassable safety mechanisms, such as: (1) Real-time collision and obstacle detection with emergency overrides, (2) user-accessible “safe mode” that disables overlays and reverts to transparent glasses operation, and (3) hardware-enforced isolation between critical perception components and third-party apps.

Other measures for verifying FM-generated spatial content before rendering could be taken to prevent malformed or dangerous outputs from reaching users, such as through sanity checks.

Privacy Threats

Privacy challenges are equally acute, driven by persistent multimodal sensing. Continuous capture of visual, audio, and biometric data creates inherent risks of sensitive information leakage, particularly when raw streams are sent to untrusted edge/cloud services or third-party apps [174–176]. For example, the distributed architecture described in Section 6.2.3 may require sensor data or inferred context to move between on-device, edge, and cloud components, creating more points where sensitive data could be exposed. Traditional permission models, designed for discrete app usage, could be inadequate for XR glasses with always-on nature, where sensing is ambient rather than episodic.

This persistent monitoring could introduce unique *bystander privacy* risks absent in conventional systems. Features like real-time object or people recognition inevitably process data about non-consenting individuals in the user’s field of view [171]. Consider an “identify friend” capability, where the XR glasses must scan faces in a crowd to match against the user’s contacts, collecting biometric signatures without notification or consent. Even anonymised processing (e.g., pseudonymised age and gender identifiers) raises ethical questions about pervasive surveillance.

App store policies (e.g., Google Play’s data safety disclosures) and self-reported data safety disclosures often lack rigorous enforcement, allowing privacy-invasive apps to reach users [177]. While techniques like limiting or blurring device FOV show promise [176], they cannot eliminate risks in unavoidable scenarios like social recognition or environmental scanning.

Privacy Mitigation

Architectural safeguards, such as on-device processing, could be essential to prevent sensitive data transmission. For instance, Apple Vision Pro adopts a private cloud computing design to address this concern [178]. Additional measures could include: 1. Bystander notification mechanisms where feasible (e.g., subtle lights or sounds indicating scanning), or 2. auditable logs of data access and FM inferences for post-hoc accountability. Such strategies must balance functionality, usability, and privacy while remaining enforceable across distributed agent ecosystems.

6.3.2 Validation of Spatial Capabilities

XR glasses introduce profound software engineering challenges in testing and validation, stemming from their tight coupling with unpredictable real-world environments. Unlike traditional software or even VR applications, where testing occurs in relatively controlled synthetic worlds, XR glasses (within AR/MR context) must blend virtual content with live physical contexts in real time. This creates environment-dependent failure modes, meaning failures whose occurrence depends on the surrounding physical context, such as room layout, lighting, moving people, or sensor noise. These failures are difficult to reproduce, debug, and systematically validate. This contrasts fundamentally with traditional context-aware systems like mobile apps, where environmental triggers typically yield discrete and reproducible behavioural changes (e.g., geolocation-based recommendations [179]).

Real-world Testing Challenges

The core difficulty lies in the dynamic interplay between physical reality and generated digital content. Virtual objects must anchor correctly to real surfaces, respect physical laws (e.g., gravity, collisions), occlude properly with real geometry, and respond sensibly to environmental changes (e.g., lighting, motion). For instance, a failure scenario could be a navigation overlay directs the user toward a doorway, but unmodelled real-world clutter (e.g., a moved chair) creates a collision hazard.

Such issues arise from the virtually infinite variation in real-world conditions that cannot be exhaustively enumerated [2]. Effective testing should ensure contextually appropriate responses in dynamic environments. Existing AR apps cope with some of these problems through manual field testing, limited sensor replay, or app-specific oracles. These practices are useful but do not yet provide scalable assurance across the wide range of physical contexts that everyday XR glasses may encounter. The ap-

proaches proposed in previous chapters (XRINTTEST and INTENXION) have demonstrated the ability to generate realistic 3D user interactions for XR testing purposes, but they primarily focused on VR apps. They therefore address one part of the broader XR remit: interaction synthesis and execution in controlled 3D environments. AR and MR extend this remit by adding live physical context, dynamic sensing, and real-world safety constraints.

Recent work has begun addressing specific gaps, such as oracle prediction for virtual object misplacement in AR [180], yet these approaches still rely on manual intervention, limiting scalability.

Path Forward: Specialised Simulation Frameworks

To address these challenges, we suggest developing specialised simulation frameworks that can systematically emulate diverse real-world conditions. Inspired by autonomous driving simulators like CARLA, which support testing of complex systems through controlled scenarios [181], similar infrastructure could be adapted for XR glasses to enable scalable testing of spatial capabilities across various real-world conditions. The work in Chapters 4 and 5 could be adapted to this setting by treating simulated AR/MR environments as richer scenes under test. The interaction taxonomy and IFG could still describe user actions and interaction flows, while the simulator would add controllable real-world variables such as lighting, occlusion, surface geometry, bystander motion, and sensor noise. Such infrastructure should:

- Synthesise realistic physical environments at scale, parametrically varying geometry, materials, lighting, weather, and dynamic elements (people, vehicles).
- Emulate sensor streams (cameras, IMUs, depth sensors) with physically accurate noise models and calibration variations.
- Support scenario-based testing, allowing developers to define critical paths (e.g., “test navigation overlay robustness around moving crowds”) and automatically generate edge-case variants.
- Provide ground-truth oracles for validation, such as perfect 3D reconstructions, known collision states, or human-rated interaction quality.

These capabilities could accelerate development cycles, catch environment-specific defects early, and support continuous integration/testing pipelines—essential for FM-powered systems where behaviours evolve rapidly through retraining and fine-tuning. They could also bridge the gap between today’s VR-centric testing tools and the real-world demands of XR glasses, enabling the rigorous assurance that pervasive spatial computing requires.

6.3.3 Explainability

The spatial intelligence (Section 6.2.1) and multimodal interfaces (Section 6.2.2) that define XR glasses often rely on complex AI models, which can suffer from their inherent “black-box” nature. Users receive guidance, overlays, and recommendations without insight into *why* the system reached those conclusions, creating risks of misplaced trust or undetected errors (Section 6.3.1) [182]. Given their always-on operation and deep integration into daily life, XR glasses require **Explainable AI (XAI)** techniques that clearly reveal decision rationales to users, especially in safety-critical scenarios [183].

Consider a user navigating a busy street: the glasses highlight a “safe area” around detected obstacles. If the system misclassifies a cyclist as a pedestrian, the user has no way to question or override the

recommendation without explicit cues. Without explanations, errors propagate unnoticed, leading to significant risks in safety-critical contexts.

Unique Explainability Challenges

Testing and validating XR glasses pose significant challenges due to the dynamic and unpredictable environments in which these devices operate (Section 6.3.2). XR glasses amplify traditional XAI challenges in several ways:

- **Non-determinism and variability:** FMs exhibit stochastic behaviour [184], producing different explanations or even decisions for identical inputs. Spatial contexts also vary continuously, requiring explanations that remain consistent yet contextually relevant.
- **Personalisation tensions:** Users may differ in explanation preferences. For instance, some may prefer concise summaries (e.g., *“table is stable”*), while others may prefer detailed traces (e.g., *“80% confidence based on texture analysis and edge detection”*). Adaptive XAI must support these preferences without introducing further variability [183].
- **Scalability across open worlds:** Current XAI evaluation methods are often model-specific [185, 186] and may assume bounded inputs. XR glasses face unbounded real-world variation, making comprehensive explanation validation infeasible without specialised infrastructure.

We identify these gaps as core challenges that require joint effort from both AI and SE perspectives. We propose that future research directions could include extending the simulation environment described in Section 6.3.2.

Path Forward: Simulation-driven XAI Pipelines

A simulation environment that captures end-to-end pipelines, from raw sensor streams through FM inference to rendered outputs, would enable systematic explanation testing.

For example, the framework can support stress-testing with scenario-based explanations. By generating thousands of environmental variants (e.g., varying lighting and motion), it can verify whether explanations remain accurate, consistent, and actionable. In addition, causal analysis could be utilised to identify misbehaviours [187]. Simulation ground truth could be used to trace erroneous decisions back through the pipeline. For instance, a tracing message could be *“misplacement due to 15° depth estimation error”*.

Integrating such techniques into **Machine Learning Operations (MLOps)** pipelines could position XAI validation as a fundamental component of trustworthy XR development, akin to functional correctness. Developers could iterate on models with automated explanation quality metrics, identifying issues before deployment. For deployed systems, runtime monitoring could also flag explanation misbehaviours (e.g., declining confidence scores) and trigger fallbacks or human review.

6.4 Broader Challenges

Beyond the technical research problems discussed above, the responsible development and deployment of XR glasses must also address profound societal implications. This section examines three critical non-technical domains: ethical governance, accessibility and inclusivity, and open development ecosystems.

Progress in these areas will be essential if XR glasses are to serve diverse users equitably, foster healthy developer communities, and align with broader societal values rather than exacerbate existing divides.

6.4.1 Ethics Governance

The always-on, perception-altering nature of XR glasses raises serious ethical concerns about their impact on human cognition, attention, and autonomy.

Cognitive Dissociation Risks Avatar-centric social features (e.g., real-time communication with other users' avatars) could contribute to dissociative identity disorders, where prolonged immersion blurs the boundary between physical and virtual selves [188]. Users might struggle to maintain a coherent sense of identity or reality, particularly vulnerable populations such as children or those with pre-existing mental health conditions. Unlike discrete VR sessions, XR glasses' continuous operation amplifies this risk, as virtual overlays become a persistent layer of everyday perception.

Attention Exploitation XR glasses could enable new forms of human attention exploitation. Unlike current addictive interfaces (e.g., personalised social media feeds), their always-on displays can inject unskippable advertisements, notifications, or behaviourally manipulative content directly into users' **Field of View (FOV)**. This content could bypass traditional consent mechanisms such as closing an app or scrolling past a feed. This latent "cognitive hijacking" could lead to information overload and decision fatigue, potentially degrading human mental autonomy at psychological or neurological levels [189].

To mitigate these harms, we suggest collaboration between AI and SE communities to work with policymakers on regulatory frameworks, like the **General Data Protection Regulation (GDPR)**⁹, to establish attention sovereignty principles. These could restrict manipulative patterns and support user cognitive control. In addition, technical measures could include manipulative pattern detection via software testing approaches and enforcement mechanisms embedded in XR OSs, such as hard limits on overlay density, and runtime cognitive load monitoring.

By treating human attention as a finite, protectable resource, these measures could prevent XR glasses from becoming tools of exploitation and instead position them as enhancers of cognitive agency.

6.4.2 Accessibility and Inclusivity

XR glasses could adapt to diverse user capabilities through multimodal interfaces. For example, gaze dwell-time, head tracking, or subtle **Electromyography (EMG)** gestures (Section 6.2.2) might benefit users with motor impairments, replacing physically demanding controller or touch interactions. Furthermore, for visually impaired users, spatial intelligence can translate visual scenes into audio descriptions, haptic patterns, or spatialised soundscapes [156]. Systems like Apple's SceneScout demonstrate how multimodal agents can provide accessible interactions with spatial content for blind users [190]. These capabilities could make digital assistance more equitable, turning XR glasses into powerful augmentations for users who currently face barriers with conventional devices.

While XR glasses show promise for supporting individuals with disabilities, addressing these accessibility issues may require coordinated efforts from various stakeholders. Key challenges in the emerging XR market may include uncertain adoption rates among users with disabilities. Early XR glasses may

⁹<https://gdpr-info.eu/>

target mainstream consumers, sidelining niche needs. In addition, unlike mature platforms with established accessibility guidelines (e.g., [Web Content Accessibility Guidelines \(WCAG\)](#) for web), XR lacks standardised inclusive design patterns. Developers may overlook spatial-specific needs, such as ensuring audio descriptions work with dynamic overlays or gaze controls that accommodate varying eye conditions.

Software engineering could play a crucial role in addressing these challenges, where established accessibility testing techniques [191] could be adapted for XR platforms. By embedding accessibility as a concern in XR platforms, from OS APIs to developer tooling, software engineering measures could help ensure XR glasses expand human capability rather than create new digital divides.

6.4.3 Open Development Ecosystems

The nascent XR glasses ecosystems face challenges with limited open resources and a scarcity of available apps. Many existing apps are closed-source or commercial, with limited documentation of best practices. Unlike mature platforms like Android, which benefit from vibrant open repositories (e.g., F-Droid¹⁰), XR currently lacks strong community-driven infrastructures for knowledge sharing. Such ecosystems could be beneficial for practitioner learning and SE research communities to develop and evaluate new techniques. This scarcity of open-source XR resources may partly result from competitive market dynamics, where companies tend to protect proprietary advancements.

