# PredART: Towards Automatic Oracle Prediction of Object Placements in Augmented Reality Testing

Tahmid Rafi
md.tahmidulislam.rafi@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

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

## ABSTRACT

While the emerging Augmented Reality (AR) technique allows a lot of new application opportunities, from education and communication to gaming, current augmented apps often have complaints about their usability and/or user experience due to placement errors of virtual objects. Therefore, identifying noticeable placement errors is an important goal in the testing of AR apps. However, placement errors can only be perceived by human beings and may need to be confirmed by multiple users, making automatic testing very challenging. In this paper, we propose PredART, a novel approach to predict human ratings of virtual object placements that can be used as test oracles in automated AR testing. PredART is based on automatic screenshot sampling, crowd sourcing, and a hybrid neural network for image regression. The evaluation on a test set of 480 screenshots shows that our approach can achieve an accuracy of 85.0% and a mean absolute error, mean squared error, and root mean squared error of 0.047, 0.008, and 0.091, respectively.

## KEYWORDS

Augmented Reality, Virtual Objects, Placement Error

**ACM Reference Format:**
Tahmid Rafi, Xueling Zhang, and Xiaoyin Wang. 2022. PredART: Towards Automatic Oracle Prediction of Object Placements in Augmented Reality Testing. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3551349.3561160

## 1 INTRODUCTION

Augmented Reality (AR) is an emerging technology that allows software users to view real-world scenes and objects with computer-enhanced perceptual information [10]. There have been tens of thousands of AR apps available for various AR devices (e.g., Android phones with ARCore support [7], iPhones with ARKit support [2], Hololens [3], Oculus [4]), and they have found numerous application scenarios such as computer-aided manual operations (e.g., driving support [35], medical treatments [19]), navigation [41], education [49], remote conference [27], and entertainment [38]. Compared with traditional GUI apps, AR applications can affect users' daily lives in a deep and seamless manner, so bugs in AR apps may lead to more severe consequences [51]. For example, the misbehavior of an AR driving navigation app may cause immediate damage to the physical world surrounding the users.

In AR apps, to enable experiences that make virtual content appear to be attached to real-world objects (e.g., surfaces such as tables, floors, walls, or human faces), virtual objects are typically placed at certain points (called *anchors*) [22] relative to the real-world objects (called *trackables*) [22] identified by AR devices. However, bugs in the applications and platforms, as well as limitations of existing computer vision and sensoring techniques, may cause miscalculation of *trackables'* positions and imperfect placement of virtual objects. These will cause human users to feel the added virtual object as being floating or misplaced. In certain cases, such placement errors may even occlude real-world objects or other virtual objects, leading to dysfunction or more severe consequences. To make sure AR systems and apps meet users' expectations, precision of virtual object placement is an important measurement of AR applications' usability during AR software testing [29]. In this paper, we refer to the distance from a virtual object's placed location at run-time to its desired location (on the trackables) as its *Placement Gap*.

AR software testing is very difficult in real-world scenarios because it is typically infeasible to construct enough physical scenes (e.g., various indoor rooms and outdoor settings under different lighting conditions) to exercise AR apps. Therefore, frameworks often provide virtual reality scenes for testing purposes. For example, Figure 1 and Figure 2 show two virtual scenes provided by GoogleAR [7] and Unity [6], respectively, for testing purposes. Within virtual reality scenes, although a test script can automatically move the camera to view different parts of the scene and place virtual objects at different locations, the testing process is still often manual because a human tester needs to either watch the test execution or watch videos or screenshots recorded during test execution to decide whether an object placement is noticeably imprecise. Furthermore, as shown later in our dataset (section 4), different users may have different feelings about the same object placement gap, especially when the wrong placement is not far from the proper position. Therefore, multiple human testers may be required to acquire an unbiased and thorough understanding of object placement accuracy in a test execution. Note that the distance between the object placement position and the ground truth position (which is often not accessible in physical scenes but accessible in virtual scenes) may not be a good oracle, because various factors such as the viewing angle, distance, and the object size/character may have influence on whether an object placement looks real or unrealistic.

*ASE '22, October 10–14, 2022, Rochester, MI, USA*
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3561160

