ARTRIP: Automatic AR Testing with Randomized Interaction Patterns

Maria Rivera, Lisette Isais, and Xiaoyin Wang
The University of Texas at San Antonio,
Texas, USA,
maria.rivera@utsa.edu, lisette.isais@utsa.edu, xiaoyin.wang@utsa.edu

***Abstract*—Augmented Reality (AR) applications increasingly permeate domains such as gaming, retail, education, and healthcare. Despite their rapid adoption, systematic testing of AR apps remains underexplored due to their dependence on complex real-world contexts, sensor data, and diverse user interactions. In this paper, we propose *ARTRIP* (Automatic AR Testing with Randomized Interaction Patterns), a novel testing technique designed to explore AR applications with randomized interaction patterns. Unlike existing random testing approaches such as Monkey, ARTRIP uses a randomized interaction pattern to enhance the chance of covering more complicated interaction sequences. We describe the methodology, present a prototype implementation, and evaluate its effectiveness through case studies on four popular AR apps. Results suggest that ARTRIP achieves higher coverage than Monkey, highlighting its potential as a practical AR testing framework.**

I. Introduction

Augmented Reality (AR) overlays digital content onto physical environments, enabling immersive and interactive user experiences. Applications such as Pokemon Go, IKEA Place, and medical visualization tools demonstrate AR's potential across diverse domains. However, testing AR applications remains difficult because their correctness depends not only on software logic but also on environmental context (lighting, surfaces, motion), multi-modal sensor inputs, and unpredictable user behaviors.

Despite increasing interest in AR, the field of AR testing is still in its formative stages. Testing engineers often rely on ad-hoc heuristics, lacking formalized standards for evaluating AR systems. Significant challenges include handling non-standard, 3D interaction paradigms beyond traditional WIMP interfaces, dealing with sensor inaccuracies (e.g., gyroscope drift, accelerometer noise), and accommodating the vast fragmentation in AR-capable hardware. Current best practices advocate for multi-dimensional testing—covering functional, compatibility, usability, performance, and environmental robustness—but typically still depend on manual testing and physical device experiments due to simulation limitations [1]. Furthermore, usability evaluations in AR often lack standardized instruments, particularly in specialized domains like education and training [1].

Modern AR frameworks, such as Unity with AR Foundation, support the use of simulated VR scenes to facilitate testing of AR applications. Instead of relying solely on real-world camera input, developers can create synthetic 3D environments (e.g., virtual rooms, furniture layouts, or outdoor settings) that emulate realistic surfaces, lighting conditions, and object interactions. These simulated VR scenes can be replayed deterministically, ensuring that every testing session is conducted under identical conditions, which is critical for reproducibility. One key strength of this approach is that it enables systematic evaluation of AR features, such as object placement and tracking, without the variability introduced by physical surroundings. In addition, it allows rapid iteration: developers can script a wide range of test environments to expose corner cases that may be difficult or impractical to reproduce in reality.

While simulated scenes make automatic AR testing possible, there are still major challenges on performing more effective automatic testing. While Monkey is applicable, it generates random UI events without considering the visual or spatial context. Since AR interactions depend on camera input, object recognition, and gestures like swiping or pinching on specific areas, Monkey cannot align its events with the content of the playback. As a result, it fails to trigger realistic AR behaviors or reproduce meaningful interaction patterns for testing.

In this paper, we propose ARTRIP, a novel testing technique to address the above challenges. ARTRIP takes advantage of the intuition that AR app users often needs to perform multiple AR gestures in the same area to trigger more complicated behaviors. For example, after tapping the screen to place a virtual object, further interaction with the object such as rotating, resizing, color changing, or removing typically needs to happen in similar areas of the screen. Therefore, ARTRIP extends random testing from the following aspects. First, when interacting with the AR apps, ARTRIP first creates a short randomized sequence of randomized different gestures. For example, we can have a gesture sequence of Tap, Swipe, Pinch, LongTap. Second, the whole sequences of gestures will be emitted to a localized area on the screen, with randomized coordinates. We evaluated our approach on four Android AR apps, the results show that ARTRIP enhances coverage by 2.3 to 10.3 percentage points, compared with Monkey.