Research communities can benefit from actively monitoring industrial development and utilising curated knowledge platforms (e.g., research newsletters that analyse emerging tools and technology stacks) to foster timely innovation. Openness also supports the security and privacy concerns associated with XR. Transparency, verifiable open-source code, likely monitored by independent third parties, could help build trust in XR systems. Enhanced academia-industry collaboration could be valuable for developing safer, more user-centred XR software.

Research communities can drive change through developing curated knowledge platforms (e.g., newsletters, wikis), tracking industrial toolchains and distilling actionable insights. Efforts should be taken to bridge academia-industry gaps by creating joint challenges, shared datasets, and funding for open toolchains that solve mutual pain points.

By prioritising openness, the XR community can transform XR glasses from siloed corporate competitions into a vibrant, trustworthy platform that benefits developers, researchers, and end users alike.

6.5 Answer to ORQ7

This chapter provides an answer to ORQ7 from Section 1.1.

ORQ7: *How might XR software evolve in the future, what new challenges are likely to arise, and in what ways can software testing help to address these challenges?*

6.5.1 XR Software Evolution

XR software will shift from today's discrete VR/AR apps toward always-on AI-powered XR glasses that continuously overlay intelligent digital content onto the physical world. Foundation models will

¹⁰<https://f-droid.org/>

drive spatial understanding, physics-aware content generation, and agent-mediated interactions, where OS-level and domain-specific agents collaborate to handle multimodal inputs (e.g., gaze, gesture, voice, EMG) and orchestrate seamless workflows. Distributed architectures will span on-device sensing/rendering, edge inference, and cloud services, positioning XR glasses as potential smartphone successors through embodied human-AI partnerships.

6.5.2 New Challenges

This evolution introduces profound challenges. Always-on sensing creates privacy risks for bystanders and attack surfaces for malicious overlays that could cause physical harm. Environment-dependent behaviours defy traditional testing, as virtual content must integrate flawlessly with infinite real-world variations. Black-box FMs demand real-time explanations to build trust in safety-critical guidance. Societally, persistent overlays risk cognitive dissociation and attention exploitation, while accessibility gaps and proprietary ecosystems threaten equitable adoption.

6.5.3 Role of Software Testing

Software testing becomes central to trustworthy XR glasses, evolving from bug-finding to proactive assurance. Specialised simulators, inspired by autonomous driving platforms like CARLA, enable scalable validation by parametrically generating real-world scenarios and providing ground-truth oracles for spatial accuracy, physics compliance, and interaction quality. These environments should also support XAI testing, verifying explanation fidelity across diverse contexts and user preferences.

Security testing could fuzz multimodal inputs and FM prompts to expose perceptual vulnerabilities, while privacy assurance could validate data flows and bystander protections. Accessibility toolchains could simulate impairments to enforce inclusive design from the start. Cross-layer fault injection measures could help ensure graceful degradation in distributed systems. Open testing frameworks, such as benchmarks, agent harnesses, [Continuous Integration/Continuous Delivery or Deployment \(CI/CD\)](#) pipelines, can accelerate ecosystem growth and independent verification.

By embedding simulation-driven, XAI-aware, and ethically-focused testing into MLOps pipelines, software engineering transforms XR glasses from risky experiments into safe, inclusive platforms that responsibly realise their transformative potential.

6.6 Conclusion

This chapter has projected XR software beyond the applications developed with 3D engines and the testing approaches developed in Chapters 4 and 5, envisioning always-on AI-powered XR glasses as the next paradigm in spatial computing [3]. By establishing a human-in-the-loop conceptual framework (Section 6.1), mapping their distinctive capabilities (Section 6.2), and analysing the resulting AI-SE research problems (Section 6.3) alongside societal challenges (Section 6.4), it charts a roadmap for this emerging domain.

Though consumer-grade XR glasses are still emergent, precursor technologies (e.g., smart glasses, XR HMDs) demonstrate the field's research potential. Given the XR SE research community is gradually gaining momentum (e.g., a recent special issue on *Virtual and Augmented Reality Software Engineering*

in *Automated Software Engineering*), we expect a rapid growth of this domain in the FM era. Through early interdisciplinary collaboration across AI, software engineering, and HCI, researchers can steer XR glasses toward safe, inclusive, and human-centred applications that responsibly augment rather than supplant human capabilities.

Chapter 7

Conclusions and Future Work

This chapter concludes the thesis and looks ahead to what comes next. The work began by asking a simple but underexplored question: what does it really mean to test software that lives not on a flat screen, but in the space around us? From this starting point, the central research objective was defined as follows: *How can XR applications be systematically and automatically tested to improve test performance and more reliably uncover bugs?*

Chapter 2 established the technical foundation by introducing the core principles of XR technology. It described how Unity-based XR applications are built from code and scenes, and how the XR Interaction Toolkit, a high-level XR user interaction framework, structures user interactions. It also explained why spatial, physics-driven behaviours fundamentally differ from traditional GUI logic. This chapter framed XR apps as hybrids of program code and spatial configuration, foreshadowing the later focus on testing approaches that are aware of XR scene configurations.

Chapter 3 broadened the perspective through a systematic mapping study of existing research. The study revealed that most prior work concentrates on functional correctness and basic interaction verification, yet typically employs simplified notions of input (e.g., point-and-click interfaces or abstracted events). It revealed a clear gap in addressing realistic [Six Degrees of Freedom \(6DoF\)](#) controller inputs and rich spatial interactions, which are central to modern XR experiences. This gap provided the motivation and direction for the rest of the thesis.

Chapters 4 and 5 transitioned from analysis to implementation and empirical evaluation. Chapter 4 introduced INTENXION as a first step toward bringing structure and automation to XR interaction testing. It proposed a three-layer taxonomy of XR interactions, including core tasks, atomic actions, and compositional patterns. Subsequently, we showed how this taxonomy could be turned into a set of test automation APIs for Unity-based XR applications. Through eight representative interactable object types, the chapter demonstrated that XR interaction designs can be decomposed into testable building blocks and validated via hardware-free tests.

Chapter 5 extended these foundations by developing and evaluating XRINTTEST, an automated XR testing tool grounded in the [Interaction Flow Graph \(IFG\)](#) model. Drawing on an empirical analysis of ten open-source XR scenes, the chapter identified 456 interactions and showed that the vast majority are built from grab and trigger primitives. XRINTTEST utilises the IFG to explore scenes, exercise interactions, and check outcomes. The evaluation showed XRINTTEST achieving 93% interaction coverage, substantially outperforming a random baseline. The remaining uncovered interactions

mainly reflect runtime accessibility and modelling constraints, such as dynamic scene changes and object configurations, rather than a failure of the underlying interaction model. XRINTTEST also discovered subtle configuration bugs that human testers had missed. In doing so, the chapter illustrated how abstract models and automated testing approaches can turn complex 3D interaction spaces into tractable, testable artefacts.

Finally, Chapter 6 extended the scope of discussion toward emerging AI-powered XR software. Rather than proposing a new testing tool, the chapter used the thesis's earlier findings on XR interaction modelling, automation, and oracle challenges as a starting point for identifying future software engineering problems for AI-powered XR glasses. The always-on, multimodal nature of these devices may allow them to blend spatial intelligence with foundation models, creating new testing challenges. The chapter proposed a human-in-the-loop conceptual framework and examined advanced capabilities such as spatial intelligence, multimodal interfaces, and distributed software architectures. It identified new research problems in validation, explainability, and security, as well as broader societal challenges like ethics, accessibility, and openness. In doing so, it connected the thesis's concrete contributions to a broader vision of AI-centric software engineering for XR.

The remainder of this chapter reflects on the limitations of the studies presented and discusses how they shape the agenda for future work. Section 7.1 summarises key threats to validity and scope constraints across the empirical studies and tools. Section 7.2 then outlines promising research directions that could guide the next generation of XR software testing research, including both extensions of the current frameworks and prospective avenues inspired by the vision for XR glasses.

7.1 Limitations

This section outlines the main limitations of the thesis, particularly the approaches proposed in Chapters 4 and 5. While our study demonstrates the feasibility and potential of testing strategies for XR environments, several factors constrain the generalisability and completeness of our evaluation. Specifically, we discuss the generalisability of our proposed techniques and the availability of the experimental subjects. Together, these discussions clarify the boundaries of our current contributions and highlight potential mitigation directions.

7.1.1 Generalisability

A potential threat to external validity lies in the selection of **Virtual Reality (VR)** applications used in our evaluation. While our approach focuses primarily on Unity-based VR implementations, we argue that the underlying principles and findings are broadly generalisable across **Extended Reality (XR)** ecosystems. Unity currently dominates VR development with over 60% market share [55] and offers integrations with virtually all major XR platforms [21]. Furthermore, Unity remains the most widely adopted platform for evaluating XR software testing research [13]. These factors together make Unity-based projects a representative and practical foundation for developing and validating our approaches.

Our approach and evaluation were designed from the perspective of modern XR interaction practices, particularly those mediated by **Head-mounted Displays (HMDs)** equipped with **Six Degrees of Freedom (6DoF)** hand-held controllers. This focus was motivated by two key considerations. First, Unity's strong

ecosystem and native support from leading hardware manufacturers (e.g., Meta and Pico¹) ensure that a vast number of real-world projects can directly benefit from our methods. Second, the controller-based interaction paradigm currently represents the most prevalent form of rich, fine-grained interaction in XR environments, making it an ideal basis for devising and testing automated exploration techniques.

Importantly, the conceptual design of our approach transcends both specific XR technologies (e.g., VR) and particular development platforms (e.g., Unity). The generalisation of our method can be discussed from two complementary perspectives: *conceptual* and *technical*.

At the *conceptual level*, our approach is founded on the “interactor-interactable” user interaction pattern, which serves as the prevailing model for managing user input across modern XR apps. Within this paradigm, users employ input devices to perform actions on objects within a spatial environment, which is foundational not only to VR but also to [Augmented Reality \(AR\)](#) and [Mixed Reality \(MR\)](#) systems. Consequently, our approach is expected to remain applicable to XR frameworks that follow this interaction model, although this claim still requires further empirical validation on AR and MR projects.

At the *technical level*, the most direct parallel platform beyond Unity is Unreal Engine, another leading engine for XR development that employs a conceptually similar interaction paradigm. Unreal Engine’s [Motion Controller](#) component² manages input from XR input devices much like Unity’s “Interactor”. Furthermore, objects are made interactive by adding components like the [Grab Component](#)³, which are analogous to Unity’s [XR Grab Interactable](#) provided by Unity’s XR Interaction Toolkit (mentioned in Section 2.2). Therefore, porting our approach to Unreal Engine would primarily involve a systematic mapping between engine-specific APIs and constructs, rather than a redesign of the underlying logic or abstractions. The foundational principles behind our test automation library (Chapter 4) and automated exploration strategy (Chapter 5) would remain unchanged under such a translation.

Additionally, the same principles could be directly extended beyond VR to other XR modalities, including AR and MR. In fact, Unity’s XR Interaction Toolkit, which is a primary subject of our work, is directly applicable for building AR/MR applications⁴. Although AR/MR technologies introduce additional real-world context, many of their interaction patterns, such as grabbing, triggering, and controller movement, remain analogous. Our decision to evaluate only VR applications was therefore motivated by pragmatic considerations, particularly the availability of publicly accessible open-source projects suitable for automated testing. At the time of evaluation, our search did not identify AR/MR projects with sufficiently rich interaction structures to support meaningful experimentation, and even Unity’s official MR template contained only a single interactable object. This rationale is also reflected in Chapters 4 and 5, where the thesis scope is defined around HMD-based, controller-driven XR interactions.

Looking forward, we plan to extend our approach to accommodate custom interaction patterns and broader scopes of spatial interaction. These interactions could be generated dynamically at runtime through learning-based methodologies (e.g., [Reinforcement Learning \(RL\)](#) or [Imitation Learning \(IL\)](#) [59], which will be discussed further in Section 7.2). The current work represents an initial step in this direction. Future studies will aim to rigorously evaluate our solutions’ generalisation capabilities across broader XR modalities and adapt them to evolving patterns of spatial interaction.

