Towards Automatic Oracle Prediction for AR Testing: Assessing Virtual Object Placement Quality under Real-World Scenes

Xiaoyi Yang
xy3371@rit.edu
Rochester Institute of Technology
Rochester, New York, USA

Yuxing Wang
yw2009@rit.edu
Rochester Institute of Technology
Rochester, New York, USA

Tahmid Rafi
md.tahmidulislam.rafi@utsa.edu
University of Texas at San Antonio
San Antonio, Texas, USA

Dongfang Liu
dongfang.liu@rit.edu
Rochester Institute of Technology
Rochester, New York, USA

Xiaoyin Wang
xiaoyin.wang@utsa.edu
University of Texas at San Antonio
San Antonio, Texas, USA

Xueling Zhang
xueling.zhang@rit.edu
Rochester Institute of Technology
Rochester, New York, USA

Abstract

Augmented Reality (AR) technology opens up exciting possibilities in various fields, such as education, work guidance, shopping, communication, and gaming. However, users often encounter usability and user experience issues in current AR apps, often due to the imprecise placement of virtual objects. Detecting these inaccuracies is crucial for AR app testing, but automating the process is challenging due to its reliance on human perception and validation. This paper introduces VOPA (**V**irtual **O**bject **P**lacement **A**ssessment), a novel approach that automatically identifies imprecise virtual object placements in real-world AR apps. VOPA involves instrumenting real-world AR apps to collect screenshots representing various object placement scenarios and their corresponding metadata under real-world scenes. The collected data are then labeled through crowdsourcing and used to train a hybrid neural network that identifies object placement errors. VOPA aims to enhance AR app testing by automating the assessment of virtual object placement quality and detecting imprecise instances.

In our evaluation of a test set of 304 screenshots, VOPA achieved an accuracy of 99.34%, precision of 96.92% and recall of 100%. Furthermore, VOPA successfully identified 38 real-world object placement errors, including instances where objects were hovering between two surfaces or appearing embedded in the wall.

CCS Concepts

• **Software and its engineering** → **Software testing and debugging**.

Keywords

Augmented Reality, Automated Test Oracle, Machine Learning

ACM Reference Format:
Xiaoyi Yang, Yuxing Wang, Tahmid Rafi, Dongfang Liu, Xiaoyin Wang, and Xueling Zhang. 2024. 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 (ISSTA '24), September 16–20, 2024, Vienna, Austria.* ACM, Vienna, Austria, 13 pages. https://doi.org/10.1145/3650212.3680315

ISSTA '24, September 16–20, 2024, Vienna, Austria

ACM ISBN 979-8-4007-0612-7/24/09
https://doi.org/10.1145/3650212.3680315

1 Introduction

Augmented Reality (AR) is an emerging technology that overlays digital content onto users' view of the real world in real time, creating an interactive and enriched experience. Its applications span across different industrial domains, including gaming, education, healthcare, retail, and entertainment [46, 50, 56, 57, 60, 64]. The AR market is expected to grow rapidly in the coming years [4], driven by the increasing accessibility of affordable AR-enabled devices such as smartphones with AR platforms [9] [3] and smart glasses such as Hololens [5] and Oculus [6]. Additionally, the increasing popularity of AR applications across different industries also contributes to this growth. Notable examples of popular AR applications include Pokémon Go [25], Snapchat filters [27], IKEA Place [22], and Google Translate [19]. These successful AR applications showcase the potential of AR to transform the way we interact with the world around us.

Compared with traditional graphical user interface (GUI) apps, AR apps have a more significant and seamless impact on users' daily lives. Bugs in AR apps may lead to more severe consequences as they directly affect users' interaction with the physical world. For instance, the misbehavior of an AR driving navigation app may cause immediate damage to the physical environment surrounding the users [67]. As shown in Figure 1, an AR navigation app is displaying a virtual object of a bright green column to indicate the direction for turning right. However, the virtual object partially blocks the view of a moving car in front of the driver (highlight with a red oval), which could potentially influence the driver's decision-making and lead to accidents.

Proper placement of virtual objects requires techniques like Simultaneous Localization and Mapping (SLAM) [43, 55] or object recognition [35, 54] to analyze the user's environment. The detection of surfaces and walls, as well as the precise positioning and rotation of virtual objects, are critical factors that significantly impact the quality of virtual object placement.

However, bugs in AR applications and platforms, as well as limitations in computer vision and sensing techniques, can lead to miscalculations in the positioning of real-world objects, resulting

Figure 1: An example of improper virtual object placement

Figure 2: An example of a floating and partially visible virtual object from a real-world AR app

in imprecise placement of virtual objects. These errors may cause virtual objects to appear misaligned, floating in the air, or even partially visible, as illustrated in Figure 2.

Early adopters of mobile AR applications have experienced such issues. For instance, Amazon's ARView application encountered problems with placing virtual objects on shiny physical surfaces and accurately sizing models to fit the physical environment [13]. Similarly, IKEA's AR application faced issues where virtual objects would float or drift away from the original anchor point, particularly in low-light conditions [15]. In this paper, we refer to these issues as *imprecise placement*.

Assessing and assuring the quality of an AR application is more complex compared to traditional software products [68]. AR development frameworks such as Unity [8], ARKit [3], and ARCore [9], provide testing frameworks to assist developers in testing their AR apps using real or virtual devices and scenes. However, a major challenge in AR testing is its heavy reliance on human intervention. While recording and playback[10] have reduced human effort to some extent by enabling the reuse of scene videos, allowing users to avoid physically visiting a location for each AR feature test, testers still need to carefully observe test execution or analyze recorded videos and screenshots to assess object placement precision. In addition, due to the subjective nature of human perception, different testers may have different opinions on the acceptability of imprecise object placement. To ensure an unbiased and comprehensive evaluation, multiple testers are often required for each test case, leading to a considerable amount of human effort in AR testing.

To reduce human effort and potential bias in the testing process, an existing work PredART [65] utilize the virtual reality (VR) test scene and an scene controller from Unity MARS Test Framework to create virtual object placement. The screenshot taken from the test scenes were then used to train a model to rate object placement. While VR scenes are often used for initial testing of AR apps due to their high flexibility, testing of AR apps in real-world scenes is still necessary because the simulation in virtual scenes may not fully reflect real-world scenarios. In the real world, the environment can be much more chaotic and unpredictable and virtual reality scenes may not capture all environmental factors such as lighting conditions, shadows, and reflections that may significantly impact AR app performance. In addition, PredART was designed for testing AR apps in VR scenes (e.g., Unity Mars), relying on VR-scene features that might not always be accessible in real-world scenes. For example, PredART utilizes features such as the camera's location and the distance between the placed objects and surfaces. These data can be easily accessed in a virtual scene but are often impossible to measure in real-world scenes. As a result, PredART cannot be directly applied to the testing of AR apps in real-world scenes. Instead, an approach that utilizes information available in real-world scenes is needed.

In this paper, we develop a novel framework, VOPA (**V**irtual **O**bject **P**lacement **A**ssessment), to explore the feasibility of automatically predicting the quality of object placement in real-world AR scenes. The prediction results can be used as automatic oracle in AR software testing and raise warnings to human testers only when a potential imprecise placement is found. VOPA involves four major steps. First, we instrument AR apps to extract runtime metadata and introduce mutations for various imprecise object placements such as objects floating above surfaces, ensuring a balanced dataset. Second, we run these apps in various designed test scenarios, collecting metadata and screenshots for both normal and mutated placements. Third, we use Amazon Mechanical Turk [30] to label the realisticness of each screenshot. Finally, we train a classification model using the labeled data to predict realism levels and identify imprecise placements in unseen real-world AR app screenshots.

To evaluate the performance of VOPA in assessing the quality of object placement, we applied it to unseen test screenshots under various evaluation configurations, yielding the following results:

- In random evaluation, the dataset were randomly split into training (90%) and test (10%) sets. VOPA achieved a precision of 96.92% and recall of 100% on the testing set.
- In cross-object evaluation, the dataset were split into training and test sets based on object types. VOPA achieved a precision of 95.65% and recall of 91.67% on the testing set.
- In cross-app evaluation, the dataset were split into training and test sets based on apps. The evaluation was repeated multiple times to make sure each app served as a test app. VOPA achieved an average precision of 92.13% and recall of 83.86% on the test set.
- In cross-scene evaluation, the dataset were split into training and test sets based on scenarios. The evaluation was repeated multiple times to make sure each scenarios served as the test

scenario. VOPA achieved an average precision of 89.38% and recall of 65.80% on the test set.
- VOPA outperforms ResNet [49], the state-of-the-art neural network architecture for image recognition task, by 9.87%
- In our test set, VOPA successfully identified 38 out of total 42 real-world placement errors, such as objects hovering between two surfaces or appearing embedded in the wall.