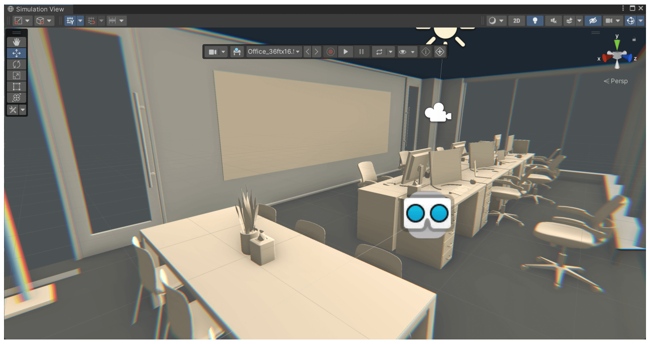


**Figure 1: Virtual AR Testing scene provided by Unity MARS**

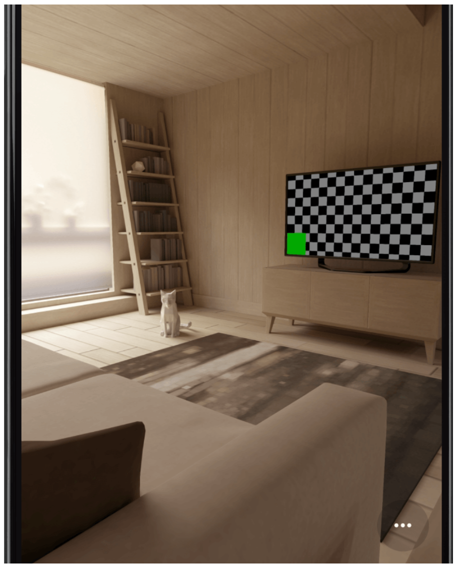


**Figure 2: Virtual scene provided with Google ARCore**

To reduce human effort and potential bias in the testing of object placements, in this paper, we develop a novel technique, PredART, to explore the feasibility of predicting whether the placement of an object is noticeably imprecise according to human users. The prediction results can be used for automatic assertions in AR software testing and raise warnings to human testers only when a potential imprecise placement is found. Our research process follows the general practice of design science [24] [59]. In particular, we first analyzed the problem domain to find out that the judgment of whether object placements are realistic relies on cognition from multiple human users. Furthermore, we found that it is possible to automatically move cameras in test scenes to generate a large number of screenshots for the training and testing of deep learning models. Based on these observations, we designed our deep learning-based solution in three major steps. First, we use a simple virtual test scene and an automatic scene controller to create screenshots of virtual objects with different placement gaps from various viewing angles and distances. Second, we send the screenshots to a crowd-sourcing website (e.g., Amazon Mechanical Turk [9]) to ask for multiple normal users to label each screenshot and calculate a reality score based on the labels. Third, we train a deep image regression model with the labeled screenshots together with their recorded camera meta information and use the model for test oracle prediction. The major challenge in the third step is that the image forms a much larger feature vector than the meta information, so off-the-shelf deep regression tools train a model dominated by the image features. To address the challenge, we combine a convoluted neural network (CNN) and a multiple-layer perceptron (MLP) to construct a hybrid deep learning model, so that the two parts of the features can be better balanced.

The goal of our solution is to achieve high accuracy in the prediction of human judgements on object placements, and to outperform state-of-the-art directly applicable learning models. The solution should also be applicable in different scenarios. To evaluate whether PredART achieves these goals, we studied the accuracy of PredART by applying it to four test scenes from the Unity MARS Test Framework [6] to explore the effectiveness of oracle prediction with deep image regression. In particular, we randomly moved the camera in the test scene, placed virtual objects on different surfaces, and collected 480 testing screenshots. Then, we used Mechanical Turk [9] to label them, and applied PredART to predict their user ratings as test oracles. Our empirical evaluation has the following findings:

- PredART can achieve a Mean Absolute Error (MAE) of 0.047, a Mean Squared Error (MSE) of 0.008 and a Root Mean Squared Error (RMSE) of 0.091. PredART can achieve an accuracy of 85.0% on predicting the average user ratings.
- PredART largely outperforms ResNet [23], the state-of-the-art image regression technique, by 56.9 percentage points, showing that the combination of CNN and MLP to balance feature weights in a hybrid neural network is effective.
- The object type and scene have a minor influence on prediction accuracy.
- PredART can still achieve an MAE of 0.101 and an MSE of 0.041 when performing cross-object prediction.

In sum, this paper makes the following major contributions:

- A novel method for predicting the reality of virtual objects with placement errors that can be used as a test oracle in augmented reality testing.
- A labeled public dataset[1] for future research efforts.
- An empirical evaluation based on 480 screenshots taken during a random exploration of four Unity Mars testing scenes with three different virtual object placements.

The remainder of the paper is organized as follows. In Section 2, we will introduce some background knowledge of AR Testing and the techniques used in PredART. In Section 3, we will describe the details of each step in our approach. After that, we present our empirical evaluation results in Section 4, and discuss some important issues in Section 5. Before we conclude the paper in Section 7, we further summarize related works in Section 6.

## 2 BACKGROUND

In this section, we introduce the background knowledge of several tools and techniques used in our research. Unity Mars is the testing environment we target. Amazon Mechanical Turk is the platform

[1]Our dataset and scripts are all available at https://sites.google.com/view/predart2022

we use to label our dataset, and image regression, CNN, and ResNet are techniques we leverage or compare to.

## 2.1 Unity Mars Test Framework

Unity [1] is a real-time 3D development platform that includes a game-engine (Unity game engine) and an editor. It is used in a variety of industries, including cross-platform game development (mobile, PC/console, and AR/VR), automotive, transportation, and manufacturing (design and prototyping), architecture, engineering, and construction (engineering design, simulation), and film (animation, cinematic). Currently, there are multiple platforms for XR development (ARCore [22], ArKit [2], Hololense [3], Occulus [4], etc.) with their own unique implementations and device-dependent sets of features, which introduces additional complexity for software testing. As a cross-platform development tool, Unity provides a superset of XR[2] features in different package structures. To separate platform and device-specific implementations from the core game engine, Unity introduced a plug-in framework called the XR plug-in framework [5]. This enables software and hardware providers to develop their own Unity plugins to integrate with the Unity engine and make full use of its features via a common API. The XR plug-in framework exposes APIs for common functionalities supported by Unity. These APIs are grouped into multiple subsystems (Figure 3) (e.g., display, faces, image tracking, object tracking, etc.), collectively called XR subsystems.

Unity introduced MARS (Mixed and Augmented Reality Subsystem) in 2020 [37], which provides a suite of specialized AR software tools to help developers author, test, and launch cross-compatible AR apps [36]. Unity MARS provides extensive testing facilities for app developers. The MARS basic environment simulation pack includes a variety of virtual scenes that mimic real-world settings in indoor scenes (living room, bedroom, kitchen, dining room, and office), large indoor scenes (museum, factory, and warehouse), and outdoor scenes (park, and backyard). Unity MARS comes with separate templates of pre-authored scenes and support scripts representing single use-case examples of Unity MARS features. The MARS simulation system supports simulation of planes, body tracking, facial landmarks, point clouds, and raycasting.

A MARS session contains a programmable ARCamera which simulates the movement of a human user (called a MARS user) inside the virtual scene. There are three ways the programmable ARCamera can be controlled: 1) Connected Device—feeding movement and position information from an actual device connected via USB in real time; 2) Playables—playing back a pre-recorded motion along a path inside a virtual scene; 3) Custom Script—moving inside the virtual scene on a pre-programmed path using a custom ARPose driver script. Connected Device—feeding and Playables are commonly used in regression testing with manual test cases, while custom scripts can facilitate automatic testing. In our approach and evaluation, we rely on custom scripts to automatically move the camera and create the training set and the evaluation set.