¹<https://www.picoxr.com/>

²<https://dev.epicgames.com/documentation/unreal-engine/motion-controller-component-setup-in-unreal-engine>

³<https://dev.epicgames.com/documentation/unreal-engine/vr-template-in-unreal-engine#grab>

⁴<https://docs.unity3d.com/Packages/com.unity.template.mixed-reality@2.1/manual/index.html>

7.1.2 Limited Subjects

The benchmark dataset, XRBENCH3D, primarily consists of tutorial and sample VR applications, which may raise questions regarding the representativeness of our evaluation subjects. We acknowledge this as a potential limitation but argue that these applications nonetheless offer sufficient diversity and realism to effectively evaluate our proposed approaches.

To mitigate representativeness concerns, we carefully curated and manually verified each subject to ensure that the selected applications collectively reflect a diverse spectrum of modern VR interaction patterns. The dataset includes projects that span a variety of interaction types, such as object manipulation, multi-step puzzle, or task completion. It represents the core mechanisms that underpin a broad range of modern VR capabilities.

In particular, Scenes 3, 4, 7, and 10 of the dataset each feature over 50 distinct interactions, demonstrating a wide variety of event types and object behaviours. These include both predefined interaction templates and custom interaction scripts that emulate real-world VR design practices. Notably, Scenes 7 and 9 represent complete VR puzzle-solving games that require players to explore the environments, manipulate objects, and follow specific interaction sequences to complete game levels. Therefore, these scenes provide complex and realistic subjects for evaluating the robustness and scalability of our automated exploration technique.

It is important to note that VR remains an emerging field, and the number of high-quality, open-source VR applications is currently limited [13]. As such, constructing a representative and reusable benchmark dataset remains a broader challenge for the XR research community. Our dataset reflects the first set of publicly available Unity-based VR projects that offer functionality and interaction variety. We view this as a significant starting point for systematic evaluations.

To promote transparency and reproducibility, we have made all our code and datasets publicly available online. Specifically, the implementations of XRINTTEST and the XRBENCH3D benchmark can be accessed at <https://github.com/ruizhengu/XRintTest> and <https://github.com/ruizhengu/XRBench3D>, respectively. This open access is intended to facilitate replication and extension of our work, enabling other researchers to validate results, integrate new interaction models, and contribute to the creation of standardised benchmarks for XR software testing as the number of open-source XR applications continues to grow.

7.2 Future Work

While our study establishes a foundation for automated testing in XR environments, several promising directions remain open for further exploration. Future work can build upon our framework to enhance adaptability, scalability, and realism in XR software testing. We envisage three primary avenues for advancement: agent-based testing, testing XR-specific features, and specialised XR simulation platforms. The following subsections outline these future directions in detail.

7.2.1 Agent-based Testing

Although our approach comprehensively addresses fundamental XR interaction types, many custom and dynamic interaction patterns remain beyond the current system's scope. We propose a two-phase research direction to address this limitation. The first phase involves conducting a systematic empirical

study to characterise common custom interaction patterns, examining how developers design object relationships, physics behaviours, and interaction constraints in real-world XR projects. This study would also identify key challenges these patterns pose for automated testing, particularly in defining robust and generalisable test oracles capable of verifying correctness across varied scenarios.

Building on these insights, the second phase would explore the integration of deep [Reinforcement Learning \(RL\)](#) and [Imitation Learning \(IL\)](#) techniques to automatically adapt testing strategies to previously unseen or evolving interaction patterns. RL and IL have demonstrated remarkable success in domains that demand spatial reasoning and dynamic control, such as bimanual robotic manipulation [192], Android GUI testing [193], and video game exploration [59]. These methods could be adapted to learn how to perform and verify complex 3D interactions in VR environments, such as object assembly, tool use, or sequential puzzle-solving, without relying solely on predefined interaction graphs (e.g., the [Interaction Flow Graph \(IFG\)](#)).

Leveraging the underlying architecture of XRINTTEST, we envision XR test agents capable of autonomously exploring scenes, identifying interactable objects, and executing actions strategically to maximise coverage and uncover unexpected behaviours. Moreover, the [Interaction Flow Graph \(IFG\)](#) introduced in our current framework could serve as a foundation for interaction fuzzing [194], where test agents intentionally execute interactions that deviate from observed patterns to stress-test boundary conditions and error-handling logic. Such an approach could significantly enhance both the breadth and depth of automated XR testing. A preliminary version of this idea could be explored manually by executing invalid or unexpected interaction sequences, such as triggering before grabbing, releasing an object midway through a required sequence, or attempting socket placement with an incompatible object. These manually seeded behaviours would provide an initial fault model for later automated interaction fuzzing.

LLM-based XR testing agents. For XR testing, LLM agents could be useful in interpreting testing objectives, planning interaction sequences, and explaining detected failures. However, their practical adoption may introduce three challenges. First, LLM outputs are non-deterministic and may contain hallucinations [184]. The LLM should therefore perform high-level reasoning, while a deterministic runtime validates and executes generated test actions. Second, repeated reasoning over large XR projects creates latency and context-window pressure. Structured representations, such as [Abstract Syntax Tree \(AST\)](#) or IFGs, could provide compact project context and reduce unnecessary model input [195, 196]. Third, useful outputs require grounding in relevant code, scene definitions, interaction models, and execution evidence. Retrieval-augmented and ReAct (reasoning and acting)-style workflows could gather this context before acting [197, 198]. When evidence remains insufficient, the agent should request user clarification through a human-in-the-loop process rather than guess.

DSL-guided XR test generation and maintenance. Future work could extend INTENXION with an XR interaction [Domain-Specific Language \(DSL\)](#) that expresses testing intent instead of test scripts. An LLM-based agent could translate a user’s natural-language query into a DSL specification, which a deterministic generator would then convert into executable INTENXION test cases [41, 199]. This intermediate representation could also improve regression-test maintenance. When an XR app evolves, implementation-level test cases may fail even though their intended interaction behaviours remain unchanged. The stable DSL specification could guide the repair or regeneration of these tests. Compared

with the current library interface, a DSL would improve readability and maintainability [200], but would require additional effort to design and validate its syntax and mappings to executable actions [201].

In summary, advancing toward agent-based testing offers a pathway to more generalisable, intelligent, and adaptive exploration of interactive 3D environments. By combining data-driven learning with structural interaction modelling, future XR testing frameworks could autonomously evolve with the applications they assess, keeping pace with the growing complexity of immersive environments.

7.2.2 Testing XR-specific Features

This thesis primarily focuses on the testing of XR user interactions, with particular attention to *3D Interaction* mechanisms as defined in Unity’s *XR Interaction Toolkit (XRI)* (<https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.1/manual/object-interaction.html>). These 3D interactions correspond to the “Interactor–Interactable” paradigm (see Section 7.1.1), in which users employ gestures or controllers to manipulate objects within a spatial environment. This paradigm represents the most prevalent and characteristic interaction model in contemporary XR applications, forming the foundation of immersive user experiences.

It is important to distinguish these 3D interactions from the other two major categories of XRI: *Locomotion* and *UI Interaction*. Although all three are integral to XR experiences, our work explicitly prioritises 3D interactions for one key reason. They involve the richest and most complex spatial relationships, encompassing manipulation, grabbing, triggering, and object constraint behaviours that reflect core challenges in testing XR systems. These interactions require analysis of their inherent actions and conditions, which makes automated testing particularly difficult.

While locomotion functionality (e.g., teleportation, continuous movement, navigation) is not directly modelled in our current approach, our testing framework inherently accommodates certain aspects of controller-based movement, as these motions are tightly coupled with interaction events. Furthermore, 2D UI interactions in XR applications are excluded from the current study. Unity’s implementation of UI interactions largely builds upon the conventional 2D UI framework (such as **Event Trigger** discussed in Section 4.1), an area that has been extensively studied in prior software testing research [54, 55]. Given this existing body of work, our focus is deliberately placed on the 3D interaction space, which remains a comparatively underexplored but critically important area for ensuring the functional reliability of immersive XR experiences.

Future extensions of our framework could naturally expand support to these additional categories of XR-specific features. For example, modelling locomotion systems could enable automated assessment of navigation and teleportation functionalities, while integrating UI interaction testing could facilitate unified validation across mixed 2D–3D interfaces.

7.2.3 Specialised XR Simulation Platforms

Beyond interaction modelling at the engine level, this thesis also highlights the importance of specialised XR simulation platforms for testing applications on emerging XR glasses (see Chapter 6) or AR/MR devices in general. Such platforms can provide controlled, repeatable, and scalable environments for evaluating XR apps under realistic device constraints, without requiring continuous access to physical hardware and real-world surroundings. They could be particularly valuable for XR glasses, which often

impose strict limits on compute, energy, network bandwidth, and sensing capabilities compared to traditional VR headsets.

Such specialised platforms would complement the engine-level techniques proposed in this thesis by enabling end-to-end testing that spans from interaction logic to deployment contexts on XR glasses. In particular, integrating our testing approaches (i.e., XRINTTEST and INTENXION) with XR-glasses simulators would allow evaluation of how interaction patterns behave under real-world settings. As XR glasses move toward everyday, mobile usage scenarios, this line of work becomes increasingly important for ensuring reliability, safety, and user comfort in next-generation immersive systems.

7.2.4 Investigating Testing Practices Among XR Software Developers

Given the limited publicly accessible knowledge about XR software engineering practices in industrial contexts (as discussed in Section 6.4), gaining insight into how XR software is developed and tested remains an important open challenge. In particular, understanding industry perspectives would provide valuable resources for advancing automated XR testing research. For instance, these perspectives can cover prevailing best practices, distinctions between XR and conventional software in terms of the *Software Development Life Cycle (SDLC)*, and attitudes toward the adoption of automated testing.

To collect such information, a range of systematic empirical research approaches could be employed. One promising direction involves conducting developer surveys with professionals engaged in real-world XR software development. Surveys can yield first-hand insights into practitioners' perspectives, experiences, and concerns regarding the adoption of automated testing techniques in XR contexts [58, 202].

Complementary to surveys, semi-structured interviews with industry practitioners can provide a more detailed and nuanced understanding through open-ended yet guided discussions [203]. For example, an interview question such as *“What is your perspective on how and when automated XR testing could be integrated into the development process of XR software?”* might elicit responses like: *“Automated testing could be used after a developer completes a *Pull Request (PR)*, in parallel with unit testing. Once automated XR interaction tests identify defects and developers address them, the updated version could be handed over to manual testers for further validation. We also see potential for integrating automated XR interaction testing within the *Continuous Integration/Continuous Delivery or Deployment (CI/CD)* pipeline.”*⁵.

Beyond qualitative inquiries, controlled studies could be designed to quantitatively assess the impact of automated testing techniques on software testing performance, such as improvements in defect detection, testing efficiency, and reliability, compared to traditional testing practices [40].

Collectively, such empirical investigations can provide a comprehensive understanding of current industry practices, reveal the practical challenges that XR developers face, and inform the design of future tools and methodologies that align more closely with real-world development needs.

7.3 Final Remarks

This thesis has advanced the understanding and practice of automated software testing within the emerging domain of *Extended Reality (XR)*. By systematically mapping existing research, modelling

⁵This illustrative question and response are derived from an initial conversation with a quality assurance manager at FitXR (<https://fitxr.com/>)

XR user interactions, and developing practical automation strategies and benchmarks, it provides not only methodological foundations but also practical prototypes for testing immersive, interactive systems. The proposed methodologies and open-source tools demonstrate how test automation and automated testing can effectively evaluate XR user interactions, offering empirical evidence of their capabilities. Furthermore, by publicly releasing reusable datasets and tools, this research has laid a foundation for reproducibility and future collaboration within the XR testing community. Collectively, these contributions extend the state of the art in XR software testing by introducing empirically grounded approaches for assessing complex human–computer interactions in XR environments, laying a solid foundation for more practical, robust and scalable testing solutions for XR software.

Appendix A

Primary Studies of the Systematic Mapping Study

This appdenix provides the information regarding the primary studies used for the systematic mapping study for XR software testing in Chapter 3. It includes the list of the primary studies ans their publication venues.

A.1 List of Primary Studies

The list corresponds to the studies prefaced with “PS” throughout the paper.