The evaluation results demonstrate that VOPA can precisely identify placement errors in real-world AR apps. In summary, this paper makes the following major contributions:

- A novel approach to instrument real-world AR apps to generate a large number of screenshots that represent various object-placement scenarios under real-wold scenes.
- A labeled public dataset [28] of object placement screenshots from real-world scenes, which may serve as a valuable resource for research in the field of AR application analysis.
- A novel classifier trained on screenshots and metadata from diverse real-world scenarios. It effectively assesses the quality of object placement and identifies imprecise instances in real-world AR apps.

2 Background

2.1 ARCore

Google's ARCore platform enables AR experiences on Android devices, providing tools for developers to create apps that place virtual objects in real environments. ARCore uses advanced computer vision and motion tracking technologies to understand the physical environment around the user and anchor virtual objects accurately in the real world [39]. ARCore uses SLAM[72] for motion tracking, detecting *Feature Point* in camera images to estimate device position and movement. The inertial measurements are then combined with the feature points to determine the camera's coordinates and orientation, which is defined as *Pose* (position and rotation) in ARCore. Pose in ARCore refers to the 6-degree-of-freedom position and orientation of an object in 3D space.

In ARCore, a *plane* is a detected flat, horizontal surface in the real world. ARCore identifies planes by tracking multiple feature points on surfaces and their boundaries. This environmental understanding allows users to place virtual objects using *anchors* - fixed reference points in the real world. The app renders 3D models based on the anchor's pose, maintaining a stable relationship with the plane. This ensures virtual objects appear realistically aligned with detected surfaces in the camera image[41].

2.2 Sceneform

ARCore is a powerful fundamental tool for creating AR experiences on mobile devices. However, it does not provide developers with functions for controlling the AR scene and processing the 3D models of virtual objects. Developers often need to utilize additional OpenGL-based libraries for rendering virtual content, such as Sceneform. Sceneform[7] is an Android app development framework that utilizes ARCore and OpenGL to simplify the development process of AR apps. While ARCore focuses on motion tracking and environmental understanding, Sceneform aims to facilitate the creation of AR scenes and the rendering of 3D models. Sceneform extends the functionality of ARCore and abstracts away the complexities of OpenGL programming in the context of Android development.

Unlike the *Anchor* setting in ARCore which attaches to the plane, Sceneform introduces the concept of *Node* and its subclass, *AnchorNode*. By setting the node as a child of the AR scene or another node, Sceneform establishes a stable relationship between different objects. Sceneform can be considered an extension of ARCore since it is built on top of it. Sceneform inherits the Pose value directly from ARCore without conversion and allows it to track the ARCore's Pose starting from the AR scene.

2.3 CNN and ResNet

Convolutional Neural Network (CNN) is a type of artificial neural network that is widely used for image recognition tasks. CNNs consist of multiple layers that can be seen as a regularized version of the fully connected Multi-Layer Perceptrons (MLPs). The high number of parameters in MLPs, where each neuron in a layer is connected to every neuron in the next layer, can lead to overfitting due to repeated training on the noise in the training data. To address this issue, CNNs implement regularization, such as the Dropout Layer, which randomly drops some units to zero during training. CNNs have been demonstrated to be effective in image classification and regression tasks.

The Residual Network (ResNet) [49] was first presented in 2015, offering a groundbreaking architecture designed to address the issues of vanishing and exploding gradients in deep learning models. This innovative approach incorporated a technique known as *skip connections* within the network. By bypassing the training of certain layers and establishing a direct link to the output, *skip connections* enable the network to focus on learning residual mappings rather than individual layers. As a result, ResNet has become the most popular neural network in the field of computer vision. In this work, ResNet is employed as the baseline method.

3 Approach

Our approach has four main steps, as shown in Figure 3. First, we instrument the AR apps to extract relevant metadata and insert mutations in some object placements to create imprecise placements. Second, we run the instrumented AR apps on our test device, performing both mutated and unmutated object placements in real-world environments while collecting runtime metadata and capturing screenshots. Next, we use Amazon Mechanical Turk to label the realisticness level of each screenshot. Finally, the labeled screenshots and their corresponding metadata will be used to train a model for predicting the realisticness level and identifying instances of imprecise object placement in real-world AR apps.

3.1 Metadata List

To identify the factors that could impact the realisticness of virtual object placement in an AR app, we examined ARCore's documentation and identified which metadata can be acquired while the AR app runs and virtual objects are placed within real-world scenes.
Object Translation and Rotation: Object position and orientation in ARCore's coordinate system, determining if the object is precisely placed. This data can be obtained by `Pose.getTranslation()` and `Pose.getRotationQuaternion()` from ARCore [38].

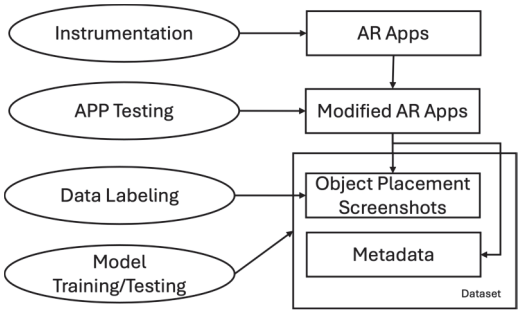
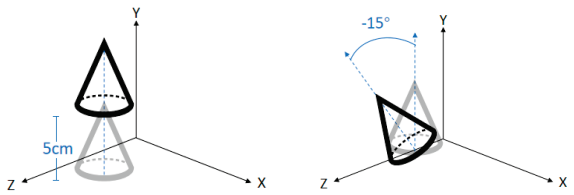


Figure 3: Approach Overview

Camera Translation and Rotation: Camera position and orientation in ARCore's coordinate system, determining the viewing angle, shaping users' perceptions of the displayed image, and influencing the realisticness of virtual objects. This data can be acquired using the method `Frame.getCamera()` from ARCore [37].
Distance: The Euclidean distance between the camera and the virtual object in ARCore's coordinate system. This distance affects the user's perception and can be calculated using the Euclidean distance formula with the translation of the camera and the object.
Light Condition: Light condition is another significant factor affecting the visual quality of real-world environment [18]. We measure the illuminance using the app "Light Meter LM-3000" [24].

3.2 Object Placement Mutation

In AR apps, instances of incorrect object placements are less frequent than those of correct placements. If we only capture screenshots during normal object placement, our dataset would become imbalanced and lack examples of imprecise placements. To address this, we introduce mutations that modify the position and rotation of virtual objects, allowing us to generate a dataset that includes various imprecise placements for training the model.



(a) Mutation on position, the object moved in the direction of Y-Axis by five centimeters.

(b) Mutation on rotation, the object was rotated clockwise around Z-Axis by 15 degrees.

Figure 4: Mutation Examples

ARCore uses a 3D coordinate system to describe the position and rotation of a virtual object through three axes: X, Y, and Z. The X-axis goes from left to right, the Y-axis goes from bottom to top, and the Z-axis goes towards the viewer, perpendicular to the screen. We introduce 'mutations' to the object's position and rotation, such as sinking or floating along the Y-axis, or tilting around the X or Z-axis. Figure 4 shows visual examples of an object floating and rotating. We do not consider moving the object along the X and Z axes or rotating it around the Y-axis. This is because such changes will not affect the realisticness of the object. The object would still be fully placed on the surface but turned to a different side. Table 1 shows the types of errors we implanted in the object placement and the corresponding offsets.