[2]XR is an umbrella term that includes virtual reality, augmented reality, and mixed reality.

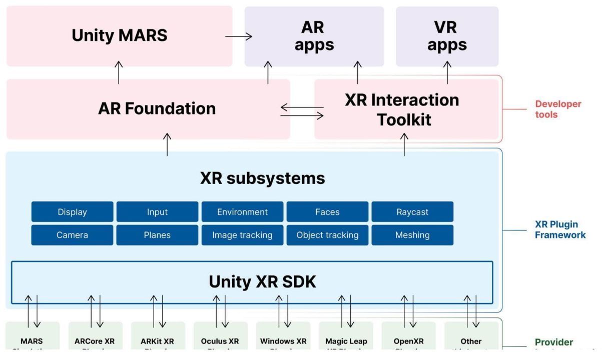


**Figure 3: Unity Platform Architecture**

## 2.2 Amazon Mechanical Turk

Amazon Mechanical Turk [9] is a crowdsourcing website provided by Amazon that allows individuals or organizations (called requesters) to outsource their processes and work to remotely located workers who can perform the tasks online. The requesters post tasks called HITs (Human Intelligence Tasks), which could include anything from simple data labeling, such as identifying the content in an image, to more subjective efforts such as filling out surveys, summarizing texts, and fixing language errors. Amazon Mechanical Turk provides worker filters for the requesters to choose only qualified workers. Some common criteria include the number of HITs being approved and the percent of HITs being approved. More advanced filter criteria may further select a specific group, such as females, senior citizens, or students. However, Amazon Mechanical Turk does not provide any criteria on worker expertise, so tasks that rely on common sense can be relatively well finished, while tasks requiring strong expertise, such as software bug fixing, may not be suitable for it. In this paper, we are using it to judge the reality of virtual objects with placement gaps, which relies mostly on intuition rather than expertise.

## 2.3 Image Regression, CNN and ResNet

Regression analysis is a statistical process that estimates the relationships between a dependent variable and a number of independent variables, called features. Image regression is a technique to predict a numeric value from an image. The basic difference between image classification and image regression tasks is the target variable. In image regression, the target value is continuous, while in image classification, the target value is discrete. For example, if we need to predict house prices based on the images of houses, it will be an image regression task.

CNN is a powerful neural network architecture mainly used for both image classification and image regression. CNNs are regularized versions of MLP, which usually mean fully connected networks. In the fully connected network, each neuron in one layer is connected to all neurons in the next layer. This characteristic of "full connectivity" makes them prone to overfitting. Therefore, it needs regularization to prevent it from overfitting. CNN takes advantage of hierarchical patterns in the data points and assembles patterns of increasing complexity by using smaller and simpler patterns embedded in filters.

ResNet [23], which was proposed in 2015, introduced a new architecture called the Residual Network. In order to solve the problem of the vanishing and exploding of gradient functions, this architecture introduced the concept of a Residual Network. There is a technique called *skip connections* in this network. The skip connection skips the training of a few layers and connects directly to the output. By allowing the network to fit the residual mapping, it learns the underlying mapping instead of layers. ResNet is the most popular neural network used in computer vision. It is able to handle image inputs as well as additional meta information by concatenating them to image features. In our paper, we use ResNet as our baseline approach and propose a hybrid neural network combining MLP and CNN to solve our specific problem.

## 3 APPROACH

As shown in Figure 4, our approach has three main steps. The first step is the automatic generation of training screenshots, in which we use an automatically moving camera to take pictures of different virtual objects being placed with different placement gaps. The second step is data labelling, in which we use Amazon Mechanical Turk to have multiple users label the screenshots and calculate their user ratings. The third step is deep image regression, in which we use the labeled screenshots to train a hybrid deep neural network combining CNN (Convolutional Neural Network) and MLP (Multi-Layer Perceptron) network for prediction.

### 3.1 Creating Screenshots

Whether the placement of a virtual object is realistic depends on various factors, e.g., distance, viewing angle, height of the observer (represented as horizontal angle), and the dimension of the virtual object itself with respect to the surrounding items. Therefore, we need a dataset that exhaustively covers a wide range of values for these variables. Therefore, our first step involves manually setting up a testing scene with an object to be placed, and automatically placing the ARCamera in different poses (positions and viewing angles) with custom scripts. The screenshots are grouped into two sets for deep image regression. These two groups served as the training set and the test set.

For the first group of screenshots (training set), we consider a basic indoor living room scene with only a table (Figure 5). We choose an object model from a set of three models of different sizes (small sized object—apple, medium sized object—table lamp, and large sized object—chair) to be placed on a horizontal plane (*Placement Plane*). The small and medium-sized objects are placed on the tabletop plane, while the large-sized object is placed on the floor plane. The objects are placed at varying placement gaps[3], ranging from $-2$ cm to 8 cm for small-sized objects, $-4$ cm to 8 cm for medium-sized objects, and $-4$ cm to 10 cm for large-sized objects. Positive and negative placement gap thresholds are chosen by trial and error until the results are visually significantly imprecise.

For each object placed at a different placement gap, we use a custom ARPose driver script to set the camera pose (position and rotation). We consider the distance of the camera from the center of the target object to be 2 ft and 4 ft for small and medium-sized objects and 4 ft and 6 ft for large-sized objects. We choose these distances because when using them, the placed object fits well on the screen. For each combination of object, placement gap, and distance, we then consider the horizontal viewing angle (with respect to the horizontal plane) and the vertical viewing angle (with respect to an arbitrary vertical plane). The horizontal viewing angle is chosen from $20^\circ$, $40^\circ$, and $60^\circ$, and the vertical viewing angle is chosen from a range from $0^\circ$ to $300^\circ$ with $60^\circ$ increment. Refer to Table 1 for a detailed description of the combination of feature values.

Given the position of the object and other feature values (placement gap, distance, horizontal and vertical viewing angle), the ARPose drive script calculates the position and rotation of the camera by solving a spherical equation. For each camera position, Unity renders a new frame and the ARPose driver script takes a screenshot using Unity's ScreenCapture API.