ID	Title
PS1 [89]	360° Stereo image based VR motion sickness testing system
PS2 [204]	A Study of User Privacy in Android Mobile AR Apps
PS3 [97]	An Agent-based Architecture for AI-Enhanced Automated Testing for XR Systems, a Short Paper
PS4 [205]	An Approach for Model Based Testing of Augmented Reality Applications
PS5 [36]	An automated functional testing approach for virtual reality applications
PS6 [84]	An Exploratory Study of Bugs in Extended Reality Applications on the Web
PS7 [99]	ARCHIE++ : A Cloud-Enabled Framework for Conducting AR System Testing in the Wild
PS8 [206]	Automated Testing of Functional Requirements for Virtual Reality Applications
PS9 [83]	Automated testing of virtual reality application interfaces
PS10 [207]	Automated Usability Evaluation of Virtual Reality Applications
PS11 [208]	Automatic determination of text readability over textured backgrounds for augmented reality systems

Continued on the next page

ID	Title
PS12 [96]	Blackbox Testing on Virtual Reality Gamelan Saron Using Equivalence Partition Method
PS13 [98]	Detecting and Preventing Faked Mixed Reality
PS14 [209]	Detecting framerate-oriented cyber attacks on user experience in virtual reality
PS15 [92]	Detection of Security and Privacy Attacks Disrupting User Immersive Experience in Virtual Reality Learning Environments
PS16 [210]	DyTRec: A Dynamic Testing Recommendation tool for Unity-based Virtual Reality Software
PS17 [18]	Exploiting deep reinforcement learning and metamorphic testing to automatically test virtual reality applications
PS18 [91]	Hidden in Plain Sight: Exploring Privacy Risks of Mobile Augmented Reality Applications
PS19 [12]	Less Cybersickness, Please: Demystifying and Detecting Stereoscopic Visual Inconsistencies in Virtual Reality Apps
PS20 [211]	LUTE: A Locomotion Usability Test Environment for Virtual Reality
PS21 [90]	Measurement of exceptional motion in VR video contents for VR sickness assessment using deep convolutional autoencoder
PS22 [212]	Performance Analysis of Augmented Reality Based on Vuforia Using 3D Marker Detection
PS23 [82]	PredART: Towards Automatic Oracle Prediction of Object Placements in Augmented Reality Testing
PS24 [105]	Scriptless Testing for Extended Reality Systems
PS25 [213]	Software Testing Automation of VR-Based Systems With Haptic Interfaces
PS26 [214]	Test automation for augmented reality applications: a development process model and case study
PS27 [11]	Towards Automatic Oracle Prediction for AR Testing: Assessing Virtual Object Placement Quality under Real-World Scenes
PS28 [215]	Towards the Systematic Testing of Virtual Reality Programs
PS29 [10]	Understanding VR Software Testing Needs from Stakeholders' Points of View
PS30 [216]	Unravelling the Black Box: Enhancing Virtual Reality Network Security with Interpretable Deep Learning-Based Intrusion Detection System
PS31 [81]	Virtual Reality (VR) Automated Testing in the Wild: A Case Study on Unity-Based VR Applications
PS32 [26]	VRGuide: Efficient Testing of Virtual Reality Scenes via Dynamic Cut Coverage

Continued on the next page

ID	Title
PS33 [80]	VRTest: An Extensible Framework for Automatic Testing of Virtual Reality Scenes
PS34 [94]	Youkai: A Cross-Platform Framework for Testing VR/AR Apps

A.2 Publication Venues of Primary Studies

This section we provide the publication venues of the primary studies, as long as their

#	Venue	Type	Domain	No.
1	International Symposium on Software Testing and Analysis (ISSTA)	Symposium	SE	2
2	Symposium on Virtual Reality (SVR)	Symposium	XR	2
3	Software Testing, Verification and Reliability (STVR)	Journal	SE	2
4	International Workshop on Virtual and Augmented Reality Software Engineering (VARSE)	Workshop	XR + SE	2
5	International Conference on Automated Software Engineering (ASE)	Conference	SE	2
6	ACM Symposium on Virtual Reality Software and Technology	Symposium	XR	1
7	International Conference on Vocational Education and Training (ICOVET)	Conference	General	1
8	International Conference on the Foundations of Software Engineering (FSE)	Conference	SE	1
9	International Journal of Frontiers in Engineering Technology	Journal	General	1
10	International Symposium on Mixed and Augmented Reality	Conference	XR	1
11	International Symposium on Software Reliability Engineering (ISSRE)	Symposium	SE	1
12	International Workshop on Quality and Measurement of Software Model-Driven Development	Workshop	SE	1
13	International Workshop on Artificial Intelligence in Software Testing	Workshop	SE	1
14	International Conference on Software Engineering (ICSE)	Tool	SE	1
15	International Workshop on Security for XR and XR for Security	Workshop	XR + SP	1
16	Journal of Interactive Media (i-com)	Journal	HCI	1
17	Jurnal Buana Informatika (JBI)	Journal	General	1

Continued on the next page

#	Venue	Type	Domain	No.
18	Pacific Rim International Symposium on Dependable Computing (PRDC)	Conference	General	1
19	Research Challenges in Information Science (RCIS)	Conference	General	1
20	International Conference on Virtual Worlds and Games for Serious Applications (VS-Games)	Conference	XR	1
21	International Conference on Reliability, Maintainability and Safety (ICRMS)	Conference	General	1
22	ACM Transactions on Computer-Human Interaction	Journal	HCI	1
23	IEEE Annual International Symposium Virtual Reality	Conference	XR	1
24	ACM Transactions on Privacy and Security	Journal	SP	1
25	Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)	Conference	General	1
26	Applied Sciences	Journal	General	1
27	Assembly Automation	Journal	General	1
28	Conference on Graphics, Patterns and Images	Conference	XR	1
29	HCI International (HCII)	Conference	HCI	1
30	IEEE Transactions on Services Computing	Journal	General	1
31	International Conference on Multimedia Information Processing and Retrieval (MIPR)	Conference	General	1
32	IEEE Transactions on Visualization and Computer Graphics	Journal	CG	1
33	Immersive Projection Technology Workshop	Workshop	XR	1
34	International Conference on Actual Problems of Electronic Instrument Engineering (APEIE)	Conference	General	1
35	International Conference on Computer Science and Electronics Engineering	Conference	General	1
36	International Conference on Emerging Trends & Innovation in ICT (ICEI)	Conference	General	1
37	International Conference on Information and Communication Technology Convergence (ICTC)	Conference	General	1
38	International Conference on Metaverse Computing, Networking and Applications (MetaCom)	Conference	XR	1
39	The Computer Journal	Journal	General	1

Appendix B

Applying XRINTTEST for Test Automation

This appendix provides practical guidance on how to install, configure, and run XRINTTEST on Unity-based XR projects. The steps below assume a working Unity project and describe how to (1) check and prepare the environment, (2) integrate the required packages and scripts, (3) configure scenes and construct the [Interaction Flow Graph \(IFG\)](#) used for automated testing, and (4) execute XRINTTEST for dynamic exploration and generate test results.

B.1 Requirement Checking and Installation

B.1.1 Verify Unity Installation

1. In a terminal, navigate to the root directory of XRINTTEST.
2. Run:

```
sh check_unity.sh
```

3. If Unity is not detected, download and install a supported (e.g., [Long-Term Support \(LTS\)](#)) Unity version from: <https://unity.com/>.

B.1.2 Add Required Packages and Scripts

1. From the root directory of XRINTTEST, run:

```
sh add_requirements.sh 'unity_project_path'
```

Replace ‘unity_project_path’ with the path to your Unity project.

2. This script adds the necessary Unity packages and XRINTTEST scripts to the target project.

B.2 Scene Configuration

B.2.1 Open the Scene Under Test

1. Start the Unity Editor.
2. Open the Unity project into which XRINTTEST has been integrated.
3. In the project directory, open the scene you wish to test (i.e., **Scene Under Test (ScUT)**).

B.2.2 Configure the Scene

In the Unity Editor menu bar, as shown in Figure B.1, open:

Tools → Configure Scene

This step prepares the scene for automated XR interaction testing (e.g., ensuring required components and settings are applied).

B.3 Tool Execution

B.3.1 Interaction Flow Graph Construction

With the ScUT open in the Unity Editor, as shown in Figure B.1, go to:

Tools → Generate IFG

The tool analyses the scene, identifies interactable objects and interaction scripts, and automatically constructs the **Interaction Flow Graph (IFG)** used by XRINTTEST for guided exploration and testing.

B.3.2 Execute XRINTTEST

1. In the Unity scene *Hierarchy*, double-click the **XR Origin (XR Rig)** object at the root of the scene to select it.
2. In the *Inspector*, locate the **XRINTTEST (Script)** and **Random Baseline (Script)** components.
 - To run XRINTTEST, enable (tick) **XRINTTEST (Script)** and ensure **Random Baseline (Script)** is disabled (unticked).
 - To run the random baseline, disable **XRINTTEST (Script)** and enable **Random Baseline (Script)** instead.
3. Click the **Play** button in the Unity Editor to start the scene and execute the selected testing mode.

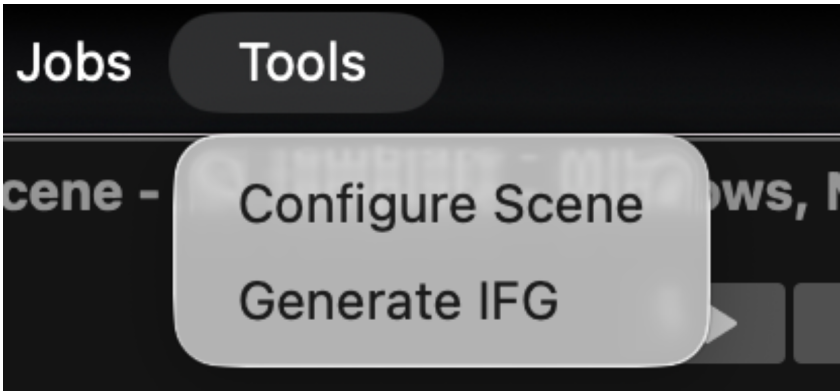


Figure B.1: Scene configuration and IFG generation are implemented as editor windows in Unity Editor

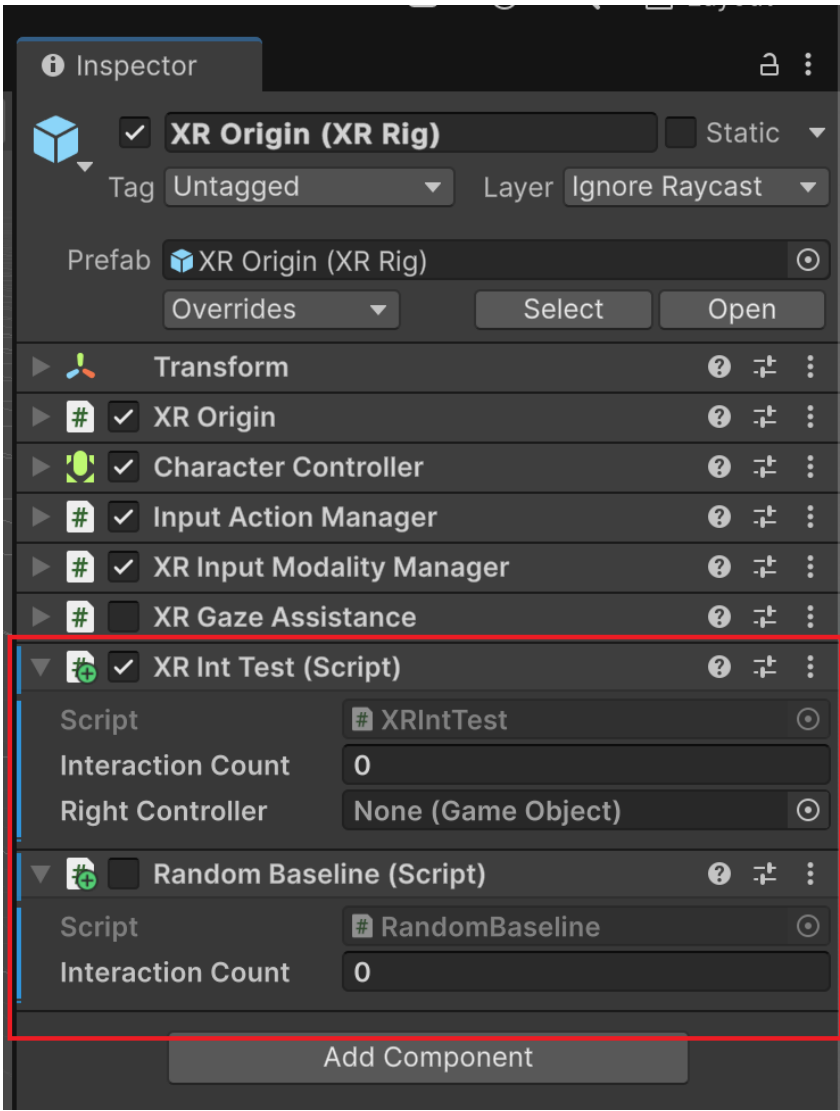


Figure B.2: Enter Caption

Appendix C

Applying INTENXION for Test Automation

This appendix provides practical guidance on how to install, configure, and develop test scripts with INTENXION on Unity-based XR projects (proposed in Chapter 4). It is a test automation framework for XR applications developed with Unity, providing a set of high-level testing APIs for scripting and validating XR user interactions. More details regarding the installation and execution could be accessed at <https://github.com/ruizhengu/IntenXion>.