Table 1: Errors implanted in object placement

Floating or Sinking	-5cm	5cm	10cm	15cm
Rotation around X-axis	-30°	-15°	15°	30°
Rotation around Z-axis	-30°	-15°	15°	30°

3.3 App Instrumentation

The instrumentation involves decoding the app's APK (Android Package) files into smali code[1] using APKTool[70] and then inserting our code to the smali files. To collect metadata during testing, we instrument the relevant APIs mentioned in Section 3.1 to log the coordinates for object position and the quaternion for object rotation. To introduce mutation on the object position and rotation, it is important to understand how ARCore determines the accurate coordinates to place the object into the desired location. By investigating ARCore's documentation, we identify two important APIs related to virtual object placement. The method `hitTest()` [20] is used to determine the correct placement of the object in the scene when the user touches the screen to initiate an object placement. It returns a `HitResult`[21] object containing a `Pose`[40]. The method `createAnchor()` then [29] can place an anchor on the plane based on this `Pose`. To instrument these APIs, we locate the invocation of methods `hitTest()` and `createAnchor()` in the apps' smali code and insert our code for to introduce the desired offset into the object's coordinates. After instrumenting the apps, we rebuild the samli code into APK files for testing.

3.4 Data Collection at Run Time

We run the modified AR apps on our test device, placing virtual objects in various real-world scenes to collect metadata and screenshots. This involves three major phases:
Real-world scenes setup. The realisticness of placing a virtual object relies heavily on the detected surface where the object is positioned. Therefore, we consider different types of surfaces in our test scenes. Specifically, we set up a basic living room environment with two types of surface scenarios: single surface and multiple surfaces. We also simulate different light conditions, represent three common real-life illuminance levels [69]: 1) Weak (below 100 lux); 2) Intermediate (100-200 lux); 3) Strong (above 200 lux).

(a) Without carpet

(b) With a carpet

Figure 5: Single-surface scenarios

Object placement on a single surface. For object placement on a single surface, we set up two floor scenarios: one with a carpet and the other without. (See Figure 5 for the two different floor settings.) To place virtual objects on a single surface, we start by setting the center spot where the object will be positioned. We then move

[1]Assembler for the dex format used by Dalvik

the device to different distances and viewing angles around the center spot to capture various visuals of the virtual object. Figure 6 illustrates the visual representation of the mobile device and virtual object layout in the single surface scenario. We consider eight different directions, spaced 45 degrees apart, to cover all sides of the virtual object. Additionally, we set the mobile device's height above the ground at three basic levels: 50, 75, and 100 centimeters. For the distance between the device and the object, rather than using fixed distances, we dynamically adjust the distance based on the object's size and the screen display, allowing for close, medium, and distant placements. This adaptive approach is necessary because objects of different sizes may require varying distances to appropriately appear on the screen. For example, small virtual objects like an apple can be difficult to see if they are too far from the camera. Conversely, large objects like a virtual sofa may not fit on the screen if the camera is positioned too close.

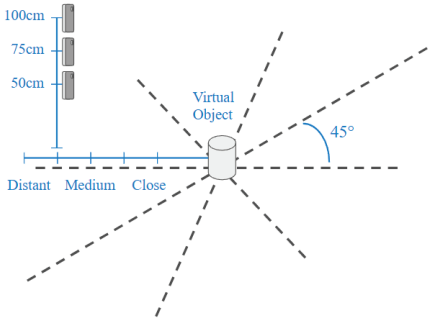
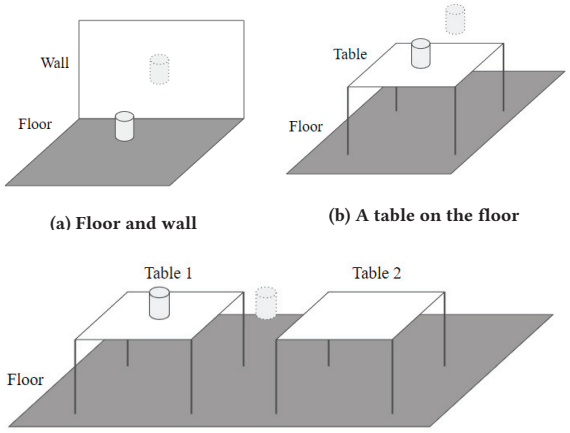


Figure 6: Single surface scenario

Object placement involving multiple surfaces. For object placement involving multiple surfaces, we consider three specific scenarios to emulate real-world user experiences with AR apps. (See Figure 7 for a visual representation of the three different types of surface scenarios.) The first scenario (Figure 7a) involves a wall and a floor to assess whether the app considers both horizontal and vertical surfaces for object placement. For instance, when placing a sofa on the floor against the wall, it should not appear embedded in the wall or floating above the floor. The second scenario (Figure 7b) involves a single table on the floor. We use it to test whether the object can be placed on the table correctly. Additionally, we examine how the app handles situations where users try to place the object outside the table's edge, verifying if the surface boundary is accurately detected. The third scenario (Figure 7c) involves two tables side by side with a gap between them. We assess the AR app's ability to detect both tables accurately and determine if the object is correctly placed. Additionally, we explore if the app allows placing the object between the two tables when the gap is smaller than the object's size. We also check whether the app allows placing the object in a larger gap, causing the object to float in the air.

Screenshots and metadata collection. We capture screenshots of each virtual object placement in the test scenes and record the corresponding metadata. We use the app Screenshot Touch[26] to take the screenshot at the moment when the virtual object is placed. Figure 8 shows four screenshots from our dataset, describing different virtual objects placed under different scenes with or without mutations. During testing, we utilize the Android Debug Bridge (ADB)

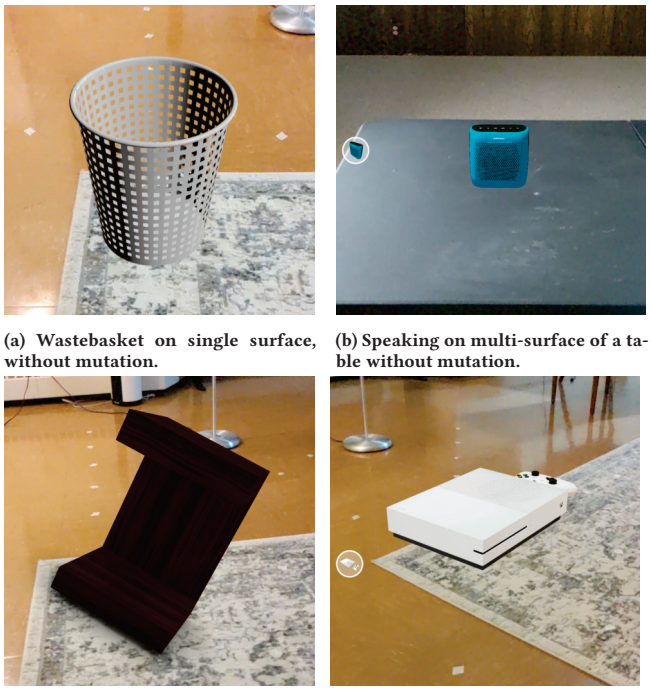


(a) Floor and wall

(b) A table on the floor

(c) Two tables on the floor

Figure 7: Multi-surface scenarios: accurate placement (solid object) vs. imprecise placement (dashed object)



(a) Wastebasket on single surface, without mutation.

(b) Speaking on multi-surface of a table without mutation.

(c) Night stand on single surface, with mutation on rotation around z-axis by -30 degrees.

(d) Console system on single surface, with mutation of floating 15cm above the floor.

Figure 8: Exemplar screenshots in dataset

[14] to store the log file, which contains the metadata discussed in Section 3.1. An example of logged metadata is shown in Figure 9. To match the extracted metadata with the corresponding screenshot, we rely on the associated timestamps as a reference. The metadata with the closest timestamp to the screenshot's timestamp is paired with it. An example of the matched data is illustrated in Figure 10. The first element represents the screenshot's filename, and the subsequent elements are the corresponding metadata values.

```
04-04 15:00:21.263  5999  5999 I System.out: VOPA_Sceneform_Translation: [x=0.14776681, y=-0.9594432, z=-3.359169]
04-04 15:00:21.263  5999  5999 I System.out: VOPA_Sceneform_Rotation: [x=4.265079E-17, y=-0.025083011,
z=-1.0701466E-18, w=0.99968535]
04-04 15:00:21.263  5999  5999 I System.out: VOPA_Sceneform_Distance: 3.560304
04-04 15:00:21.263  5999  5999 I System.out: VALID
04-04 15:00:21.263  5999  5999 I System.out: VOPA_EnvironmentalHdrAmbientSphericalHarmonics: [0.43873906,
0.42230427, 0.32862025, 0.03969823, 0.05527267,
0.06397745, -0.037636556, -0.055696487, -0.03693772, -0.031330064, -0.032977797, -0.035283126, 0.006046541,
0.01938016, 0.013080776, 0.41167116, 0.41901797,
0.36780155, -0.0078899115, -0.03352903, -0.060524367, -0.02677574, -0.029909037, -0.008625224, -0.17854609, -0.184
95284, -0.20203444]
04-04 15:00:21.263  5999  5999 I System.out: VOPA_EnvironmentalHdrMainLightDirection: [-0.41023418,
0.6576849, -0.6317899]
04-04 15:00:21.263  5999  5999 I System.out: VOPA_EnvironmentalHdrMainLightIntensity: [0.39242443, 0.45289382,
0.38250157]
04-04 15:00:21.263  5999  5999 I System.out: VOPA_CameraRotation: [-0.0849067, -0.14348356, -0.707101, 0.6871764]
04-04 15:00:21.263  5999  5999 I System.out: VOPA_CameraTranslation: [-0.049164467, 0.06637402, 0.044458557]
```