II. Background and Motivation

Unity Mars. Unity MARS (Mixed and Augmented Reality Studio) is an extension of the Unity engine specifically designed to simplify the development and testing of AR applications. It provides a simulation environment where

2151-0849/25/$31.00 ©2025 IEEE
DOI 10.1109/ASEW67777.2025.00068

developers can model real-world conditions such as planes, surfaces, lighting, and user movement without requiring a physical device or environment. This allows testers to run AR applications against configurable synthetic scenes, ensuring that behaviors such as object placement, occlusion, and interaction can be validated consistently. Unity MARS also supports parameterized testing, where multiple environmental variations (e.g., different room layouts or lighting conditions) can be generated automatically to evaluate app robustness. By enabling controlled and repeatable testing of AR behaviors, Unity MARS reduces the dependency on physical hardware setups and accelerates both debugging and quality assurance for AR applications.

AR Gestures. AR gestures in frameworks like ARCore and ARKit are essential interaction techniques that allow users to manipulate virtual objects naturally within real-world environments. Both frameworks commonly support gestures such as tap for selecting or placing objects, pinch for scaling, drag or pan for moving objects along detected surfaces, and rotation gestures for adjusting object orientation. These gestures are typically built on top of native touch input systems, enabling developers to integrate them seamlessly into AR applications while maintaining consistency with standard mobile interaction patterns. By combining these intuitive gestures with spatial understanding from the AR frameworks—such as plane detection and motion tracking—developers can create immersive and user-friendly AR experiences that feel both responsive and realistic.

III. Approach

In this section, we introduce **ARTRIP**, a novel random testing technique tailored for AR applications. Unlike conventional random input generation approaches, ARTRIP is designed to capture the unique characteristics of AR interactions, where users often perform multiple gestures on the same region of the screen to manipulate virtual objects. ARTRIP generates and executes randomized gesture sequences localized in specific areas of the screen to effectively trigger complex application behaviors.

A. Overview of ARTRIP

ARTRIP builds upon the intuition that user interactions in AR apps are not uniformly distributed across the screen but are instead concentrated around regions where virtual objects are placed. To model this behavior, ARTRIP generates randomized gesture sequences and executes them within a localized area. By focusing interaction clusters in one region, the technique is more likely to trigger compound gestures such as repeated zoom-ins, drag-and-hold actions, or overlapping gestures that reveal defects in AR object manipulation. The localized randomness reflects real-world usage patterns, where users often interact with virtual objects within confined regions of the screen. As shown in Figure 1, the overall workflow consists of three key steps:

1) **Gesture Sequence Construction** – generate short randomized sequences of different gesture types.
2) **Localized Area Selection** – select a random region of the screen to serve as the target interaction area.
3) **Sequence Execution** – emit the gesture sequence within the localized area with randomized parameters.

B. Gesture Sequence Construction

A central feature of ARTRIP is the generation of *short randomized gesture sequences* rather than independent gestures. This design better reflects real-world AR usage, where gestures are often performed in succession.

- **Gesture Types:** ARTRIP supports a diverse set of gestures, including *Tap, Long Tap, Swipe, Pinch, Spread, and Rotate*.
- **Sequence Length:** The sequence length is randomized within a configurable range (i.e., 2–4 gestures).
- **Diversity Enforcement:** To avoid repetitive sequences, ARTRIP ensures that gestures within a sequence are varied, and limits the number of same gesture to 2. For example, a sequence may be: Tap $\rightarrow$ Tap $\rightarrow$ Pinch $\rightarrow$ Long Tap.

This approach allows ARTRIP to simulate realistic multi-step interactions, such as placing an object, resizing it, and then rotating it.

C. Localized Area Selection

To reflect the object-centric nature of AR interactions, ARTRIP localizes the execution of gesture sequences to a specific area on the screen. Test runs are configured by specifying the region of focus (e.g., a bounding box around an AR object) and a probability distribution over interaction types. The engine then produces random yet spatially coherent sequences, which are executed on the target AR app. Failures are identified by monitoring rendering consistency, gesture recognition accuracy, and application stability.