C.1 Requirements

Before using INTENXION, ensure the following:

- A supported (e.g., [LTS](#)) Unity version is installed.
- The Unity Test Framework package is enabled in the project.
- XR Interaction Toolkit and associated XR plugins are installed and configured, consistent with the VR Template setup.

C.2 Project Setup

C.2.1 Integrating INTENXION into an Existing Project

If you have your own Unity XR project and wish to use INTENXION:

1. Clone the INTENXION repository:

```
git clone https://github.com/ruizhengu/IntenXion.git
```

2. Copy the scripts within the testing API folder `lib` into your project's `Assets/Scripts` directory.

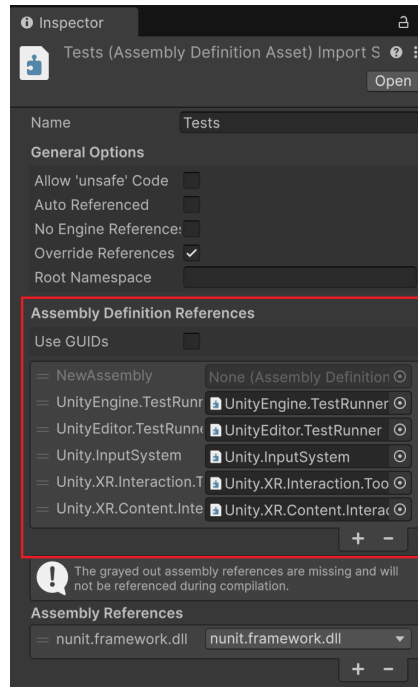


Figure C.1: Test Assembly

C.3 Test Assembly Configuration

To use INTENXION with the Unity Test Framework, configure a Play Mode test assembly:

1. In your Unity project, create a folder for tests, e.g., **Assets/Tests**.
2. Inside this folder, create an Assembly Definition file. Alternatively, copy the `/IntenXion/lib/Tests.asmdef` from the GitHub repository to the `/Assets/Tests` folder of your Unity Project.
3. Add necessary Assembly Definition References (i.e., dependencies of the tests). For instance, the tests require the dependency of `UnityEngine.XR.Interaction.Toolkit`¹, which could be added into the *Assembly Definition References* in the test assembly settings as shown in Figure C.1.

C.4 Writing Interaction Tests with INTENXION

The tests using INTENXION typically follow this structure:

1. Load a target XR scene.
2. Identify the subjects of the test, such as the user controller and the target object.
3. Use the provided APIs to drive controller-like actions (e.g., grab, release, trigger).
4. Assert expected interaction outcomes (e.g., object state changes, activation events).

¹<https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/api/UnityEngine.XR.Interaction.Toolkit.html>

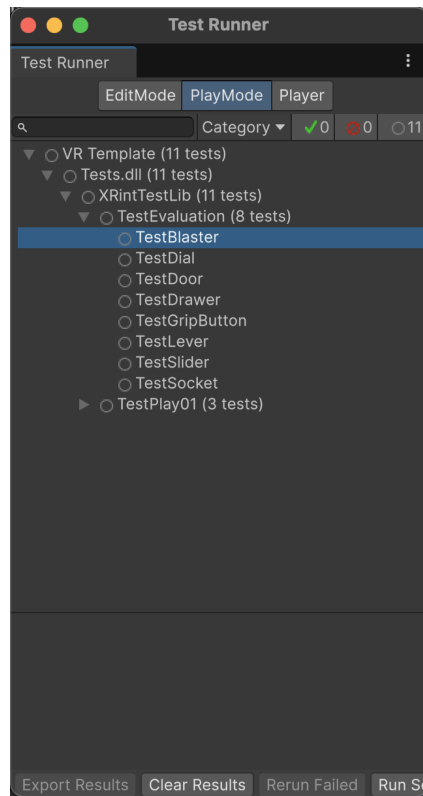


Figure C.2: Execute test cases

C.5 Running Tests

1. Open the Unity Test Runner: Window → General → Test Runner Window→General→Test Runner.
2. Switch to the *PlayMode* tab.
3. Select your INTENXION test assembly or individual test cases.
4. Click **Run All** or **Run Selected test(s)** to execute the interaction tests.

Test results (pass/fail) and any logged diagnostics will appear in the Test Runner and Console windows. Interaction-specific failures should correspond to assertion violations in the tests developed with INTENXION.

Bibliography

- [1] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino, “Augmented reality: A class of displays on the reality-virtuality continuum,” *Telem manipulator and Telepresence Technologies*, vol. 2351, 01 1994.
- [2] R. Suzuki, M. Gonzalez-Franco, M. Sra, and D. Lindlbauer, “Everyday AR through AI-in-the-Loop,” in *Proc. of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. ACM, 2025.
- [3] R. Gu, J. Zhang, J. M. Rojas, and D. Shin, “On the promises and challenges of ai-powered xr glasses as embodied software,” in *Proceedings of the 2nd ACM International Conference on AI-Powered Software*, 2025.
- [4] M. Abrash, “Creating the future: Augmented reality, the next human-machine interface,” in *IEEE Intl. Electron Devices Meeting (IEDM)*, 2021.
- [5] I. Rodriguez and X. Wang, “An empirical study of open source virtual reality software projects,” in *2017 ACM/IEEE Intl. Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2017, pp. 474–475.
- [6] S. Kavanagh, A. Luxton-Reilly, B. Wuensche, and B. Plimmer, “A systematic review of virtual reality in education,” *Themes in Science and Technology Education*, vol. 10, no. 2, pp. 85–119, 2017.
- [7] S. Tadeja, P. Seshadri, and P. Kristensson, “Aerovr: An immersive visualisation system for aerospace design and digital twinning in virtual reality,” *The Aeronautical Journal*, vol. 124, no. 1280, p. 1615–1635, 2020.
- [8] A. Lele, “Virtual reality and its military utility,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 4, no. 1, pp. 17–26, 2013.
- [9] Y. Kim, H. Kim, and Y. O. Kim, “Virtual Reality and Augmented Reality in Plastic Surgery: A Review,” *Archives of Plastic Surgery*, vol. 44, no. 3, pp. 179–187, 2017.
- [10] 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 *Symposium on Virtual and Augmented Reality (SVR)*, 2020, pp. 57–66.

- [11] 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.
- [12] 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.
- [13] R. Gu, J. M. Rojas, and D. Shin, "Software testing for extended reality applications: A systematic mapping study," *Automated Software Engineering*, 2025.
- [14] R. Doerner, W. Broll, P. Grimm, and B. Jung, Eds., *Virtual and Augmented Reality (VR/AR): Foundations and Methods of Extended Realities (XR)*. Springer Intl. Publishing, 2022.
- [15] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyandé, and J. Klein, "Automated testing of android apps: A systematic literature review," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 45–66, 2019.
- [16] T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, and Z. Su, "Guided, stochastic model-based gui testing of android apps," in *Proc. of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 2017, p. 245–256.
- [17] P. Bouvier, F. De Sorbier, P. Chaudéyrac, and V. Biri, "Cross benefits between virtual reality and games," in *Intl. Conf. and Industry Symposium on Computer Games, Animation, Multimedia, IPTV, Edutainment and Security (CGAT'08)*, 2008.
- [18] S. A. Andrade, F. L. S. Nunes, and M. E. Delamaro, "Exploiting deep reinforcement learning and metamorphic testing to automatically test virtual reality applications," *Software Testing, Verification and Reliability*, vol. 33, no. 8, 2023.
- [19] S. Zein, N. Salleh, and J. Grundy, "A systematic mapping study of mobile application testing techniques," *Journal of Systems and Software*, vol. 117, pp. 334–356, 2016.
- [20] V. Garousi, A. Mesbah, A. Betin-Can, and S. Mirshokraie, "A systematic mapping study of web application testing," *Information and Software Technology*, vol. 55, no. 8, pp. 1374–1396, 2013.
- [21] J. Roberts, "The ar/vr technology stack: A central repository of software development libraries, platforms, and tools," 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2305.07842>
- [22] J. Scheibmeir and Y. K. Malaiya, "Quality model for testing augmented reality applications," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 2019, pp. 0219–0226.
- [23] X. Qiao, P. Ren, S. Dustdar, L. Liu, H. Ma, and J. Chen, "Web ar: A promising future for mobile augmented reality—state of the art, challenges, and insights," *Proc. of the IEEE*, vol. 107, no. 4, pp. 651–666, 2019.
- [24] U. Technologies, "Unity - manual: Gameobject," 2024, accessed: 2024-08-13. [Online]. Available: <https://docs.unity3d.com/Manual/class-GameObject.html>

- [25] A. E. Walsh, "Understanding scene graphs," *Dr.Dobb's Journal*, vol. 27, no. 7, pp. 17–26, 2002.
- [26] X. Wang, T. Rafi, and N. Meng, "Vrguide: Efficient testing of virtual reality scenes via dynamic cut coverage," in *2023 38th IEEE/ACM Intl. Conf. on Automated Software Engineering (ASE)*. IEEE Computer Society, 2023, pp. 951–962.
- [27] Y. M. Kim, I. Rhiu, and M. H. Yun, "A systematic review of a virtual reality system from the perspective of user experience," *Intl. Journal of Human-Computer Interaction*, vol. 36, no. 10, pp. 893–910, 2020.
- [28] V. Emery, J. Jacko, T. Kongnakorn, V. Kuruchittham, S. Landry, G. Nickles, A. Sears, and J. Whittle, "Identifying critical interaction scenarios for innovative user modeling," in *Proc. of the 1st Intl. Conf. on Universal Access in Human-Computer Interaction*, 2001, pp. 481–485.
- [29] D. Breen, E. Rose, and R. Whitaker, "Interactive occlusion and collision of real and virtual objects in augmented reality," *Proc. of Eurographics Poitiers, France*, 2000.
- [30] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge University Press, 2008.
- [31] H. Washizaki, Ed., *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0*. IEEE Computer Society, 2024. [Online]. Available: <https://www.swebok.org>
- [32] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.
- [33] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surv.*, vol. 43, no. 2, 2011.
- [34] P. McMinn, "Search-based software testing: Past, present and future," in *2011 IEEE Fourth Intl. Conf. on Software Testing, Verification and Validation Workshops*, 2011, pp. 153–163.
- [35] M. Foster, C. Wild, R. M. Hierons, and N. Walkinshaw, "Causal test adequacy," in *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*, 2024.
- [36] A. C. Corrêa Souza, F. L. S. Nunes, and M. E. Delamaro, "An automated functional testing approach for virtual reality applications," *Software Testing, Verification and Reliability*, vol. 28, no. 8, 2018.
- [37] F. Molina, A. Gorla, and M. d'Amorim, "Test Oracle Automation in the Era of LLMs," *ACM Trans. Softw. Eng. Methodol.*, 2025.
- [38] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. ACM, 2014.
- [39] T. T. Chekam, M. Papadakis, Y. L. Traon, and M. Harman, "An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. IEEE Press, 2017.