Figure 9: An example of metadata in the log file

3.5 Data Labeling

Our data labeling is conducted through Amazon Mechanical Turk [31], which is the most popular crowdsourcing platform. We submit all screenshots as HITs (Human Intelligence Task) to request workers to rate the realisticness of the virtual object in each screenshot based on the following instructions we provide:

- Ignore any icons, frames, and pop-ups that appear on the image. Focus on the virtual object in the AR environment.
- Rate based on position, rotation, distance, and shadow.
- Do NOT rate based on the color or texture of the model.
- The rating should be subjective, based on the first impression of the image, without any professional knowledge.

```
Screenshot_2023-03-29_211958.jpg |
[4.2641736E-17, -0.032459974, -1.3848793E-18, 0.99947304]
| [0.09680539, -1.3694235, -1.2173684] | 1.6386143 |
[-0.15395819, -0.2127009, -0.68388754, 0.6807004] |
[-0.0071497895, -0.40975007, 0.10674538] | 3 | 0 | 3
```

Figure 10: A example of matched screenshot and metadata

For each HIT, workers were asked to rate the screenshots on a scale from -2 to 2, indicating the level of realisticness from low to high (see Figure 11). A rate of -2 indicates very imprecise placement, -1 represents somewhat imprecise placement, 1 denotes somewhat precise placement, and 2 indicates very precise placement. We intentionally designed negative numbers for imprecise placement and positive numbers for precise placement to make the rating more intuitive. The option of 0 is excluded to avoid neutral ratings and ensure meaningful assessment. We also provide example screenshots of the four rates for reference, with an emphasis that workers should rate based on their on their own judgment.

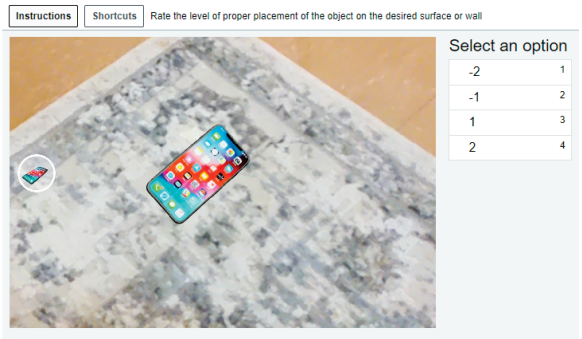


Figure 11: A HIT example

The labeling process is subjective as different workers may feel different on the same screenshot. To obtain representative labels, each screenshot is assigned to five distinct workers through five HITs. The ratings from these five workers are then averaged to form a single label for each screenshot, which will be used for model training. To calculate the average rating, the original rating scale ranging from -2 to 2 is first converted into a scale from 1 to 4. After computing the average of the five ratings, the final label for a screenshot is assigned based on the closest number within the 1 to 4 scale to the average rating. To ensure the selection of qualified and responsible workers, we applied a filter that considered workers with an approval rate of over 90% based on existing guidelines [59].

3.6 Hybrid Image Classification

The labeled dataset will be used to build a classification model that predicts the realisticness level of object placement. Our classifier adopts a multimodal model, utilizing images and metadata as inputs. Figure 12 shows the model's architecture.

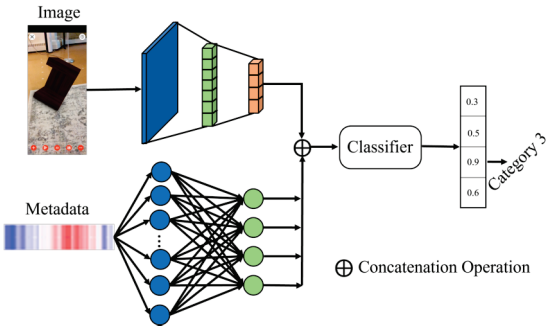


Figure 12: Structure of the Hybrid Model

The hybrid model uses two separate networks to independently extract features from images and metadata. Numeric features are processed by Multilayer perceptron (MLP)[66] to output four-dimensional features. At the same time, image features go through the Convolutional Neural Network (CNN)[62] to generate four-dimensional features. These features are then concatenated and used as input for the classifier. The classifier predicts category probabilities based on the concatenated features, and the category with the highest probability is the predicted result. The CNN module has one convolution layer with a 3×3 convolution kernel and two dense layers with ReLU activation. The MLP module consists of two dense layers. The classifier includes two dense layers, with the output of the last layer passed through a Softmax activation. During training, the model weights in the neural network are randomly initialized. The model is optimized using *Categorical Cross Entropy (CCE)* and an Adam optimizer [51] with learning decay. CCE measures the difference between the probability distributions of one-hot encoded CNN computed class labels and ground truths. As shown in Equation.1, n denotes the total number of categories, y denotes the one-hot vector representing the ground truth where $y_i = 1$ if the item belongs to *i-th* category and $y_i = 0$ if otherwise, and p_i denotes the predicted probability that this item belongs to *i-th* category[47].

$$CCE = -\sum_{i=1}^{n} y_i \log(p_i) \tag{1}$$

The model is trained with the initial learning rate, and then the learning rate gradually decays according to the set learning decay until a local minimum is reached. To ensure a robust model and

prevent data splitting issues, we employ 5-fold cross-validation during training.

4 Evaluation

Our evaluation aims to answer following research questions:

- RQ1: Can VOPA effectively predict the realisticness level of virtual object placements and identify imprecise ones?
- RQ2: Does VOPA outperform the state-of-the-art deep image learning technique?
- RQ3: What kind of imprecise object placement can VOPA identify in real-world AR apps?

4.1 Dataset

We compiled our list of AR apps based on categories such as "AR Apps" [11] and "AR Games" [12] provided by Google Play Store. Additionally, we manually added some well-known AR apps and conducted searches on Google Play Store using keywords like "AR" or "Augmented Reality." It's important to note that Google Play Store's search relies on keywords within the Title, Description, and Promo Text, along with considerations such as compatibility[16, 17]. Therefore, search results obtained solely using the keyword "AR" can be ambiguous. The generated app list included numerous applications containing the keyword "AR" that did not necessarily include AR feature, such as the app "Indeed Job Se**ar**ch", "G**ar**denscapes" or "Calend**ar**". To address this, we excluded non-AR apps from our dataset by checking if the app code contained any popular AR SDKs listed in Table 2. We identified 227 AR apps, with ARCore being the most prevalent at 214 apps, representing 94.25% of the total.

Table 2: The occurrence and percentage of each AR SDK for Android in collected apps. Apps might have multiple SDKs imported. SDKs with a count of zero have been omitted

AR SDK	Occurrence	Percentage
ARCore	214	94.27%
Vuforia	34	14.98%
Wikitude	7	3.08%
EasyAR	6	2.64%
Microsoft Mixed Reality	5	2.2%
OpenCV	1	0.44%
ARZone	1	0.44%
Total	**227**	

For the 214 ARCore apps , we searched for invocations of methods `hitTest()` and `createAnchor()`, or the occurrence of the `Pose` object (refer to Section 3.3). We found 83 apps that directly invoked these APIs for object placement. We then executed each of these 83 apps on our test device and examined the decompiled smali code to determine if they could be used for our study. We selected only the apps that met the following criteria for our final app list.
① *Detection of Physical Environment*: Apps should have features that detect the physical environment for precise virtual object placement. Apps lacking this feature, such as an app displaying a virtual 3D solar system in front of the camera without interaction with the physical environment, were excluded (22 apps).
② *Functionality*: Apps should be fully functional. Apps with crash or freezing issues and were excluded.(12 apps).
③ *Accessibility*: Apps should be freely accessible without requiring a business account or purchase of accompanying products for the AR experience. Apps failing to meeting this criterion, such as some interior design apps that require user's business accounts for login, were excluded (20 apps).
④ *Compliant with experimental configuration*: The app should provide an AR experience that aligns with our experimental configuration. Apps designed for large outdoor objects placement, such as cars or buildings, were excluded(12 apps).
⑤ *Other Exclusions*: A few apps were excluded for reasons such as heavy obfuscation making the code unreadable (4 apps), APK decoding errors (1 app), and lack of AR features (1 app).