For the second group of screenshots (test set), we consider a set of three indoor scenes (dining room, living room, and office), one large indoor scene (factory/warehouse), and one outdoor scene (backyard) as the test scene. Compared to the scenes used for the first group of screenshots, the scenes used in this phase contain more objects, making them closer to real scenes. Therefore, we can also test whether a model trained in simpler scenes can be used in more complicated scenes. We further use a pseudo-random number generator to choose different feature values within a range.

Four different placement gaps are chosen for each scene and object combination, ranging from $-2$ cm to 8 cm for small-sized objects, $-4$ cm to 10 cm for medium-sized objects, and $-6$ cm to 12 cm for large-sized objects with a 1 cm increment step. Similarly, for each scene, virtual object, and placement gap combination, two different floating-point distance values are chosen from the range of 1 ft to 5 ft for small and medium-sized objects, and from the range of 3 ft to 8 ft for large-sized objects. For horizontal and vertical viewing angles, we chose values ranging from $10^\circ$ to $80^\circ$ with $1^\circ$ increment and from $0^\circ$ to $359^\circ$ with $1^\circ$ increment step, respectively.

### 3.2 Data Labelling

Determining whether virtual object placement is realistic relies on users' perception. Therefore, screenshot labeling needs to be done by multiple qualified and responsible users, and Amazon Mechanical Turk is the most popular platform to support it. We collect all of the screenshots and package them as HITs on Amazon Mechanical Turk. For each HIT, we ask the worker to observe the colored virtual object and choose one of two labels: *realistic* and *floating* or *cutoff*. We do not use the term *unrealistic* because it may mislead the worker to consider factors other than the placement gap (e.g., the color of the object). To further help them, we give example screenshots as shown in Figure 6, where virtual objects are placed exactly on the surface, largely above the surface, and largely below the surface, respectively. To make sure the workers are qualified and responsible, we use the filter to select only workers who have finished more than 10,000 approved HITs and have had more than 98% of their finished HITs approved, according to an existing guideline [40].

[3]Placement gap refers to the distance between the placement plane and the plane parallel to the placement plane passing through the object's bottom-most point, where 0cm means the object's bottom-most point is touching the placement plane, negative placement gap means the object is submerged into the placement plane, and positive placement gap means the object is floating over the placement plane.

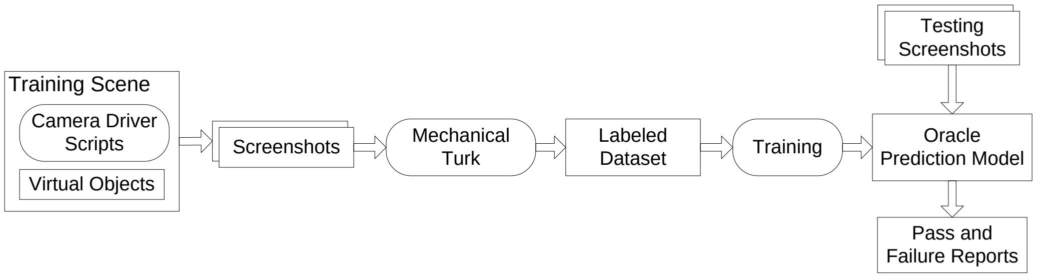


**Figure 4: PredART Overview**

**Table 1: Training Set Feature Description**

| Features | Small sized object (apple) | Medium sized object (table lamp) | Large sized object (chair) |
|---|---|---|---|
| Placement Gap | −2cm / 0cm / 2cm / 4cm / 6cm / 8cm | −4cm / −2cm / 0cm / 2cm / 4cm / 6cm / 8cm | −4cm / −2cm / 0cm / 2cm / 4cm / 6cm / 8cm / 10cm |
| Distance | 2ft / 4ft | 2ft / 4ft | 4ft / 6ft |
| Horizontal viewing angle | $20°$, $40°$, $60°$ | | |
| Vertical viewing angle | $0°$, $60°$, $120°$, $180°$, $240°$, $300°$ | | |
| Total # of combination | $6 \times 2 \times 3 \times 6 = 216$ | $7 \times 2 \times 3 \times 6 = 252$ | $8 \times 2 \times 3 \times 6 = 288$ |
| Total # of training screenshot | $180 + 252 + 288 = 720$ | | |

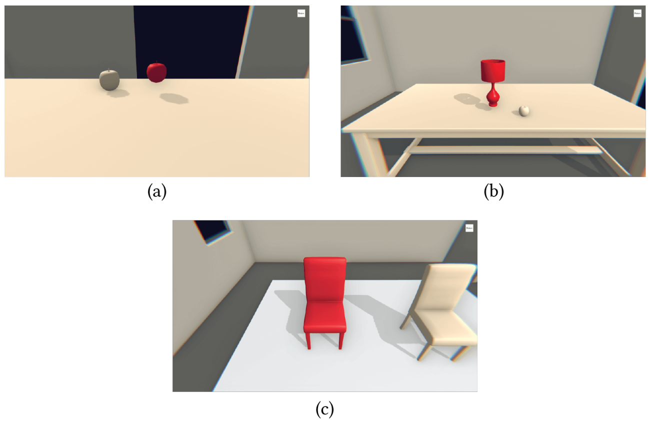
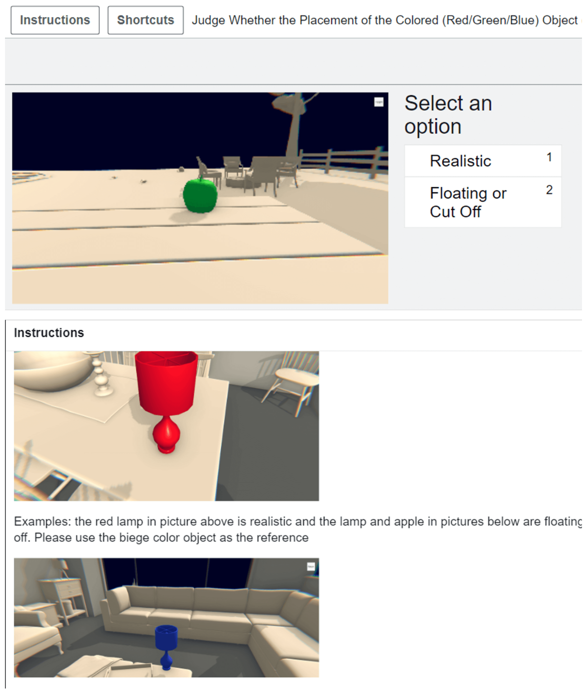