- **Region Sampling:** A random coordinate is first selected as the *center point* of the interaction area.
- **Area Definition:** Around this center point, a small bounding box is defined to constrain all gestures in the sequence. In our implementation, a bounding box with 50 pixel as its edge is used as the localized area.
- **Multiple Areas:** Over the course of testing, ARTRIP samples different localized areas across the screen to maximize coverage.

By constraining gestures to a region, ARTRIP increases the likelihood of triggering complex object manipulations, such as object placement followed by scaling or removal.

D. Sequence Execution

Once a gesture sequence and a localized area are defined, ARTRIP executes the sequence on the AR application under test.

- **Parameter Randomization:** Each gesture is assigned randomized parameters, such as swipe direction, pinch distance, or rotation angle.
- **Sequential Emission:** The gestures are performed in order, respecting natural temporal spacing between them.

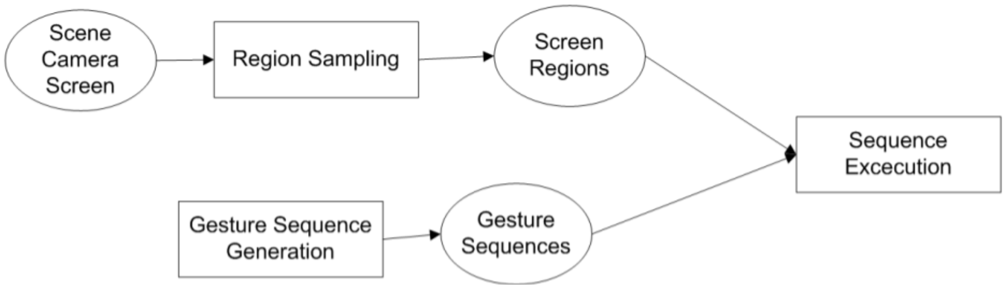

Fig. 1. Overview

- **State Observation:** After executing the sequence, ARTRIP monitors application state changes (e.g., new object placement, object transformations, or crashes).

This systematic yet randomized execution strategy enables ARTRIP to discover complex interaction paths that are often missed by conventional random testing tools.

E. Running Example

As an example, consider an AR furniture placement app. ARTRIP may randomly generate the sequence Tap $\rightarrow$ Pinch $\rightarrow$ Rotate in a localized area. Executing this sequence near the center of the screen could place a chair, resize it, and then rotate it. This mimics realistic user behavior and allows ARTRIP to expose potential defects in object manipulation logic, rendering, or state management.

IV. Evaluation

We evaluate ARTRIP on a set of representative AR applications to answer the following research question: How does ARTRIP compare to Monkey in terms of code coverage?

A. Dataset of AR Applications

We selected four open-source Android AR applications that utilize ARCore to represent different usage scenarios of AR technology. The apps cover diverse domains, including gaming, furniture placement, object measurement, and educational visualization. Table I summarizes the four applications used in our evaluation.

TABLE I
AR applications used in evaluation

Application	Domain
Hello AR	Sample App Developed by Google
AR Simulator	App Simulation App for Object Placement
AR Terrain	App for creating 3D terrain on planes
ARSnap	App for taking photos with AR objects

B. Evaluation Metrics

We use **code coverage** as the primary evaluation metric, since coverage provides a quantitative measure of how much application logic is exercised during testing. Specifically, we collected *method coverage*.

To measure coverage, we integrate each application with **JaCoCo**, a widely-used Java code coverage library. During testing, ARTRIP and Monkey generate interaction events, and JaCoCo collects coverage data by instrumenting the bytecode. After each test session, we extract coverage reports and compute the coverage statistics for comparison.

C. Experimental Setup

To set up a simulated AR testing environment in Unity MARS, we leveraged its built-in VR elements and environment templates. We created a virtual scene by placing planes, walls, and interactive objects to emulate realistic surfaces and furniture layouts. Lighting and camera parameters were adjusted to mimic typical AR device conditions, ensuring that virtual objects would behave consistently under simulated tracking. Once configured, this scene could be reused across multiple test sessions, allowing ARTRIP to execute gesture sequences in a controlled, repeatable environment.

D. Results and Comparison with Monkey

Table II reports the coverage results of ARTRIP and Monkey across the four apps. We observe that ARTRIP consistently achieves higher coverage than Monkey in all applications.