- [40] 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.
- [41] M. Utting, A. Pretschner, and B. Legeard, “A taxonomy of model-based testing approaches,” *Software Testing, Verification and Reliability*, vol. 22, no. 5, pp. 297–312, 2012.
- [42] T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, and Z. Su, “Guided, stochastic model-based gui testing of android apps,” in *Proc. of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. ACM, 2017, p. 245–256.
- [43] Y. Jiang, X. Xiang, Q. Guo, Q. Gong, and X. Gong, “Adge: Automated directed gui explorer for android applications,” in *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)*, 2025.
- [44] D. Lai and J. Rubin, “Goal-driven exploration for android applications,” in *2019 34th IEEE/ACM Intl. Conference on Automated Software Engineering (ASE)*, 2019, pp. 115–127.
- [45] S. Yang, H. Wu, H. Zhang, Y. Wang, C. Swaminathan, D. Yan, and A. Rountev, “Static window transition graphs for android,” *Automated Software Engg.*, vol. 25, no. 4, p. 833–873, 2018.
- [46] R. Ferdous, F. Kifetew, D. Prandi, I. S. W. B. Prasetya, S. Shirzadehhajimahmood, and A. Susi, “Search-based automated play testing of computer games: A model-based approach,” in *Search-Based Software Engineering*, U.-M. O’Reilly and X. Devroey, Eds. Springer International Publishing, 2021, pp. 56–71.
- [47] I. S. W. B. Prasetya, F. Pastor Ricós, F. M. Kifetew, D. Prandi, S. Shirzadehhajimahmood, T. E. J. Vos, P. Paska, K. Hovorka, R. Ferdous, A. Susi, and J. Davidson, “An agent-based approach to automated game testing: an experience report,” in *Proceedings of the 13th International Workshop on Automating Test Case Design, Selection and Evaluation*, ser. A-TEST 2022. ACM, 2022, p. 1–8.
- [48] S. Ariyurek, A. Betin-Can, and E. Surer, “Automated video game testing using synthetic and humanlike agents,” *IEEE Transactions on Games*, vol. 13, no. 1, pp. 50–67, 2021.
- [49] A. C. Corrêa Souza, F. L. S. Nunes, and M. E. Delamaro, “An automated functional testing approach for virtual reality applications,” *Software Testing, Verification and Reliability*, vol. 28, no. 8, 2018.
- [50] A. E. Walsh, “Understanding scene graphs,” *Dr. Dobb’s Journal*, vol. 27, no. 7, p. 17, 2002.
- [51] Y. Hu, T. Azim, and I. Neamtiu, “Versatile yet lightweight record-and-replay for android,” *SIG-PLAN Not.*, vol. 50, no. 10, p. 349–366, 2015.
- [52] S. N. Modarressi, F. Maurer, and N. Wang, “Capture and replay testing tool for xr applications,” in *RealXR@AVI*, 2024. [Online]. Available: <https://api.semanticscholar.org/CorpusID:270563554>
- [53] W. Lam, Z. Wu, D. Li, W. Wang, H. Zheng, H. Luo, P. Yan, Y. Deng, and T. Xie, “Record and replay for android: are we there yet in industrial cases?” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. ACM, 2017, p. 854–859.

- [54] 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.
- [55] 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.
- [56] 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.*, no. FSE, 2024.
- [57] L. Ardito, R. Coppola, M. Morisio, and M. Torchiano, “Espresso vs. eyeautomate: An experiment for the comparison of two generations of android gui testing,” in *Proc. of the 23rd Intl. Conf. on Evaluation and Assessment in Software Engineering*, ser. EASE '19. ACM, 2019, p. 13–22.
- [58] C. Politowski, Y.-G. Guéhéneuc, and F. Petrillo, “Towards automated video game testing: still a long way to go,” in *Proc. of the 6th Intl. ICSE Workshop on Games and Software Engineering: Engineering Fun, Inspiration, and Motivation*, ser. GAS '22. ACM, 2022, p. 37–43.
- [59] Y. Zheng, X. Xie, T. Su, L. Ma, J. Hao, Z. Meng, Y. Liu, R. Shen, Y. Chen, and C. Fan, “Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning,” in *2019 34th IEEE/ACM Intl. Conf. on Automated Software Engineering (ASE)*, 2019, pp. 772–784.
- [60] F. Pastor Ricós, B. Marín, I. Prasetya, T. E. Vos, J. Davidson, and K. Hovorka, “Behavior driven development for 3d games,” *Data & Knowledge Engineering*, 2025.
- [61] M. Kempka, M. Wydmuch, G. Runc, J. Toczec, and W. Jaśkowski, “Vizdoom: A doom-based ai research platform for visual reinforcement learning,” in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016.
- [62] K. Petersen, S. Vakkalanka, and L. Kuzniarz, “Guidelines for conducting systematic mapping studies in software engineering: An update,” *Information and Software Technology*, vol. 64, pp. 1–18, 2015.
- [63] I. Börsting, M. Heikamp, M. Hesenius, W. Koop, and V. Gruhn, “Software engineering for augmented reality - a research agenda,” *Proc. ACM Hum.-Comput. Interact.*, vol. 6, 2022.
- [64] M. Kuri, S. A. Karre, and Y. R. Reddy, “Understanding software quality metrics for virtual reality products - a mapping study,” in *Proc. of the Innovations in Software Engineering Conference (Formerly Known as India Software Engineering Conference)*, ser. ISEC '21. ACM, 2021.
- [65] M. Hertzum, “Usability testing: A practitioner’s guide to evaluating the user experience,” *Synthesis Lectures on Human-Centered Informatics*, vol. 1, pp. i–105, 2020.
- [66] A. N. Ramaseri Chandra, F. El Jamiy, and H. Reza, “A review on usability and performance evaluation in virtual reality systems,” in *2019 Intl. Conf. on Computational Science and Computational Intelligence (CSCI)*, 2019, pp. 1107–1114.

- [67] C. Brown-Johnson, B. Berrean, and J. Cataldo, "Development and usability evaluation of the mhealth tool for lung cancer (mhealth tlc): A virtual world health game for lung cancer patients," *Patient Education and Counseling*, vol. 98, 01 2015.
- [68] L. Barbieri, F. Bruno, and M. Muzzupappa, "Virtual museum system evaluation through user studies," *Journal of Cultural Heritage*, vol. 26, pp. 101–108, 2017.
- [69] C. J. Chen, S. Y. Lau, K. M. Chuah, and C. S. Teh, "Group usability testing of virtual reality-based learning environments: A modified approach," *Procedia - Social and Behavioral Sciences*, vol. 97, pp. 691–699, 2013.
- [70] A. Dey, M. Billinghurst, R. W. Lindeman, and J. E. Swan, "A Systematic Review of 10 Years of Augmented Reality Usability Studies: 2005 to 2014," *Frontiers in Robotics and AI*, vol. 5, 2018.
- [71] S. Davis, K. Nesbitt, and E. Nalivaiko, "A systematic review of cybersickness," in *Proc. of the 2014 Conference on Interactive Entertainment*, ser. IE2014. ACM, 2014, p. 1–9.
- [72] K. S. B. Robert S. Kennedy, Norman E. Lane and M. G. Lilienthal, "Simulator sickness questionnaire: An enhanced method for quantifying simulator sickness," *The Intl. Journal of Aviation Psychology*, vol. 3, no. 3, pp. 203–220, 1993.
- [73] D. Kamińska, G. Zwoliński, and A. Laska-Leśniewicz, "Usability testing of virtual reality applications—the pilot study," *Sensors*, vol. 22, no. 4, 2022.
- [74] C. Qu, X. Che, S. Ma, and S. Zhu, "Bio-physiological-signals-based vr cybersickness detection," *CCF Transactions on Pervasive Computing and Interaction*, vol. 4, 2022.
- [75] R. K. Kundu, O. Y. Elsaid, P. Calyam, and K. A. Hoque, "Vr-lens: Super learning-based cybersickness detection and explainable ai-guided deployment in virtual reality," in *Proc. of the 28th Intl. Conf. on Intelligent User Interfaces*, ser. IUI '23. ACM, 2023, p. 819–834.
- [76] A. H. X. Yang, N. Kasabov, and Y. O. Cakmak, "Machine learning methods for the study of cybersickness: A systematic review," *Brain Informatics*, vol. 9, no. 1, p. 24, 2022.
- [77] X. Zhang, J. Tao, K. Tan, M. Törngren, J. M. G. Sánchez, M. R. Ramli, X. Tao, M. Gyllenhammar, F. Wotawa, N. Mohan, M. Nica, and H. Felbinger, "Finding critical scenarios for automated driving systems: A systematic mapping study," *IEEE Transactions on Software Engineering*, vol. 49, no. 3, pp. 991–1026, 2023.
- [78] B. A. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele University, Tech. Rep. EBSE-2007-01, 2007.
- [79] O. Charron, A. Lallement, D. Jarnet, V. Noblet, J.-B. Clavier, and P. Meyer, "Automatic detection and segmentation of brain metastases on multimodal mr images with a deep convolutional neural network," *Computers in Biology and Medicine*, vol. 95, pp. 43–54, 2018.
- [80] X. Wang, "VRTest: An extensible framework for automatic testing of virtual reality scenes," in *Intl. Conf. on Software Engineering: Companion Proceedings (ICSE-Companion)*. ACM, 2022, pp. 232–236.

- [81] D. E. Rzig, N. Iqbal, I. Attisano, X. Qin, and F. Hassan, "Virtual reality (vr) automated testing in the wild: A case study on unity-based vr applications," in *Proc. of the 32nd ACM SIGSOFT Intl. Symposium on Software Testing and Analysis*, ser. ISSTA 2023. ACM, 2023, p. 1269–1281.
- [82] T. Rafi, X. Zhang, and X. Wang, "Predart: Towards automatic oracle prediction of object placements in augmented reality testing," in *Proc. of the 37th IEEE/ACM Intl. Conf. on Automated Software Engineering*. ACM, 2023.
- [83] A. Bierbaum, P. Hartling, and C. Cruz-Neira, "Automated testing of virtual reality application interfaces," in *Proc. of the Workshop on Virtual Environments 2003*, ser. EGVE '03. ACM, 2003, p. 107–114.
- [84] S. Li, Y. Wu, Y. Liu, D. Wang, M. Wen, Y. Tao, Y. Sui, and Y. Liu, "An exploratory study of bugs in extended reality applications on the web," in *2020 IEEE 31st Intl. Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 172–183.
- [85] J. Priem, H. Piwowar, and R. Orr, "Openalex: A fully-open index of scholarly works, authors, venues, institutions, and concepts," 2022. [Online]. Available: <https://arxiv.org/abs/2205.01833>
- [86] P. Tramontana, D. Amalfitano, N. Amatucci, and A. R. Fasolino, "Automated functional testing of mobile applications: A systematic mapping study," *Software Quality Journal*, vol. 27, no. 1, pp. 149–201, 2019.
- [87] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proc. of the 12th Intl. Conf. on Evaluation and Assessment in Software Engineering*, ser. EASE'08. BCS Learning & Development Ltd., 2008, p. 68–77.
- [88] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proc. of the 18th Intl. Conf. on Evaluation and Assessment in Software Engineering*, ser. EASE '14. ACM, 2014.
- [89] S.-m. Jung, S.-H. Oh, and T.-k. Whangbo, "360° stereo image based vr motion sickness testing system," in *2017 Intl. Conf. on Emerging Trends & Innovation in ICT (ICEI)*, 2017, pp. 150–153.
- [90] H. G. Kim, W. J. Baddar, H.-t. Lim, H. Jeong, and Y. M. Ro, "Measurement of exceptional motion in vr video contents for vr sickness assessment using deep convolutional autoencoder," in *Proc. of the 23rd ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '17. ACM, 2017.
- [91] S. M. Lehman, A. S. Alrumayh, K. Kolhe, H. Ling, and C. C. Tan, "Hidden in plain sight: Exploring privacy risks of mobile augmented reality applications," *ACM Trans. Priv. Secur.*, vol. 25, no. 4, 2022.
- [92] S. Valluripally, B. Frailey, B. Kruse, B. Palipatana, R. Oruche, A. Gulhane, K. A. Hoque, and P. Calyam, "Detection of security and privacy attacks disrupting user immersive experience in virtual reality learning environments," *IEEE Transactions on Services Computing*, vol. 16, no. 4, pp. 2559–2574, 2023.