**Figure 5: Automatically generated training screenshots (a) a small object placed on table top plane at 4cm placement gap, (b) a medium object placed on table top plane at 6cm placement gap, (c) a large object placed on floor plane at 0cm placement gap.**

**Table 2: # of screenshots generated for test data**

| | |
|---|---|
| Scene and object combination | 12 |
| Placement Gap | 4 |
| Distance, horizontal, and vertical viewing angle combination | 10 |
| total | $12 \times 4 \times 10 = 480$ |

Judgment of realistic object placement can be difficult for some screenshots, but we do not provide an intermediate option (e.g., not very realistic) because we want the workers to try their best to make a decision instead of retreating to a simple intermediate option. Instead, we use multiple (i.e., five in our experiment) workers and average their votes to handle the controversial cases. For example, a screenshot with unanimous *realistic* votes will get a label of 1.0,



Examples: the red lamp in picture above is realistic and the lamp and apple in pictures below are floating off. Please use the biege color object as the reference

**Figure 6: An Exemplar HIT and Instructions**

while a screenshot with only three *realistic* votes out of five will get a label of 0.6. PredART uses a regression model instead of a classification for prediction so that it is able to provide a user rating as a test oracle not only when the object placement is clearly realistic or unrealistic, but also when it is disputable among users.

### 3.3 Hybrid Image Regression

Once we have the labeled screenshots along with the other variables, i.e., horizontal angle, vertical angle, distance from the object, dimension of the object, etc., the next challenge is to feed the data to a machine learning model. As the screenshots have at least thousands of pixels to be clear enough for human judgement, simply concatenating the other variables as features will significantly undermine their effects, resulting in a bias model. Therefore, a hybrid method needs to be employed to achieve more sensible results.

In particular, to make sure the feature vectors of the image (which are high-dimensional) are not dominating over meta-information features and to take advantage of CNN for image data, we developed a hybrid neural network with the structure shown in Figure 7.

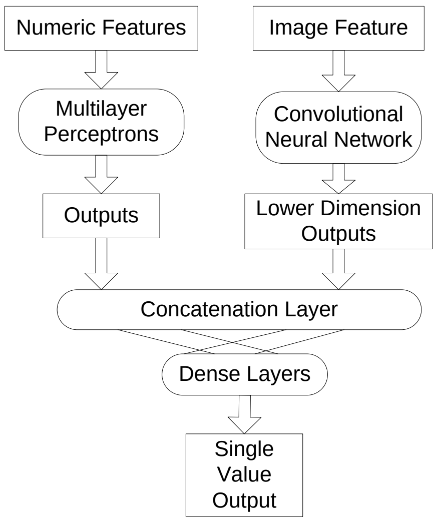


**Figure 7: Structure of the Hybrid Image Regression Model**

From the figure, we can see that the neural network model uses two separate networks to process numerical and image inputs. In particular, PredART feeds the numerical features of screenshot meta information (vertical viewing angle, horizontal viewing angle, distance, and placement gap) into an MLP network. At the same time, it feeds the screenshot as an image input into the CNN, which generates lower-dimension output. The first branch (numerical features) in particular accepts 32-d input, whereas the second branch (image features) accepts 128-d input. These branches operate independently of each other until they are concatenated.

To combine the features, we designed a concatenation layer where the output of the MLP network (for meta information) and the CNN (for image) are concatenated. With further dense layers, the concatenated output is finally reduced to a single-dimensional value, which is the final prediction output. The weights in the neural network are randomly initialized. Neural networks are stochastic algorithms. The model is compiled with "*mean_absolute_percentage_error*" loss (which computes the mean absolute percentage error between $y_{true}$ and $y_{pred}$) and an Adam optimizer [28] with learning rate decay (Learning rate decay is a technique for training modern neural networks. It starts training the network with a large learning rate and then slowly reduces/decays it until a local minima is obtained. It is empirically observed to help both optimization and generalization). To be specific, the MLP module contains 3 layers , while the CNN module contains 5 layers with 3x3 convolution kernels and padding size = 1.

## 4 EVALUATION

In this section, we present an evaluation of PredART on 480 screenshots taken from test scenes of the Unity Mars framework.