TABLE II
Method coverage comparison between Monkey and ARTRIP

Application	Monkey (%)	ARTRIP (%)
Hello AR	68.5	75.4
AR Simulator	57.6	64.1
AR Terrain	40.9	43.2
ARSnap	50.7	61.0

ARTRIP improves method coverage by 2.3 to 10.3 percentage points over Monkey, 6.5 percentage points on average. The enhancement is consistent across all subject apps. This improvement comes from ARTRIP's ability to generate localized multi-step gesture sequences that more closely resemble real user interactions. In particular, apps that rely more on object manipulation (e.g., Hello AR and AR Snap) show the largest improvements, demonstrating the effectiveness of ARTRIP in exercising complex AR-specific behaviors.

V. Discussion

ARTRIP introduces several strengths compared to existing random testing techniques such as Monkey. Despite its advantages, ARTRIP also has several limitations. First, ARTRIP's random sequence generation may still produce many unproductive interactions, leading to inefficiencies in exploring application state space. Second, while localized gestures are effective for object manipulation, they may underexplore global UI components (e.g., navigation menus or settings) that are less tied to object-centric interactions. Third, ARTRIP relies on simulated scenes for consistent testing environments, which may have limitations in mimicing real-world scenes. Finally, ARTRIP currently measures effectiveness primarily in terms of code coverage; additional metrics such as fault detection rate or user-perceived quality are needed to fully understand its impact.

VI. Related Works

Testing augmented reality (AR) applications presents unique challenges due to their reliance on real-time environmental interactions and device-specific sensors. Existing approaches often focus on traditional testing methods, such as manual testing and unit testing, which may not adequately address the complexities of AR environments.

VR-ReST [2] introduces requirements-driven automated functional testing for VR using scene-graph concepts—ideas that transfer to AR interaction testing. A notable advancement is the development of frameworks like ARCHIE [3], which collects user feedback and system state data to identify and debug issues in AR applications in real-world settings. ARCHIE++ [4] extends ARCHIE with a cloud-enabled pipeline that scales in-the-wild AR testing and telemetry collection across diverse devices and environments. VRTest [5] describes a extensible framework for automatic testing of VR scenes, and VRGuide [6] extended VRTest with an exploration strategy supported by cut theory. XR testing mapping study [7] provides the first systematic mapping of software testing for XR, surveying techniques, tools, datasets, and open challenges. Test Automation for AR [8] a process model and case study demonstrating partial automation of AR application testing, including interaction variants. Model-Based Testing for AR [9] explores model-based testing notions tailored to AR app behaviors and scene graphs. Youkai [10], [11] describes a cross-platform framework to script tests for VR/AR apps across engines/devices. On the automation of AR testing, PredART [12] proposes a hybrid CNN+MLP model to automatically predict test oracles for AR object placement realism, evaluated on Unity MARS scenes. VOPA [13] extends PredART by further considering real-world videos in the testing scenarios. In contrast, ARTRIP introduces a novel approach by generating randomized gesture sequences localized to specific screen areas, effectively simulating realistic user interactions within AR applications. This method enhances the detection of complex behaviors and interactions that traditional testing methods might overlook.

On the study and analysis of XR software, Rodriguez and Wang [14] performed the first emprical study of open source VR apps and summarized their characteristics. Zhang et al. [15] studied the major privacy leak risks in AR apps in Android. Li et al. [16] conducts the first empirical study of WebXR bugs (368 issues/33 projects), building a taxonomy that informs AR test design and oracles. Nusrat et al. [17] studied the performance optimization commits in VR applications to understand the major performance issues in VR software and how developers tackle them. Molina et al. [18] proposed a novel approach to identify dependencies between meta file [19] objects and source code in VR applications. Zhang et al. [20] performed a user study to find out how people value realisticness and privacy [21] [22] in Metaverse environments and applications.