- [93] P. Casey, I. Baggili, and A. Yarramreddy, "Immersive virtual reality attacks and the human joystick," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 2, pp. 550–562, 2021.
- [94] T. Figueira and A. Gil, "Youkai: A cross-platform framework for testing vr/ar apps," in *HCI Intl. 2022 – Late Breaking Papers: Interacting with eXtended Reality and Artificial Intelligence*. Springer Nature Switzerland, 2022, pp. 3–12.
- [95] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: A proposal and a discussion," *Requir. Eng.*, vol. 11, pp. 102–107, 03 2006.
- [96] Richard Gunawan, Yohanes Priadi Wibisono, Clara Hetty Primasari, and Djoko Budiyo, "Blackbox Testing on Virtual Reality Gamelan Saron Using Equivalence Partition Method," *Jurnal Buana Informatika*, vol. 14, no. 01, pp. 11–19, 2023.
- [97] I. S. W. B. Prasetya, S. Shirzadehhajimahmood, S. G. Ansari, P. Fernandes, and R. Prada, "An agent-based architecture for ai-enhanced automated testing for xr systems, a short paper," in *Intl. Conf. on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2021, pp. 213–217.
- [98] F. Kilger, A. Kabil, V. Tippmann, G. Klinker, and M.-O. Pahl, "Detecting and preventing faked mixed reality," in *2021 IEEE 4th Intl. Conf. on Multimedia Information Processing and Retrieval (MIPR)*, 2021, pp. 399–405.
- [99] S. M. Lehman, S. Elezovikj, H. Ling, and C. C. Tan, "Archie++ : A cloud-enabled framework for conducting ar system testing in the wild," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 4, pp. 2102–2116, 2023.
- [100] V. Mahadevan, W. Li, V. Bhalodia, and N. Vasconcelos, "Anomaly detection in crowded scenes," in *IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, 2010, pp. 1975–1981.
- [101] C. Lu, J. Shi, and J. Jia, "Abnormal event detection at 150 fps in matlab," in *IEEE Intl. Conf. on Computer Vision*, 2013, pp. 2720–2727.
- [102] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The Intl. Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.
- [103] S. Herbold and P. Harms, "Autoquest – automated quality engineering of event-driven software," in *2013 IEEE Sixth Intl. Conf. on Software Testing, Verification and Validation Workshops*, 2013, pp. 134–139.
- [104] T. E. J. Vos, P. Aho, F. Pastor Ricos, O. Rodriguez-Valdes, and A. Mulders, "testar – scriptless testing through graphical user interface," *Software Testing, Verification and Reliability*, vol. 31, no. 3, p. e1771, 2021.
- [105] F. Pastor Ricós, "Scriptless testing for extended reality systems," in *Research Challenges in Information Science*. Springer Intl. Publishing, 2022, pp. 786–794.

- [106] M. Hesenius, T. Griebel, S. Gries, and V. Gruhn, “Automating ui tests for mobile applications with formal gesture descriptions,” in *Proc. of the 16th Intl. Conf. on Human-Computer Interaction with Mobile Devices & Services*, ser. MobileHCI ’14. ACM, 2014, p. 213–222.
- [107] T. Su, Y. Yan, J. Wang, J. Sun, Y. Xiong, G. Pu, K. Wang, and Z. Su, “Fully automated functional fuzzing of android apps for detecting non-crashing logic bugs,” *Proc. ACM Program. Lang.*, vol. 5, no. OOPSLA, 2021.
- [108] S. Cheng and H. Qu, “Key Issues of Real-time Collision Detection in Virtual Reality,” *Intl. Journal of Frontiers in Engineering Technology*, vol. 3, no. 5, 2021.
- [109] Z. Wei and G. Xinlin, “The collision detection algorithm in virtual reality,” in *2012 Intl. Conf. on Computer Science and Electronics Engineering*, vol. 3, 2012, pp. 538–541.
- [110] P. Xu and Q. Sun, “Virtual reality collision detection based on improved ant colony algorithm,” *Applied Sciences*, vol. 13, no. 11, 2023.
- [111] Y. Jin, J. Geng, Z. He, C. Lv, and T. Zhao, “A capsule-based collision detection approach of irregular objects in virtual maintenance,” *Assembly Automation*, vol. 41, no. 1, pp. 89–105, 2021.
- [112] M. Zhang, D. Zhou, C. Lv, and X. Li, “Collision detection technology based on capsule model in virtual maintenance,” in *Intl. Conf. on Reliability, Maintainability and Safety (ICRMS)*, 2014, pp. 1150–1155.
- [113] K. Kia, J. Hwang, and J. H. Kim, “Effects of error rates and target sizes on neck and shoulder biomechanical loads during augmented reality interactions,” *Applied Ergonomics*, vol. 113, p. 104107, 2023.
- [114] F. Pastore, L. Mariani, and G. Fraser, “Crowdoracles: Can the crowd solve the oracle problem?” in *2013 IEEE Sixth Intl. Conf. on Software Testing, Verification and Validation*, 2013, pp. 342–351.
- [115] P. Thomas, S. Spielman, N. Craswell, and B. Mitra, “Large language models can accurately predict searcher preferences,” in *Proc. of the 47th Intl. ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR ’24. ACM, 2024, p. 1930–1940.
- [116] Z. Liu, C. Li, C. Chen, J. Wang, B. Wu, Y. Wang, J. Hu, and Q. Wang, “Vision-driven automated mobile gui testing via multimodal large language model,” 2024. [Online]. Available: <https://arxiv.org/abs/2407.03037>
- [117] Z. Qiu, W. Liu, H. Feng, Z. Liu, T. Z. Xiao, K. M. Collins, J. B. Tenenbaum, A. Weller, M. J. Black, and B. Schölkopf, “Can large language models understand symbolic graphics programs?” 2024. [Online]. Available: <https://arxiv.org/abs/2408.08313>
- [118] C. Paduraru, A. Stefanescu, and A. Jianu, “Unit test generation using large language models for unity game development,” in *Proc. of the 1st ACM Intl. Workshop on Foundations of Applied Software Engineering for Games*, ser. FaSE4Games 2024. ACM, 2024, p. 7–13.
- [119] Google, “UIAutomator,” <https://developer.android.com/training/testing/other-components/ui-automator>, accessed: 2025-07-24.

- [120] Appium, “Appium,” <https://appium.io/>, accessed: 2025-07-24.
- [121] R. Gu and J. M. Rojas, “A test automation framework for user interaction in extended reality applications,” in *40th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, 2025.
- [122] A. Plopski, T. Hirzle, N. Norouzi, L. Qian, G. Bruder, and T. Langlotz, “The eye in extended reality: A survey on gaze interaction and eye tracking in head-worn extended reality,” *ACM Comput. Surv.*, 2022.
- [123] 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.
- [124] 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.
- [125] —, “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.
- [126] —, “Object Interaction, Part 3: Releasing Objects,” <https://developers.meta.com/horizon/blog/object-interaction-part-3-releasing-objects/>, 2017, accessed: 2025-07-24.
- [127] —, “Object Interaction, Part 4: Constrained Interactions,” <https://developers.meta.com/horizon/blog/-object-interaction-part-4-constrained-interactions/>, 2017, accessed: 2025-07-24.
- [128] Unity Technologies, “Unity Test Framework,” <https://docs.unity3d.com/Packages/com.unity.test-framework@2.0/>, accessed: 2025-07-24.
- [129] —. 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.
- [130] Appium, “Actions,” <https://appium.github.io/appium.io/docs/en/commands/interactions/actions/>, accessed: 2025-07-24.
- [131] 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.
- [132] Unity Technologies, “Interaction Events,” <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.1/manual/interactable-events.html>, accessed: 2025-07-24.
- [133] —, “VR Template Quick Start Guide,” <https://docs.unity3d.com/Packages/com.unity.template.vr@9.0/>, accessed: 2025-07-24.
- [134] —, “XR Interaction Toolkit Examples,” <https://github.com/Unity-Technologies/XR-Interaction-Toolkit-Examples>, accessed: 2025-07-24.

- [135] 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.
- [136] Unity Technologies, “Text-based Scene Files,” <https://docs.unity3d.com/6000.2/Documentation/Manual/TextSceneFormat.html>, accessed: 2025-07-24.
- [137] S. A. de Andrade, F. L. S. Nunes, and M. E. Delamaro, “Exploiting deep reinforcement learning and metamorphic testing to automatically test virtual reality applications,” *Software Testing, Verification and Reliability (STVR)*, vol. 33, no. 8, p. e1863, 2023.
- [138] R. Gu, J. M. Rojas, and D. Shin, “XRintTest: An automated framework for user interaction testing in extended reality applications,” in *40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2025.
- [139] —, “Automated testing of prevalent 3d user interactions in virtual reality applications,” *Automated Software Engineering*, 2026.
- [140] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta, and A. M. Memon, “Mobiguitar: Automated model-based testing of mobile apps,” *IEEE Software*, vol. 32, no. 5, pp. 53–59, 2015.
- [141] D. Reid, “A model of playfulness and flow in virtual reality interactions,” *Presence: Teleoperators and Virtual Environments*, vol. 13, no. 4, pp. 451–462, 08 2004.
- [142] A. G. Clark, M. Foster, B. Prifling, N. Walkinshaw, R. M. Hierons, V. Schmidt, and R. D. Turner, “Testing causality in scientific modelling software,” *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 1, 2023.
- [143] T. Fulcini, R. Coppola, M. Torchiano, and L. Ardito, “An analysis of widget layout attributes to support android gui-based testing,” in *IEEE Intl. Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2023, pp. 117–125.
- [144] I. Arcuschin, C. Ciccaroni, J. P. Galeotti, and J. M. Rojas, “On the feasibility and challenges of synthesizing executable espresso tests,” in *IEEE/ACM Intl. Conference on Automation of Software Test (AST)*, 2022, pp. 92–102.
- [145] T. Su, Y. Yan, J. Wang, J. Sun, Y. Xiong, G. Pu, K. Wang, and Z. Su, “Fully automated functional fuzzing of android apps for detecting non-crashing logic bugs,” *Proc. ACM Program. Lang.*, vol. 5, no. OOPSLA, 2021.
- [146] Y. Jin, G. Jie, Z. He, C. Lv, and T. Zhao, “A capsule-based collision detection approach of irregular objects in virtual maintenance,” *Assembly Automation*, vol. ahead-of-print, 01 2021.
- [147] R. Coppola, T. Fulcini, S. Manzi, and F. Strada, “How to measure game testing: a survey of coverage metrics,” in *Proceedings of the ACM/IEEE 8th International Workshop on Games and Software Engineering*, ser. GAS ’24. ACM, 2024, p. 15–19.
- [148] T. Wetzlmaier, R. Ramler, and W. Putschögl, “A framework for monkey gui testing,” in *IEEE Intl. Conference on Software Testing, Verification and Validation (ICST)*, 2016, pp. 416–423.

- [149] Z. Zhu, H.-N. Dai, H. Guo, Z. Liao, and Z. Zheng, “Vrexplorer: A model-based approach for semi-automated testing of virtual reality scenes,” in *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Press, 2025.
- [150] H. Liu, Q. Liu, Z. Niu, and Y. Liu, “Dynamic and automatic feedback-based threshold adaptation for code smell detection,” *IEEE Transactions on Software Engineering*, 2016.
- [151] D. Spadini, F. Palomba, A. Zaidman, M. Bruntink, and A. Bacchelli, “On the relation of test smells to software code quality,” in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2018.
- [152] B. Yang, Z. Xing, X. Xia, C. Chen, D. Ye, and S. Li, “Uis-hunter: Detecting ui design smells in android apps,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2021.
- [153] S. Li, B. Li, Y. Liu, C. Gao, J. Zhang, S.-C. Cheung, and M. R. Lyu, “Grounded gui understanding for vision based spatial intelligent agent: Exemplified by virtual reality apps,” 2024. [Online]. Available: <https://arxiv.org/abs/2409.10811>
- [154] A. Plopski, T. Hirzle, N. Norouzi, L. Qian, G. Bruder, and T. Langlotz, “The eye in extended reality: A survey on gaze interaction and eye tracking in head-worn extended reality,” *ACM Comput. Surv.*, vol. 55, no. 3, Mar. 2022.
- [155] S. Li, Y. Wu, Y. Liu, D. Wang, M. Wen, Y. Tao, Y. Sui, and Y. Liu, “An exploratory study of bugs in extended reality applications on the web,” in *IEEE 31st Intl. Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 172–183.
- [156] E. Waisberg, J. Ong, M. Masalkhi, N. Zaman, P. Sarker, and A. Tavakkoli, “Meta smart glasses—large language models and the future for assistive glasses for individuals with vision impairments,” *Eye*, 2023.
- [157] L. Stappen, G. Rizos, and B. Schuller, “X-AWARE: ConteXt-AWARE human-environment attention fusion for driver gaze prediction in the wild,” in *Proc. of the 2020 Intl. Conference on Multimodal Interaction (ICMI)*. ACM, 2020.
- [158] Z.-M. Wang, M.-H. Rao, S.-H. Ye, W.-T. Song, and F. Lu, “Towards spatial computing: recent advances in multimodal natural interaction for extended reality headsets,” *Frontiers of Computer Science*, 06 2025.
- [159] I. Börsting, M. Heikamp, M. Hesenius, W. Koop, and V. Gruhn, “Software engineering for augmented reality - a research agenda,” *Proc. ACM Hum.-Comput. Interact.*, no. EICS, 2022.
- [160] S. D. Pendleton *et al.*, “Perception, planning, control, and coordination for autonomous vehicles,” *Machines*, 2017.
- [161] D. Wu, F. Liu, Y.-H. Hung, and Y. Duan, “Spatial-MLLM: Boosting MLLM capabilities in visual-based spatial intelligence,” 2025. [Online]. Available: <https://arxiv.org/abs/2505.23747>
- [162] B. Zeng, “Recent advances and future directions in extended reality (XR): Exploring AI-Powered spatial intelligence,” 2025. [Online]. Available: <https://arxiv.org/abs/2504.15970>