This criteria resulted in 11 apps. Please note that due to expensive 3D models and intellectual property, high-quality AR apps involving object placement are often not free or requires business account, which limits the number of apps can be used in our dataset. However, this does not undermine the applicability of our approach.

Among the 214 ARcore apps, except the 83 apps that directly invoked the related ARCore APIs for object placement, a large portion (131 apps) used other implementations for object placement. We examined the 131 apps and found that 109 apps were developed using ARCore in Unity. Unity converts code to C++ using IL2CPP framework when generating an APK, the C++ code are compiled to be .so files and packed into the APK file[23]. Reversing binary files and insert our code for instrumentation is challenge and our approach doesn't apply. To include Unity apps in our dataset, we chose to search for open-source apps on GitHub using the keywords 'unity' and 'ARCore.' Each app was tested on our device to determine if it met our criteria mentioned above. Through this process, we collected 10 functioning Unity apps that utilized ARCore.

In total, we collected 21 AR apps as the dataset for this study. The full list of these AR apps can be found on our website[28].We instrumented these 21 apps (see Section 3.3) and then executed them on our test device to collect runtime metadata and screenshots for each object placement (see Section 3.4). This generated a dataset of 3043 screenshots and their corresponding metadata, depicting the placements of 104 different virtual objects.

4.2 Evaluation under Different Configurations

The goal of VOPA is to raise warning to app developers when imprecise placement is detected. So we consider imprecise placement as positive case in our evaluation. Note that when app developers or testers use VOPA, they could configure different "threshold" to determine what is considered imprecise based on their acceptable level of imprecise object placement. For instance, they may treat predicted labels "very imprecise," "somewhat imprecise," and "somewhat precise" as positive cases, being more stringent about object placement quality. Alternatively, they may choose to be less strict on the quality, considering only "very imprecise" as positive and the rest of the labels as negative. In this study, we take a middle-ground approach, considering predicted labels "very imprecise" and "somewhat imprecise" as positive cases, and "very precise" and "somewhat precise" as negative cases.

To answer RQ1, we conducted evaluations of VOPA under five different configurations as described below, with results presented in the first six rows of Table 3. Following the metrics for multi-class

classification[48], we calculated the accuracy, precision, recall, and F1 score for each configuration.

①*Random*: We selected 10% of the dataset as the test set, while the remaining 90% served as train set. The first row of Table 3 shows the results of this evaluation, with an accuracy of 99.34%, precision of 96.92%, recall of 100%, and F1 score of 98.44%.

②*Cross-object*: Among the 104 different virtual objects in our dataset, we randomly selected the data of 94 objects as the train set and let VOPA test on the remaining 10 objects. We intended to investigate whether VOPA can successfully identify imprecise placements of unseen objects. The second row of Table 3 shows the results of this evaluation, with an accuracy of 96.95%, precision of 95.65%, recall of 91.67%, and F1 score of 93.62%. We further inspected the evaluation metrics of each object in the test set to see if different objects exhibited varying performance. Table 4 presents the results of each object. The "Mutation" column indicates whether mutations were implanted in the object placement, while the "Size" column denotes the object size. Our observations suggest that the model faced challenges in predicting the imprecise placement of smaller objects with rotation mutation.

③*Cross-app*: To evaluate VOPA's ability to identify imprecise placement in unseen apps, we strategically divided our apps into training and test apps. The test set comprised data from test apps, while the training set included data from the training apps. Among the 21 AR apps in our study, four were from the same company and we treated them as one app to avoid potential bias due to similarities in apps from the same company. The number of data entries varies across different apps because of variations in the quantity of available 3D models in different apps. Some apps have over 400 data entries while others have less than 100. To address potential challenges arising from imbalances or overfitting, we divided the apps into seven groups, each containing more than 10% of the total data volume (304 entries). The cross-app evaluation was repeated seven times to ensure each group served as the test set. The third row of Table 3 shows the results of this evaluation, with an average accuracy of 95.21%, precision of 92.13%, recall of 83.86%, and F1 score of 85.44%. Since we use each group of data as a test set separately, here we take the average of all the experiments combined as the result to be displayed. Note that we had a limited number of apps in our dataset (21 apps) because many apps failed to meet our criteria: some were non-functional due to frequent crashes or unresponsiveness; some were paid apps; some apps' AR features did not interact with the physical environment, such as displaying virtual 3D solar system models. However, with a relatively small number of training apps, the evaluation results indicate that our tool can precisely identify bugs in object placement in unseen apps.

④*Cross-scene*: As discussed in Section 3.4, we include single and multiple surfaces in our test scenes. Specifically, we covered five different scenes in our work: 1) A floor in room 1; 2) A carpet on the floor in room 2; 3) One table on the floor in room 1; 4) Two tables on the floor in room 1; 5) A wall and a floor in room 1. To avoid potential biases in evaluation caused by the same scenes appearing repeatedly in both the train set and the test set, we conducted the cross-scene evaluation. During the cross-scene evaluation, we repeated the process five times, designating one scene as the test set each time while the remaining four scenes served as the training set. It is important to note that due to the repetitive nature of the cross-scene configuration, all values presented in this paragraph and in Table 3 represent the average across the five iterations. As shown in Table 3 (row "VOPA_Cross-Scene"), this evaluation achieved an average accuracy of 91.17%, precision of 89.38%, recall of 65.80%, and F1 score of 72.12%, indicating the robustness of our tool across varied scenes.

⑤*Cross-mutation*: As discussed in Section 3.2, we implemented mutation in our dataset to generate imprecise placement samples. To evaluate whether our tool can successfully identify unseen mutations (imprecise placement), we conducted cross-mutation experiments. For position mutation (refer to Section 3.2), we split our dataset into five groups based on the position offset: -5cm, 5cm, 10cm, 15cm, and no-mutation. The evaluation was repeated four times, with each group except the no-mutation serving as the test set and the remaining 4 groups as the training set. The values presented in Table 3 on row "cross-mutation(P)" represent the mean across these four iterations, achieving an accuracy of 99.13%, precision of 96.43%, recall of 94.88%, and F1 score of 95.63%. Similarly, for rotation mutation, we divided our dataset into nine groups (refer to Section 3.2), including the eight different types of rotation degrees and no-mutation. The evaluation was carried out eight times, each time using one group (except no-mutation) as the test set and the remaining 8 groups as the training set. The results, as shown in Table 3 on row "cross-mutation(R)", demonstrate an average accuracy of 97.29%, precision of 99.32%, recall of 90.00%, and F1 score of 94.18% across these eight iterations.

According to the result, random evaluation have the best performance as expected because the model might have already seen the test apps or objects during training. The cross-object outperforms cross-app, as it tests on unseen objects, and the model might have seen the test app during training. While in cross-app configuration, the model is tested on unseen apps plus unseen objects, as different AR apps have different virtual objects. Cross-scene has a relatively lower accuracy, for reasons similar to the Cross-app experiment results. During the data collection process, different objects are used in different scenes. Therefore, for the model, both the scenes and objects in the test set data are unseen, which affects the model's performance. The cross-mutation demonstrates excellent performance, with accuracy levels nearly equivalent to the results obtained from VOPA_Random configuration. This indicates that the model possesses a strong ability to handle unseen mutations.

4.3 Comparison between VOPA and ResNet

To answer the RQ2, we compared VOPA with ResNet, the-state-of-art image recognition model. This comparison was conducted in the same random configuration as VOPA was evaluated. As shown in the fourth row of Table 3, ResNet has an accuracy of 89.47%, precision of 100%, recall of 49.21%, F1 score of 65.96%. While ResNet performed well, VOPA outperformed it by successfully identifying some imprecise placements that ResNet had missed. The parameters of both VOPA and ResNet can be found in our website[28].