VII. Conclusion

In this paper, we presented ARTRIP, a novel random testing technique designed specifically for AR applications. Unlike conventional random testing tools such as Monkey, ARTRIP models realistic user behavior by generating short randomized gesture sequences localized in the same screen region. This design enables ARTRIP to more effectively trigger complex AR interactions, such as object placement, manipulation, and deletion. We implemented ARTRIP and evaluated it on four representative AR applications across different domains. The experimental results showed that ARTRIP consistently achieved higher code coverage than Monkey. These results demonstrate that ARTRIP can better capture the unique interaction patterns of AR apps and expose behaviors that would otherwise remain untested. In the future, we plan to extend ARTRIP with adaptive strategies that dynamically adjust gesture generation based on runtime feedback, further improving its effectiveness in testing the rapidly growing ecosystem of AR applications.

VIII. Acknowledgment

This work is in part supported by NSF Grant 2418093.

References

[1] Google, "Testing augmented reality applications," 2021. [Online]. Available: https://www.pixelqa.com/blog/post/testing-augmented-reality-applications

[2] E. C. Souza, A. R. R. Araújo, J. Silva, and R. Souza, "An automated functional testing approach for virtual reality applications," *Software Testing, Verification and Reliability*, vol. 28, no. 8, p. e1690, 2018.

[3] S. M. Lehman, S. Elezovikj, H. Ling, and C. C. Tan, "Archie: A user-focused framework for testing augmented reality applications in the wild," in *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2020.

[4] ——, "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.

[5] X. Wang, "Vrtest: an extensible framework for automatic testing of virtual reality scenes," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, 2022, pp. 232–236.

[6] X. Wang, T. Rafi, and N. Meng, "Vrguide: Efficient testing of virtual reality scenes via dynamic cut coverage," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 951–962.

[7] R. Gu, J. M. Rojas, and D. Shin, “Software testing for extended reality applications: A systematic mapping study,” *Automated Software Engineering*, 2025, early access.

[8] D. Minor, S. Baum, U. Eisenecker *et al.*, “Test automation for augmented reality applications: Development process model and case study,” *i-com*, 2023.

[9] A. Porfirio, A. R. R. Araújo, G. Gava, J. Silva, E. Souza, and R. Souza, “An approach for model based testing of augmented reality applications,” in *RCIS 2022 Workshops (CEUR-WS.org, vol. 3201)*, 2022. [Online]. Available: https://ceur-ws.org/Vol-3201/

[10] T. Figueira and A. Gil, “Youkai: A cross-platform framework for testing vr/ar apps,” in *HCI International 2022 – Late Breaking Papers: Interacting with Computers*. Springer, 2022, pp. 3–12.

[11] S. A. Andrade, F. L. S. Nunes, and M. E. Delamaro, “Automated test of vr applications,” in *Advances in Computer Graphics*, ser. LNCS. Springer, 2020, vol. 12221.

[12] T. Rafi, X. Zhang, and X. Wang, “Predart: Towards automatic oracle prediction of object placements in augmented reality testing,” in *37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2022.

[13] 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 *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024.

[14] I. Rodriguez and X. Wang, “An empirical study of open source virtual reality software projects,” in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2017, pp. 474–475.

[15] X. Zhang, R. Slavin, X. Wang, and J. Niu, “Privacy assurance for android augmented reality apps,” in *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2019, pp. 114–1141.

[16] 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 *31st IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 172–183.

[17] F. Nusrat, F. Hassan, H. Zhong, and X. Wang, “How developers optimize virtual reality applications: A study of optimization commits in open source unity projects,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 473–485.

[18] J. Molina, X. Qin, and X. Wang, “Automatic extraction of code dependency in virtual reality software,” in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 2021, pp. 381–385.

[19] F. Hassan and X. Wang, “Mining readme files to support automatic building of java projects in software repositories,” in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 277–279.

[20] X. Zhang, T. Rafi, Y. Guan, S. Li, and M. R. Lyu, “Understanding the privacy-realisticness dilemma of the metaverse,” *Automated Software Engineering*, vol. 32, no. 2, p. 53, 2025.

[21] X. Zhang, X. Wang, R. Slavin, and J. Niu, “Condysta: Context-aware dynamic supplement to static taint analysis,” in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 796–812.

[22] X. Zhang, J. Heaps, R. Slavin, J. Niu, T. Breaux, and X. Wang, “Daisy: Dynamic-analysis-induced source discovery for sensitive data,” *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 4, pp. 1–34, 2023.