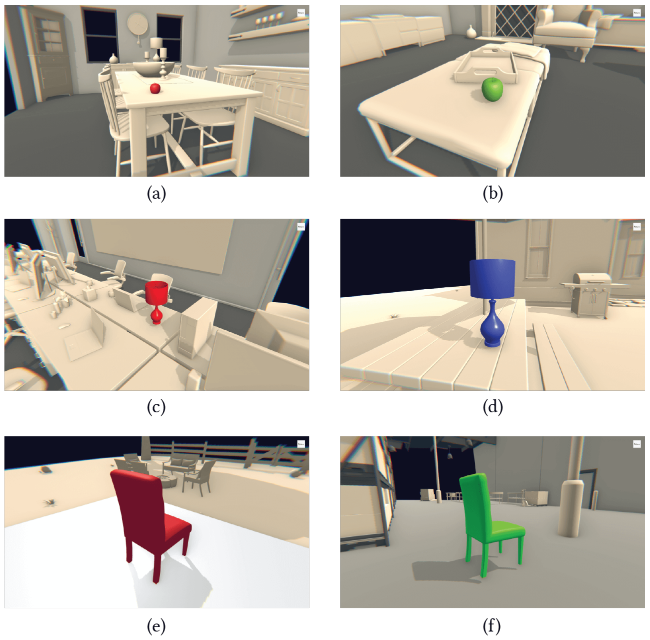


**Figure 8: Automatically generated test set screenshot (a) small object, red, misplacement:** $-1$**cm, distance:** $3.5925$**ft, x-angle:** $12°$**, y-angle:** $287°$**; (b) small object, green, misplacement:** $2$**cm, distance:** $1.794$**ft, x-angle:** $29°$**, y-angle:** $67°$**; (c) medium object, red, misplacement:** $9$**cm, distance:** $4.22$**ft, x-angle:** $28°$**, y-angle:** $50°$**; (d) medium object, blue, misplacement:** $3$**cm, distance:** $2.1555$**ft, x-angle:** $11°$**, y-angle:** $348°$**; (e) large object, red, misplacement:** $-3$**cm, distance:** $4.8533$**ft, x-angle:** $29°$**, y-angle:** $138°$**; (f) large object, green, misplacement:** $7$**cm, distance:** $5.6611$**ft, x-angle:** $11°$**, y-angle:** $122°$**;**

### 4.1 Research Questions

In our evaluation, we try to answer the following five research questions.

- **RQ1:** Is it feasible to predict users' reality judgements on virtual objects with deep image regression?
- **RQ2:** Does our proposed hybrid deep neural network outperform the state-of-the-art image regression technique?
- **RQ3:** Do testing scenario-related factors such as test scenes and virtual object types affect the prediction accuracy?
- **RQ4:** Is PredART able to perform cross-object prediction of users' reality judgement?
- **RQ5:** Is PredART able to detect unrealistic object placement during testing of AR apps in Unity Mars virtual scenes?

For RQ1, we answer the basic question of how accurately PredART can predict the realisticness of object placement. For RQ2, we compare PredART with ResNet to find out whether PredART outperforms state-of-the-art approaches that can be directly applied to the problem. ResNet [23] is a widely used, state-of-the-art model for its superior performance over other popular CNN-based methods (e.g., VGG [53] and MobileNet [25, 34]). We chose ResNet-18 for a fair comparison as it has comparable architectural parameter sizes. For RQ3, we explore the applicability of PredART to find out whether it can achieve similar accuracy in different scenarios. For RQ4, we look into whether PredART can address a practical challenge (i.e., it is impossible to train a model with all of the virtual objects on which it will make predictions). So, we evaluate PredART on cross-object prediction tasks to find out its accuracy when the virtual object in the prediction is unknown at the training phase. For RQ5, we validate the practical usefulness of PredAT by checking whether it can detect unrealistic object placements in the testing process of real AR apps in Unity Mars virtual scenes.

## 4.2 Evaluation Configuration

The construction process of our testing set is described in Section 3.1 (generation of testing screenshots) and Section 3.2 (labelling ground truth of the testing set), respectively. Figure 8 shows six screenshots in the testing set from different scenes, with different virtual objects being placed and using different meta information for the camera. Due to labelling budget and multiple labels required for each screenshot, our data set is limited to 1,200 screenshots with 480 screenshots in the test set. However, our test set covers different virtual objects, AR test scenes, misplacement values, angles, and distances. Unity Mars has a limited number of built-in AR test scenes, and thus taking more screenshots of the same scenes may not be very helpful. We consider the dataset to be representative because AR app developers are likely to use the built-in AR test scenes. Therefore, our evaluation set is close to the actual data to be used in AR app testing. We plan to expand our evaluation by incorporating AR test scenes released in new versions of Unity Mars and other popular open-source VR scenes that could be used by AR developers in their testing.

To make sure our labeling is not largely affected by irresponsible workers, we performed an outlier analysis [8] on the labels of the testing set. In particular, if a worker chooses a label different from all four other workers, the worker is considered an outlier for the specific screenshot. Since the judgement of object placement can be subjective, it is fine if a worker is sometimes an outlier. But if a worker is often an outlier (close to 50% or higher), it indicates unreliable labeling. We performed outlier analysis on the labels in our testing set, and found only 36 out of 480 (7.5%) screenshots were labeled by an (>30%) outlier (indicating the worker is an outlier more than 30% of the time). Considering each screenshot is labeled by five workers, the influence of potential unreliable labels is minimal.

We evaluate the model's performance in two different settings. In the first setting, we use all three objects' screenshots to train the model and predict realism values for all three objects' screenshots. In the second setting (cross-object setting), we use two of the objects' screenshots during training and predict the realism values for the third object. We then compare the predicted realism value $\hat{y}$ with the gathered label data $y$ as described in section 3.2. For ResNet as the baseline technique, we use its default network structure in the research paper [23] and directly concatenate numeric features and image features.

## 4.3 Image Regression Metrics

In data regression, the commonly used metrics are *mean average error (MAE)*, *mean squared error (MSE)*, and *root mean squared error (RMSE)* [57]. The three metrics measure how predicated values are different from ground truth values. The formulas for the three metrics are presented as follows. In the formula, $N$ denotes the number of data points, $y$ denotes the predicated value of each data point, and $\hat{y}$ denotes the ground truth value of each data point.

$$MAE = \frac{1}{N}\sum |y - \hat{y}| \tag{1}$$

$$MSE = \frac{1}{N}\sum (y - \hat{y})^2 \tag{2}$$

$$RMSE = \sqrt{\frac{1}{N}\sum (y - \hat{y})^2} \tag{3}$$

To gain a more intuitive understanding of the prediction results, we further introduce two metrics: *accuracy* and *accuracy(±)*. To calculate these metrics, we first convert the predicted rating $\hat{y}$ to the closest discrete rating $y$ (Equation 4). We define accuracy as the percentage of screenshots where category($\hat{y}$) = $y$. For accuracy(±) we consider predictions that were one category above or below to be accurate and calculate the percentage.