We chose to compare with ResNet because it can be directly applied to our problem with feature concatenation. While more advanced image classification models like CoatNet [36] and Model Agnostic Meta-Learning (MAML) [73] could be potential candidates

Table 3: Evaluation result of VOPA.

	Accuracy	Precision	Recall	F1-Score	TP	FP	TN	FN
VOPA_Random	99.34%	96.92%	100%	98.44%	63	2	239	0
VOPA_Cross-Object	96.95%	95.65%	91.67%	93.62%	66	3	220	6
VOPA_Cross-App	95.21%	92.13%	83.86%	85.44%	71	8	345	9
VOPA_Cross-Scene	91.17%	89.38%	65.80%	72.12%	73	21	473	40
VOPA_Cross-Mutation(P)	99.13%	96.43%	94.88%	95.63%	10	1	113	1
VOPA_Cross-Mutation(R)	97.29%	99.32%	90.00%	94.18%	46	1	124	4
ResNet	89.47%	100%	49.21%	65.96%	31	0	241	32
Random_Regression	52.63%	49.46%	64.08%	55.83%	91	93	69	51

Table 4: Performance of each object in the test set

Object	Accuracy	Precision	Recall	F1-Score	TP	FP	TN	FN	Mutation	Size
Phone	88.46%	100%	72.73%	84.21%	8	0	15	3	Rotation	Small
Wooden Drawer	92.00%	75.00%	100%	85.71%	6	2	17	0	Rotation	Large
Goalpost	95.31%	100%	76.92%	86.96%	10	0	51	3	Rotation	Medium
Toy Castle	98.48%	93.33%	100%	96.55%	14	1	51	0	Rotation	Medium
Dinnerware	100%	100%	100%	100%	1	0	2	0	None	Small
Statue	100%	100%	100%	100%	2	0	25	0	Position	Medium
Planter	100%	100%	100%	100%	5	0	21	0	Position	Medium
Burger	100%	100%	100%	100%	1	0	6	0	None	Medium
Steel drawer	100%	100%	100%	100%	18	0	29	0	Rotation	Large
Sofa	100%	100%	100%	100%	1	0	3	0	None	Large

for our task, they may require adaptations such as transfer learning. These models are highly parameterized and heavily rely on large datasets. However, the amount of data in our case (3043 samples) is insufficient to effectively train these Transformer-based models. So we did not include them in our evaluation.

4.4 Comparison between Classification and Regression

Despite VOPA is designed for a classification task, we wanted to explore whether considering it as a regression task would yield better performance. To do this, we trained a regression model using our dataset in the random configuration. We first calculated the mean of the label values without rounding and then normalized it to a value between 0 and 1. To predict the normalized score of each screenshot, we used the sigmoid activation function. For evaluating the performance of this approach, we used mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE) as the evaluation metrics and loss function for the regression model. In the following equations, N denotes the total number of data, $\hat{x}_i$ denotes the predicted score of *i-th* data, and x_i denotes the ground truth of *i-th* data.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{x}_i - x_i| \tag{2}$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{x}_i - x_i)^2 \tag{3}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{x}_i - x_i)^2} \tag{4}$$

To provide a more intuitive representation of the accuracy of the regression model, we devised a method to classify the predictions of the regression experiment, making it easier to compare with the results of the classification experiment. Correspondingly, we divided the prediction results of the regression experiment into four categories 1, 2, 3, and 4. In the equation, x denotes both the prediction and the ground truth of the data. The value of x lays in the range between 0 and 1 due to the normalization.

$$category(x) = \begin{cases} 1, & 0 \le x \le 0.25 \\ 2, & 0.25 < x \le 0.5 \\ 3, & 0.5 < x \le 0.75 \\ 4, & 0.75 < x \le 1 \end{cases} \tag{5}$$

As shown in the fifth row of Table 3, the regression model got an accuracy of 52.63% , precision of 49.46%, recall of 64.08%, and F1 score of 55.83%. VOPA significantly outperforms this regression model indicating that classification task is more suitable than regression task for assessing object placement in real-world scenes.

4.5 Real-world Imprecise Object Placement

Recall that when creating our dataset, we intentionally implanted mutations to some object placements to generate imprecise cases to ensure a more balanced dataset (see Section 3.2 Object Placement Mutation). Thus, our dataset contains both real and mutated placements. In this study, we refer to real imprecise placements as **real bugs**. The primary goal of VOPA is to identify real bugs. To assess VOPA's capability to identify real bugs and answer RQ3, we examined the evaluation results about real bugs. Table 5 shows the number of real bugs detected by VOPA under three different evaluation configurations. In the random evaluation, the test set contains seven real bugs, all of which were successfully identified by VOPA. In the cross-object evaluation, VOPA detected all three real bugs contained in the test set. In the cross-app evaluation, the evaluation was repeated seven times to ensure each app (or apps from the same company) was selected as the test set. VOPA was able to identify 38 real bugs in all repetition (42 real bugs in total).

Table 5: Real-world bugs identified by VOPA

	#real bug
VOPA_Random	7 out of 7
VOPA_Cross-Object	3 out of 3
VOPA_Cross-App	38 out of 42

Upon further investigation of the identified real bugs, we confirmed the existence of various types of placement errors in real-world AR apps. These include instances where objects appear to hover between two surfaces or are embedded within walls. Figure 13 listed some example screenshots of these real bugs.

(a) The blender is floating between tables

(b) The basket is floating over the front edge

(c) The toy is floating over the edge

(d) The piano is hanging on the wall

Figure 13: Examples of real bugs detected by VOPA

4.6 Threats to Validity

The major internal threat to the validity of our evaluation lies in the labeling process of our dataset, where human workers from MTurk were involved. This introduces the possibility of errors or biases in the labeling. To mitigate the threat, each screenshot was independently labeled by five different workers, and their individual labels were then averaged to obtain a final representative label. This helps to reduce the impact of any individual worker's potential errors. An external threat to the validity of our evaluation is the limited number of apps included in our dataset, which is because two main factors. First, several apps failed to meet our requirements: some were non-functional due to frequent crashes or unresponsiveness; some were not freely accessible or required a business account for access; some apps' AR feature did not interact with the physical environment, such as displaying virtual 3D solar system models.Second, due to instrumentation limitations with Unity apps discussed in Section 4.1, we searched for open-source Unity apps on GitHub to add to our app collection. However, most AR open-source projects on GitHub are intended for functional testing rather than commercial use, the number of Unity apps suitable for our study was limited and their functionalities were basic. To mitigate this threat, we plan to explore the usage of ARCore in a broader context and include additional apps that utilize different frameworks.

Another external threat arises from the limited coverage of real-world test scenes. Due to the inherent challenges in encompassing every potential use case, it's difficult to comprehensively cover all possible scenarios in real-world. As a result, our work primarily focuses on virtual object placement on surfaces, including both single and multi-surface scenarios. Thus, our setting scenarios may not accurately capture the complexity and nuances of real-world situations. To mitigate this limitation and enhance the diversity of our dataset, we included 104 distinct objects in our dataset and considered various factors that can potentially affect the realisticness of the virtual object placement, such as different object and camera translation and rotation, surface scenarios, and light conditions. Despite the relatively simplified test scenes, the results demonstrate that our tool can accurately identify real-world bugs, such as objects floating on surfaces or embedded in walls. This indicates the potential for identifying more bugs in more complex scenes. In future, we plan to expand our scope to include diverse and more complicated settings. Furthermore, Google ARCore and Apple ARKit recently introduced playback videos, enabling developers from various apps to reuse recordings of the same scene for testing their apps. In scenarios where scene recordings are reused, we can train our model on these commonly reused scenes, allowing developers to directly utilize our model when they reuse those scenes.

5 Discussion

5.1 Limitations of VOPA

VOPA has two major limitations. First, it operates at the screenshot level, not accounting for the dynamic nature of real user experiences. Some anomalies may be challenging to identify in static screenshots due to factors like distance and viewing angle, but they might be more apparent in a dynamic video. In this study, we collect screenshots as individual frames extracted from dynamic videos and analyze these frames to determine the quality of virtual object placements. Our future work will consider detecting imprecise placement based on videos. Second, we focus on AR apps that are developed with ARCore. While ARCore is one of the prominent AR app development frameworks, other competitive frameworks, such as ARKit[3], Wikitude[2], and Vuforia[1] are also widely used. Different frameworks may lead to variations in the performance of AR apps, as their underlying configuration of AR scenes might differ. Therefore, to achieve more universal testing strategies, we plan to adapt and evaluate our approach across various AR frameworks.