- [163] J. Parker-Holder *et al.*, “Genie 2: A large-scale foundation world model,” 2024. [Online]. Available: <https://deepmind.google/discover/blog/genie-2-a-large-scale-foundation-world-model/>
- [164] R. Suzuki, M. Gonzalez-Franco, M. Sra, and D. Lindlbauer, “XR and AI: Ai-enabled virtual, augmented, and mixed reality,” in *Adjunct Proc. of the 36th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2023.
- [165] Y. Tang, W. Huang, Y. Wang, C. Li, R. Yuan, R. Zhang, J. Wu, and L. Fei-Fei, “UAD: Unsupervised affordance distillation for generalization in robotic manipulation,” 2025. [Online]. Available: <https://arxiv.org/abs/2506.09284>
- [166] K. Y. Lam, L. H. Lee, and P. Hui, “A2W: Context-aware recommendation system for mobile augmented reality web browser,” in *Proc. of the 29th ACM Intl. Conference on Multimedia (MM)*. ACM, 2021.
- [167] M. Lachmair, M. Fischer, and P. Gerjets, “Action-control mappings of interfaces in virtual reality: A study of embodied interaction,” *Frontiers in Virtual Reality*, 2022.
- [168] M. B. I. Reaz, M. S. Hussain, and F. Mohd-Yasin, “Techniques of emg signal analysis: detection, processing, classification and applications,” *Biological procedures online*, 2006.
- [169] A. Wazwaz, K. Amin, N. Semary, and T. Ghanem, “Dynamic and distributed intelligence over smart devices, internet of things edges, and cloud computing for human activity recognition using wearable sensors,” *Journal of Sensor and Actuator Networks*, 2024.
- [170] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, “A survey on large language model (llm) security and privacy: The good, the bad, and the ugly,” *High-Confidence Computing*, 2024.
- [171] F. Roesner and T. Kohno, “Security and privacy for augmented reality: Our 10-year retrospective,” in *VR4Sec: 1st Intl. Workshop on Security for XR and XR for Security*, 2023.
- [172] K. Ren, Q. Wang, C. Wang, Z. Qin, and X. Lin, “The security of autonomous driving: Threats, defenses, and future directions,” *Proc. of the IEEE*, 2020.
- [173] H. Zhang *et al.*, “Badrobot: Jailbreaking embodied LLM agents in the physical world,” in *Intl. Conference on Learning Representations (ICLR)*, 2025.
- [174] X. Zhang, T. Rafi, Y. Guan, S. Li, and M. R. Lyu, “Understanding the privacy-realisticness dilemma of the metaverse,” *Automated Software Engineering*, 2025.
- [175] X. Pan, M. Zhang, S. Ji, and M. Yang, “Privacy risks of general-purpose language models,” in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.
- [176] X. Zhang, R. Slavin, X. Wang, and J. Niu, “Privacy assurance for android augmented reality apps,” in *2019 IEEE 24th Pacific Rim Intl. Symposium on Dependable Computing (PRDC)*, 2019.
- [177] X. Yang and X. Zhang, “A study of user privacy in android mobile AR apps,” in *Proc. of the 37th IEEE/ACM Intl. Conference on Automated Software Engineering (ICSE)*. ACM, 2023.

- [178] Apple Inc., “Apple vision pro privacy overview,” https://www.apple.com/privacy/docs/Apple_Vision_Pro_Privacy_Overview.pdf, 2024, accessed: June 2025.
- [179] C. Luo, J. Goncalves, E. Velloso, and V. Kostakos, “A survey of context simulation for testing mobile context-aware applications,” *ACM Comput. Surv.*, 2020.
- [180] 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 (ISSTA)*. ACM, 2024.
- [181] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An open urban driving simulator,” 2017. [Online]. Available: <https://arxiv.org/abs/1711.03938>
- [182] S. Atakishiyev, M. Salameh, H. Yao, and R. Goebel, “Explainable artificial intelligence for autonomous driving: A comprehensive overview and field guide for future research directions,” *IEEE Access*, 2024.
- [183] X. Xu *et al.*, “XAIR: A framework of explainable ai in augmented reality,” in *Proc. of the 2023 CHI Conference on Human Factors in Computing Systems*. ACM, 2023.
- [184] S. Ouyang, J. M. Zhang, M. Harman, and M. Wang, “An empirical study of the non-determinism of ChatGPT in code generation,” *ACM Trans. Softw. Eng. Methodol. (TOSEM)*, 2025.
- [185] G. K. Santhanam, A. Alami-Idrissi, N. Mota, A. Schumann, and I. Giurgiu, “On evaluating explainability algorithms,” 2020. [Online]. Available: <https://openreview.net/forum?id=B1xBAA4FwH>
- [186] A. M. Salih, Z. Raisi-Estabragh, I. B. Galazzo, P. Radeva, S. E. Petersen, K. Lekadir, and G. Menegaz, “A perspective on explainable artificial intelligence methods: SHAP and LIME,” *Advanced Intelligent Systems*, 2025.
- [187] H. Sun, C. M. Poskitt, Y. Sun, J. Sun, and Y. Chen, “ACAV: A framework for automatic causality analysis in autonomous vehicle accident recordings,” in *Proc. of the IEEE/ACM 46th Intl. Conference on Software Engineering (ICSE)*. ACM, 2024.
- [188] G. P. Garvey, “Dissociation in virtual reality: depersonalization and derealization,” in *The Engineering Reality of Virtual Reality 2010*, I. E. McDowall and M. Dolinsky, Eds., Intl. Society for Optics and Photonics. SPIE, 2010.
- [189] S. J. Barnes, A. D. Pressey, and E. Scornavacca, “Mobile ubiquity: Understanding the relationship between cognitive absorption, smartphone addiction and social network services,” *Computers in Human Behavior*, 2019.
- [190] G. Jain, L. Findlater, and C. Gleason, “Scenescout: Towards ai agent-driven access to street view imagery for blind users,” 2025. [Online]. Available: <https://arxiv.org/abs/2504.09227>
- [191] M. M. Eler, J. M. Rojas, Y. Ge, and G. Fraser, “Automated accessibility testing of mobile apps,” in *IEEE 11th Intl. Conference on Software Testing, Verification and Validation (ICST)*, 2018.

- [192] F. Xie, A. Chowdhury, M. C. De Paolis Kaluza, L. Zhao, L. L. Wong, and R. Yu, “Deep imitation learning for bimanual robotic manipulation,” in *Proc. of the 34th Intl. Conference on Neural Information Processing Systems*, ser. NIPS ’20. Curran Associates Inc., 2020.
- [193] A. Romdhana, A. Merlo, M. Ceccato, and P. Tonella, “Deep reinforcement learning for black-box testing of android apps,” *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 4, 2022.
- [194] T. Su, Y. Yan, J. Wang, J. Sun, Y. Xiong, G. Pu, K. Wang, and Z. Su, “Fully automated functional fuzzing of android apps for detecting non-crashing logic bugs,” *Proc. ACM Program. Lang.*, vol. 5, no. OOPSLA, 2021. [Online]. Available: <https://doi.org/10.1145/3485533>
- [195] F. Zhang, B. Chen, Y. Zhang, J. Keung, J. Liu, D. Zan, Y. Mao, J.-G. Lou, and W. Chen, “RepoCoder: Repository-level code completion through iterative retrieval and generation,” *arXiv preprint arXiv:2303.12570*, 2023.
- [196] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, “SWE-agent: Agent-computer interfaces enable automated software engineering,” *arXiv preprint arXiv:2405.15793*, 2024.
- [197] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 9459–9474.
- [198] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, “ReAct: Synergizing reasoning and acting in language models,” in *International Conference on Learning Representations*, 2023.
- [199] T. Kanstrén and O.-P. Puolitaival, “Using built-in domain-specific modeling support to guide model-based test generation,” in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, 2012, pp. 436–445.
- [200] A. Dwarakanath, D. Era, A. Priyadarshi, N. Dubash, and S. Podder, “Accelerating test automation through a domain specific language,” in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops*, 2018, pp. 226–231.
- [201] M. Mernik, J. Heering, and A. M. Sloane, “When and how to develop domain-specific languages,” *ACM Computing Surveys*, vol. 37, no. 4, pp. 316–344, 2005.
- [202] M. F. Roslan, J. M. Rojas, and P. McMinn, “Private-keep out? understanding how developers account for code visibility in unit testing,” in *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2024.
- [203] S. Cao, R. Chen, W. Yang, M. Pan, and X. Li, “A mixed-methods study of model-based gui testing in real-world industrial settings,” *Proc. ACM Softw. Eng.*, no. FSE, 2025. [Online]. Available: <https://doi.org/10.1145/3715789>
- [204] X. Yang and X. Zhang, “A study of user privacy in android mobile ar apps,” in *Proc. of the 37th IEEE/ACM Intl. Conf. on Automated Software Engineering (ASE’22)*. ACM, 2023.

- [205] P. Tramontana, M. D. Luca, and A. R. Fasolino, "An approach for model based testing of augmented reality applications," in *RCIS Workshops*, 2022.
- [206] R. R. Kirayeva, M. R. Khafizov, T. T. Turdiev, and V. V. Kugurakova, "Automated testing of functional requirements for virtual reality applications," in *2023 IEEE XVI Intl. Scientific and Technical Conference Actual Problems of Electronic Instrument Engineering (APEIE)*, 2023, pp. 1760–1764.
- [207] P. Harms, "Automated usability evaluation of virtual reality applications," *ACM Trans. Comput.-Hum. Interact.*, vol. 26, no. 3, 2019.
- [208] A. Leykin and M. Tuceryan, "Automatic determination of text readability over textured backgrounds for augmented reality systems," in *IEEE/ACM Intl. Symposium on Mixed and Augmented Reality*, 2004, pp. 224–230.
- [209] B. Odeleye, G. Loukas, R. Heartfield, and F. Spyridonis, "Detecting framerate-oriented cyber attacks on user experience in virtual reality," in *1st Intl. Workshop on Security for XR and XR for Security*, 2021.
- [210] X. Qin and F. Hassan, "Dytrec: A dynamic testing recommendation tool for unity-based virtual reality software," in *Proc. of the 37th IEEE/ACM Intl. Conf. on Automated Software Engineering*. New York, NY, USA: ACM, 2023.
- [211] B. Sarupuri, S. Hoermann, M. C. Whitton, and R. W. Lindeman, "Lute: A locomotion usability test environment for virtual reality," in *2018 10th Intl. Conf. on Virtual Worlds and Games for Serious Applications (VS-Games)*, 2018, pp. 1–4.
- [212] S. Sendari, A. Firmansah, and Aripriharta, "Performance analysis of augmented reality based on vuforia using 3d marker detection," in *2020 4th Intl. Conf. on Vocational Education and Training (ICOVET)*, 2020, pp. 294–298.
- [213] C. G. Corrêa, M. E. Delamaro, M. L. Chaim, and F. L. S. Nunes, "Software testing automation of vr-based systems with haptic interfaces," *The Computer Journal*, vol. 64, no. 5, pp. 826–841, 2020.
- [214] S. Minor, V. K. Ketoma, and G. Meixner, "Test automation for augmented reality applications: a development process model and case study," *i-com*, vol. 22, no. 3, pp. 175–192, 2023.
- [215] S. A. Andrade, F. L. S. Nunes, and M. E. Delamaro, "Towards the systematic testing of virtual reality programs," in *2019 21st Symposium on Virtual and Augmented Reality (SVR)*, 2019, pp. 196–205.
- [216] U. U. Izuazu, D.-S. Kim, and J. M. Lee, "Unravelling the black box: Enhancing virtual reality network security with interpretable deep learning-based intrusion detection system," in *Intl. Conf. on Information and Communication Technology Convergence (ICTC)*, 2023, pp. 928–931.