$$\text{category}(\hat{y}) = \begin{cases} 0.0, & \text{if } 0.0 \le \hat{y} < 0.1 \\ 0.2, & \text{if } 0.1 \le \hat{y} < 0.3 \\ 0.4, & \text{if } 0.3 \le \hat{y} < 0.5 \\ 0.6, & \text{if } 0.5 \le \hat{y} < 0.7 \\ 0.8, & \text{if } 0.7 \le \hat{y} < 0.9 \\ 1.0, & \text{if } 0.9 \le \hat{y} < 1.0 \end{cases} \tag{4}$$

## 4.4 Evaluation Results

*4.4.1 Overall Prediction Accuracy.* To answer research question **RQ1**, we calculated all the metrics for the basic setting where we apply PredART with all the training data for training and all the testing data for testing. For the basic setting, as shown in the second row of Table 4, we are able to achieve an accuracy (column Acc.) of 85.0 and an accuracy(±) (column Acc. (±)) of 96.0, indicating for 96% of the cases we are able to predict a user rating at least one level above or below the ground truth. We also achieve MAE, MSE, and RMSE of 0.047, 0.008, and 0.091, respectively, showing that on average our predicted rating is less than 0.05 away from the ground truth. To better understand the prediction results of PredART, we further draw an X-Y table as in Table 3. In the X-Y table, Column 1 presents all labeled ratings, and Columns 2–7 present the percentage of all screenshots with corresponding labeled ratings and predicated ratings. For example, the cell at Column 2 and Row 2 shows 9.8%, indicating 9.8% of screenshots are labeled as 0.0 and predicted as 0.0 (predicted score less than 0.1). In the X-Y table, the diagonal cells present the correctly predicted data points, and cells far from the diagonal cells present the percent of data whose predicted values are far away from ground truth. From the table, we can see that, for the least realistic categories (i.e., ratings of 0.0 and 0.2), sometimes PredART predicts the rating wrong, but the prediction never goes higher than 0.4; for the most realistic categories (i.e., ratings of 0.8 and 1.0), the prediction never goes

**Table 3: The X-Y Table for the prediction**

| Label \ Pred. | 0.0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 |
|---|---|---|---|---|---|---|
| 0.0 | 9.8% | 2.5% | 0.0% | 0.0% | 0.0% | 0.0% |
| 0.2 | 2.3% | 8.5% | 0.0% | 0.0% | 0.0% | 0.0% |
| 0.4 | 1.5% | 2.3% | 9.8% | 0.0% | 0.0% | 0.0% |
| 0.6 | 0.0% | 1.0% | 0.8% | 9.4% | 0.6% | 0.2% |
| 0.8 | 0.0% | 0.0% | 0.0% | 0.0% | 18.1% | 3.8% |
| 1.0 | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% | 29.4% |

**Table 4: Comparison with ResNet**

| Approach | MAE | MSE | RMSE | Acc. | Acc. (±) |
|---|---|---|---|---|---|
| PredART | 0.047 | 0.008 | 0.091 | 85.0 | 96.0 |
| ResNet | 0.175 | 0.041 | 0.202 | 28.1 | 56.9 |

under 0.8; for the blurred area (0.4 and 0.6), the prediction can go further to 0.0 and 1.0, but the likelihood is very low.

*4.4.2 Comparison with Resnet.* To answer research question **RQ2**, we compare PredART metrics with a state-of-the-art image regression technique, ResNet, in the basic setting where we use training data and test data of all three different sized objects. We present the findings in Table 4, where Columns 2–6 present the MAE, MSE, RMSE, accuracy, and accuracy (±), respectively. From the table, we can observe that PredART outperforms ResNet in all the metrics: MAE, MSE, and RMSE. PredART achieves MAE, MSE, and RMSE of 0.047, 0.008, and 0.041 respectively, while ResNet achieves MAE, MSE, and RMSE of 0.175, 0.041, and 0.202 respectively.

In terms of accuracy and accuracy (±) PredART significantly outperforms ResNet. PredART achieves an accuracy and an accuracy (±) of 85.0 and 96.0, respectively. ResNet achieves an accuracy and an accuracy (±) of 28.1 and 56.9. PredART can correctly predict ranking in situations where ResNet fails. Therefore, even when we consider one category above or below, PredART is able to outperform ResNet.

It should be noted that we compare with ResNet because ResNet can be directly applied to our problem with feature concatenation. It is possible to apply some more advanced image classification models [17] [60] with some adaptations (e.g., transfer learning). Since these adaptations themselves can be a research problem and have technical challenges, we do not consider them in our evaluation but plan to explore their effectiveness in our future work.

*4.4.3 Influence of Test Scenarios.* To answer research question **RQ3**, in our testing set, we consider different testing scenes in Unity Mars, which are much more complicated than our training scene, with many other objects as additional noises. It should be noted that for training PredART we use only one scene with one or two additional objects to offer a sense of scale for human workers. However, for testing screenshots, we use virtual scenes of three different scales: indoor scenes (dining room, living room, and office), large indoor scenes (factory, warehouse), and outdoor scenes (backyard). Each scene contains numerous objects of various shapes and sizes, unseen by the model. In some cases, the objects are partially occluded by the surrounding objects. We present the evaluation results on all five metrics for different scenes in Table 5.