5.2 Differences between VOPA and PredART

In practice, AR apps are often tested in both virtual scenes and real-world scenes because the former can provide cost-efficient and configurable environments to exercise the AR app in a large variety of scenarios, while the latter can provide experience closer to the real-world usage scenario of the AR apps, so they may reveal some bugs that are not easily revealed in the simulated scenarios.

PredART[65], an existing approach for assessing the realisticness of object placement within virtual scenes, relies on data accessible through the testing framework (i.e., Unity Mars). This data includes variables such as placement gaps and viewing angles. Despite the flexibility, VR testing scenes can never perfectly simulate real-world scenes because all the objects and lighting conditions in virtual scenes are based on precise mathematical calculation, and thus they lack the randomness and noises which are unavoidable in real world scenes. PredART can not be applied to real-world scenes, creating an evident need for complementary testing approach like VOPA, specifically designed to evaluate real-world AR scenarios. VOPA offers novel techniques that extend its usability to real-world environments. First, it instruments object placement APIs in Google ARCore to extract run-time meta data when executing AR apps in real-world scenes. Second, it mutates object placement to generate imprecise placement cases in the dataset. Additionally, this study created a dataset of object placement screenshots from real-world scenes, which serves as a valuable resource for future research in the field of AR application analysis.

6 Related Work

6.1 VR and AR App Testing

VR and AR testing is an emerging research area, and we are aware of only a few of research efforts in this area. VRTest[71] is a test framework for VR software that extracts metadata from VR scenes at runtime, and automatically controls the user's camera to explore the VR scene and interact with virtual objects. PredART[65] uses virtual scene information to predict whether object placement are realistic. We specifically discussed the difference between VOPA and PredART in Section 5.2.

6.2 Automated Testing

Automated testing, crucial for software development, enhances app stability, functionality, usability, and compatibility, ensuring optimal user experience[52]. Mao et al.[58] developed a tool, Sapienz, to effectively identify defects in Android apps, using multi-objective search-based testing to automatically explore and optimize test sequences, minimising length, while simultaneously maximising coverage and fault revelation. Zhong et al.[76] proposed an iterative Android automated testing approach aimed at improving automated testing tools' ability to understand complex interactive app widget frameworks. Wuji[75] is a software testing technique that leverages evolutionary algorithms and reinforcement learning to support the automatic testing of games. Compare to these works focus on regular apps, our work focus on AR app testing, which is an entirely different type of domain, which presents different challenges compared with trnditional apps. Different with the aforementioned works that primarily focus on testing transitional apps, our research aims to enhance AR app testing. AR testing is a new and totally different domain, presenting unique challenges compared to traditional apps.

6.3 Test Oracle Generation

The test oracle generation is essential part in automatic testing, the quality of test oracle directly impacts the performance of the testing technique since it determines if the test case is passed or failed. Two surveys[32, 63] in 2014 points out the difficulty in constructing the test oracle and proposes the research direction to solution is oracle reuse. Therefore, the amount of work on the oracle generation is very limited. The classic approach to oracle generation relies on designing a comprehensive algorithm. JSEFT[61] is a technique to automatically generate test cases for JavaScript applications, it implements a mutation-based process[45] for oracle generation that reduces the number of assertions automatically generated and targets critical and error-prone portions of the application. Donaldson et al. propose to use metamorphic testing[34] to identify defects in graphic shader compilers via semantics-preserving transformations. Zaeem et al.[74] presented a framework to automatically generate test case and insert oracles for testing user-interaction features including zooming, scrolling, pressing button in mobile apps.

The increasing use of machine learning in research has led to its application in automated testing and test oracle generation, which presenting new challenges. A survey[44] in 2021 suggested that Machine learning-based test oracle generation confronts open challenges including data collection, limitation of initial training data, and complexity of machine learning technique. Some studies have been conducted to propose solutions in various fields to address those problems. TOGA[42] is a novel framework uses unified transformer-based neural approach to infer both exception and assertion test oracles from a given test prefix and unit context. Langdon et al. [53] argued that deep learning can be used to predict partial test oracles for mutation testing. Chen et al. proposed the framework GLIB [33] that used CNN-Based deep learning model to detect UI gliches exist in graphically-rich apps, and the model was trained using screenshots collected from apps.

7 Conclusion

This paper presents VOPA, a novel approach for automatically identifying imprecise virtual object placements in real-world AR apps. VOPA involves instrumenting real-world AR apps to capture screenshot of object placement and their corresponding metadata during runtime. The collected screenshots are then labeled by Mechanical Turk. The labeled screenshots, along with their corresponding metadata, are used to train a machine learning model for predicting the level of realisticness of the object placement. The evaluation results demonstrate that VOPA achieves high precision and recall in predicting imprecise object placements, outperforming the state-of-the-art image learning model ResNet. Also, VOPA successfully identifies real object placement errors in real-world AR apps. In the future, we plan to ① expand our dataset to include more real-world scenes, ② investigate the feasibility of universal testing strategies that can cover more AR app development frameworks beyond ARCore, ③ develop an approach that use video to assess object placement and identify imprecise instances.

Acknowledgments

This work is partially supported by the National Science Foundation under Awards 2348468, 1736209, 1846467, 2007718, and 2221843.

References

[1] 2015. Vuforia. https://www.ptc.com/en/products/vuforia.
[2] 2015. Wikitude. https://www.wikitude.com/.
[3] 2018. Apple ARKit. https://developer.apple.com/augmented-reality/.

[4] 2018. MarketsandMarkets. https://www.marketsandmarkets.com/Market-Reports/augmented-reality-virtual-reality-market-1185.html.
[5] 2018. Microsoft Hololens. https://www.microsoft.com/en-us/hololens.
[6] 2018. Oculus Rift. https://www.oculus.com/.
[7] 2020. Sceneform. Retrieved April 12, 2023 from https://developers.google.com/sceneform/develop
[8] 2020. Unity MARS. https://unity.com/products/unity-mars.
[9] 2021. Google AR Core. https://developers.google.com/ar.
[10] 2021. Recording and playback introduction. https://developers.google.com/ar/develop/recording-and-playback.
[11] 2022. AR Apps. Retrieved August 1, 2022 from http://bit.ly/GooglePlay_ArApps
[12] 2022. AR Games. Retrieved August 1, 2022 from http://bit.ly/GooglePlay_ArGames
[13] 2023. Amazon's AR Tool is a useful way to see products before you buy them. https://www.dailydot.com/debug/amazon-ar-view-app/.
[14] 2023. Android Debug Bridge. https://developer.android.com/tools/adb.
[15] 2023. Avoid The Dreaded Ikea Fight With This New App. https://www.fastcompany.com/90143908/avoid-killing-your-relationship-at-ikea-with-this-new-app.
[16] 2023. Filters on Google Play. https://developer.android.com/google/play/filters Google Play Filter.
[17] 2023. Get discovered on Google Play search. https://support.google.com/googleplay/android-developer/answer/4448378?hl=en Google Play Search.
[18] 2023. Get the lighting right. https://developers.google.com/ar/develop/lighting-estimation.
[19] 2023. Google Translate,. https://play.google.com/store/apps/details?id=com.google.android.apps.translate&hl=en_US&gl=US.
[20] 2023. Hit-tests place virtual objects in the real world. https://developers.google.com/ar/develop/hit-test.
[21] 2023. HitResult. https://developers.google.com/ar/reference/java/com/google/ar/core/HitResult.
[22] 2023. IKEA Place,. https://www.ikea.com/au/en/customer-service/mobile-apps/say-hej-to-ikea-place-pub1f8af050.
[23] 2023. IL2CPP Overview. https://docs.unity3d.com/Manual/IL2CPP.html IL2CPP.
[24] 2023. Light Meter LM-3000. https://apps.apple.com/us/app/light-meter-lm-3000/id1554264761.
[25] 2023. Pokémon Go,. https://pokemongolive.com/?hl=en.
[26] 2023. Screenshot touch. https://play.google.com/store/apps/details?id=com.mdiwebma.screenshot&hl=en_US&gl=US.
[27] 2023. Snapchat Lenses,. https://lens.snapchat.com/?locale=en-US.
[28] 2023. VOPA: Virtual Object Placement Assessment. https://sites.google.com/view/vopa-for-artesting/home VOPA.
[29] 2023. Working with Anchors. https://developers.google.com/ar/develop/anchors.
[30] Amazon. 2005. Amazon Mechanical Turk. https://www.mturk.com.
[31] Amazon. 2005. Amazon Mechanical Turk. Retrieved April 12, 2023 from https://www.mturk.com/
[32] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The Oracle Problem in Software Testing: A Survey. IEEE Transactions on Software Engineering 41, 5 (2015), 507–525. https://doi.org/10.1109/TSE.2014.2372785
[33] Ke Chen, Yufei Li, Yingfeng Chen, Changjie Fan, Zhipeng Hu, and Wei Yang. 2021. Glib: towards automated test oracle for graphically-rich applicationsIn . 1093–1104.
[34] Tsong Y Chen, Shing C Cheung, and Shiu Ming Yiu. 2020. Metamorphic testing: a new approach for generating next test cases. arXiv preprint arXiv:2002.12543 (2020).
[35] Yiming Cui, Liqi Yan, Zhiwen Cao, and Dongfang Liu. 2021. Tf-blender: Temporal feature blender for video object detectionIn . 8138–8147.
[36] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. 2021. Coatnet: Marrying convolution and attention for all data sizes. Advances in Neural Information Processing Systems 34 (2021), 3965–3977.
[37] Google Developers. 2021. Frame. Retrieved December 13, 2023 from https://developers.google.com/ar/reference/java/com/google/ar/core/Frame
[38] Google Developers. 2021. Pose. Retrieved April 13, 2023 from https://developers.google.com/ar/reference/java/com/google/ar/core/Pose
[39] Google Developers. 2022. Overview of ARCore and supported development environments. Retrieved April 11, 2023 from https://developers.google.com/ar/develop
[40] Google Developers. 2022. Perform hit-tests in your Android app. Retrieved April 11, 2023 from https://developers.google.com/ar/develop/java/hit-test/developer-guide
[41] Google Developers. 2023. Fundamental concepts. Retrieved April 12, 2023 from https://developers.google.com/ar/develop/fundamentals
[42] Elizabeth Dinella, Gabriel Ryan, Todd Mytkowicz, and Shuvendu K Lahiri. 2022. TOGA: a neural method for test oracle generationIn . 2130–2141.
[43] Hugh Durrant-Whyte and Tim Bailey. 2006. Simultaneous localization and mapping: part I. IEEE robotics & automation magazine 13, 2 (2006), 99–110.
[44] Afonso Fontes and Gregory Gay. 2021. Using machine learning to generate test oracles: a systematic literature reviewIn . 1–10.
[45] Gordon Fraser and Andreas Zeller. 2010. Mutation-driven generation of unit tests and oraclesIn . 147–158.
[46] Henry Fuchs, Mark A Livingston, Ramesh Raskar, Kurtis Keller, Jessica R Crawford, Paul Rademacher, Samuel H Drake, Anthony A Meyer, et al. 1998. Augmented reality visualization for laparoscopic surgeryIn . Springer, 934–943.
[47] Nidhi Gowdra, Roopak Sinha, Stephen MacDonell, and WeiQi Yan. 2021. Maximum Categorical Cross Entropy (MCCE): A noise-robust alternative loss function to mitigate racial bias in Convolutional Neural Networks (CNNs) by reducing overfitting. (2021).
[48] Margherita Grandini, Enrico Bagli, and Giorgio Visani. 2020. Metrics for multi-class classification: an overview. arXiv preprint arXiv:2008.05756 (2020).
[49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. https://doi.org/10.48550/ARXIV.1512.03385
[50] Hirokazu Kato and Mark Billinghurst. 1999. Marker tracking and hmd calibration for a video-based augmented reality conferencing systemIn . IEEE, 85–94.
[51] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).
[52] Pingfan Kong, Li Li, Jun Gao, Kui Liu, Tegawendé F Bissyandé, and Jacques Klein. 2018. Automated testing of android apps: A systematic literature review. IEEE Transactions on Reliability 68, 1 (2018), 45–66.
[53] William B Langdon, Shin Yoo, and Mark Harman. 2017. Inferring automatic test oraclesIn . IEEE, 5–6.
[54] Dongfang Liu, Yiming Cui, Wenbo Tan, and Yingjie Chen. 2021. Sg-net: Spatial granularity network for one-stage video instance segmentationIn . 9816–9825.
[55] Dongfang Liu, Yiming Cui, Liqi Yan, Christos Mousas, Baijian Yang, and Yingjie Chen. 2021. Densernet: Weakly supervised visual localization using multi-scale feature aggregationIn , Vol. 35. 6101–6109.
[56] Lutz Lorenz, Philipp Kerschbaum, and Josef Schumann. 2014. Designing take over scenarios for automated driving: How does augmented reality support the driver to get back into the loop?In , Vol. 58. SAGE Publications Sage CA: Los Angeles, CA, 1681–1685.
[57] Michael R Lyu, Irwin King, TT Wong, Edward Yau, and PW Chan. 2005. Arcade: Augmented reality computing arena for digital entertainmentIn . IEEE, 1–9.
[58] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: Multi-objective automated testing for android applicationsIn . 94–105.
[59] Winter Mason and Sidharth Suri. 2011. How to use mechanical turk for cognitive science researchIn , Vol. 33.
[60] Zeljko Medenica, Andrew L Kun, Tim Paek, and Oskar Palinko. 2011. Augmented reality vs. street views: a driving simulator study comparing two emerging navigation aidsIn . 265–274.
[61] Shabnam Mirshokraie, Ali Mesbah, and Karthik Pattabiraman. 2015. JSeft: Automated JavaScript unit test generationIn . IEEE, 1–10.
[62] Keiron O'Shea and Ryan Nash. 2015. An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458 (2015).
[63] Mauro Pezzè and Cheng Zhang. 2014. Chapter One - Automated Test Oracles: A Survey. Advances in Computers, Vol. 95. Elsevier, 1–48.
[64] Iulian Radu. 2012. Why should my students use AR? A comparative review of the educational impacts of augmented-realityIn . IEEE, 313–314.
[65] Tahmid Rafi, Xueling Zhang, and Xiaoyin Wang. 2022. PredART: Towards Automatic Oracle Prediction of Object Placements in Augmented Reality TestingIn . 1–13.
[66] Hassan Ramchoun, Youssef Ghanou, Mohamed Ettaouil, and Mohammed Amine Janati Idrissi. 2016. Multilayer perceptron: Architecture optimization and training. (2016).
[67] Franziska Roesner, Tadayoshi Kohno, and David Molnar. 2014. Security and privacy for augmented reality systems. Commun. ACM 57, 4 (2014), 88–96.
[68] Jim Scheibmeir and Yashwant K. Malaiya. 2019. Quality Model for Testing Augmented Reality ApplicationsIn . 0219–0226. https://doi.org/10.1109/UEMCON47517.2019.8992974
[69] Engineering ToolBox. 2004. Illuminance - Recommended Light Level. Retrieved April 15, 2023 from https://www.engineeringtoolbox.com/light-level-rooms-d_708.html
[70] Connor Tumbleson and Ryszard WiÅŻniewski. 2017. Apktool-A tool for reverse engineering 3rd party, closed, binary Android apps.
[71] Xiaoyin Wang. 2022. VRTest: an extensible framework for automatic testing of virtual reality scenesIn . 232–236.
[72] Wikipedia. 2023. Simultaneous localization and mapping. Retrieved April 12, 2023 from https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping
[73] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference timeIn . PMLR, 23965–23998.
[74] Razieh Nokhbeh Zaeem, Mukul R Prasad, and Sarfraz Khurshid. 2014. Automated generation of oracles for testing user-interaction features of mobile appsIn . IEEE, 183–192.

[75] Yan Zheng, Xiaofei Xie, Ting Su, Lei Ma, Jianye Hao, Zhaopeng Meng, Yang Liu, Ruimin Shen, Yingfeng Chen, and Changjie Fan. 2019. Wuji: Automatic online combat game testing using evolutionary deep reinforcement learningIn . IEEE, 772–784.

[76] Yi Zhong, Mengyu Shi, Youran Xu, Chunrong Fang, and Zhenyu Chen. 2023. Iterative Android automated testing. Frontiers of Computer Science 17, 5 (2023), 175212.

Received 2024-04-12; accepted 2024-07-03