**Table 5: Results on Different Testing Scenes**

| Scene Name | MAE | MSE | RMSE | Acc. | Acc. (±) |
|---|---|---|---|---|---|
| Backyard | 0.038 | 0.007 | 0.083 | 87.5 | 95.8 |
| Dining Room | 0.035 | 0.003 | 0.055 | 92.5 | 100.0 |
| Factory | 0.022 | 0.002 | 0.049 | 92.5 | 100.0 |
| Living Room | 0.043 | 0.006 | 0.079 | 84.2 | 97.5 |
| Office | 0.077 | 0.017 | 0.13 | 75.8 | 90.8 |

**Table 6: Results on Different Objects**

| Object | MAE | MSE | RMSE | Acc. | Acc. (±) |
|---|---|---|---|---|---|
| Apple | 0.043 | 0.005 | 0.072 | 88.8 | 98.1 |
| Lamp | 0.034 | 0.005 | 0.073 | 90.0 | 97.5 |
| Chair | 0.064 | 0.014 | 0.119 | 76.3 | 92.5 |

From the table, we can observe that PredART obtains the lowest accuracy of 75.83 and the lowest accuracy (±) of 90.83 in the office scene. The reason behind these results may be that the office scene contains a lot of additional objects such as monitors, mice, pens, chairs, etc., resulting in more noise for the image regression model. As the number of surrounding objects in the vicinity becomes smaller, PredART's performance increases. In the living room, we achieve an accuracy of 84.17 and 87.5. For the backyard scene, the dining room scene, and the factory scene, we achieved the best performance. We achieve an accuracy and accuracy (±) of 97.5 and 95.83, respectively, for the backyard scene. For the dining room scene and the factory scene, we achieve an accuracy and accuracy (±) of 92.5 and 100, respectively.

We further studied how different virtual objects might affect the prediction results. The comparison of prediction results is shown in Table 6. We compute the metrics values for each of the three virtual objects used in the testing for each row of row 2-4 in the table. From the table, we can see that for the apple and the lamp, our image regression model predicts precise results, with an accuracy of 88.8 and 90.0, respectively. The accuracy (±) is also very high, reaching 98.1 and 97.5. In contrast, the model does not perform very well for the chair. The reason may be that the chair is much larger and is placed on the ground (while the other two are both placed on tables). Furthermore, the shape of the chair is less regular than that of the apple and the lamp.

*4.4.4 Cross-Object Prediction.* Cross-object prediction is very important in the application of PredART because it could be difficult to label training data for all virtual objects to be tested. To answer research question **RQ4**, we calculated all the metrics for the cross-object setting, in which we applied PredART with screenshots of two objects as training data and screenshots of the third object as testing data. The metric in Table 7) shows that our prediction results are comparable to the results in Table 6. We even get better results for the apple. In particular, for the apple, we achieved an accuracy of 97.5 and an accuracy (±) of 98.8. The training data of the apple may have caused some overfitting issues.

As the object size gets bigger, the results become worse, especially for the chair, which is understandable because training data on smaller virtual objects may not be able to cover the cases of larger objects. From this result, we can see that object size matters, so it is important to include virtual objects of all sizes in the training data.

**Table 7: Cross Object Prediction**

| Object | MAE | MSE | RMSE | Accuracy | Accuracy (±) |
|---|---|---|---|---|---|
| Apple | 0.048 | 0.006 | 0.079 | 97.5 | 98.7 |
| Lamp | 0.087 | 0.041 | 0.203 | 86.2 | 90.6 |
| Chair | 0.167 | 0.075 | 0.273 | 63.1 | 78.8 |

**Table 8: The X-Y Table for cross-object prediction on apple**

| Label \ Pred. | 0.0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 |
|---|---|---|---|---|---|---|
| 0.0 | 22.5% | 0.0% | 0.0% | 0.6% | 0.0% | 0.0% |
| 0.2 | 0.0% | 16.2% | 0.0% | 0.0% | 0.0% | 0.0% |
| 0.4 | 0.0% | 0.0% | 11.9% | 0.0% | 0.0% | 0.0% |
| 0.6 | 0.0% | 0.0% | 0.0% | 9.4% | 0.6% | 0.0% |
| 0.8 | 0.0% | 0.0% | 0.0% | 0.0% | 10.6% | 1.2% |
| 1.0 | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% | 26.9% |

**Table 9: The X-Y Table for cross-object prediction on lamp**

| Label \ Pred. | 0.0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 |
|---|---|---|---|---|---|---|
| 0.0 | 6.9% | 0.0% | 0.6% | 0.0% | 0.0% | 0.0% |
| 0.2 | 0.0% | 3.1% | 0.0% | 0.0% | 0.0% | 3.1% |
| 0.4 | 0.0% | 0.0% | 11.2% | 0.0% | 0.0% | 4.4% |
| 0.6 | 0.0% | 0.0% | 0.0% | 5.0% | 0.0% | 1.2% |
| 0.8 | 0.0% | 0.0% | 0.0% | 0.0% | 20.0% | 4.4% |
| 1.0 | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% | 40.0% |

We further draw three X-Y tables for each virtual object to understand more detailed prediction results. Table 8 represents the X-Y table for cross-object prediction results on the small-sized object—apple, where medium-sized and large-sized objects are used for training. We see that 97.5% of the values fall on the diagonal column (also represented in row 2 and the accuracy column of Table 7). As we move to the medium-sized object—lamp, where small and large-sized objects are used for training, we find 13.75% of the data outside of the diagonal column (Table 9). Finally, we apply PredART to small and medium-sized objects for training and the large-sized object—chair for testing. where we have 63.125 accuracy and 78.75 accuracy (±) (row 4, column 4-5 of Table 7).

We also notice that, in all three cases, the most inaccurate predictions fall in the last column of the corresponding X-Y tables. This means that we are predicting some unrealistic screenshots as realistic, which could result in false negatives in testing. Such cases are especially severe for the lamp with a ground truth label of 0.2, where half of the screenshots are predicted as 1.0. Therefore, we can see that, although the general metrics show acceptable results, the X-Y table can be more helpful in understanding the details. We may need to develop further techniques to lead the prediction result bias to the unrealistic side, so that even if there are prediction errors, they will be false positives, which are less harmful.

## 4.5 Validation on AR Apps

To further validate whether our approach can detect unrealistic object placements in AR apps, we applied PredART to screenshots

**Table 10: The X-Y Table for cross-object prediction on chair**

| Label \ Pred. | 0.0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 |
|---|---|---|---|---|---|---|
| 0.0 | 2.5% | 1.2% | 0.6% | 0.0% | 0.0% | 1.9% |
| 0.2 | 0.0% | 7.5% | 0.0% | 0.0% | 0.0% | 2.5% |
| 0.4 | 0.0% | 0.6% | 8.1% | 0.0% | 1.2% | 3.1% |
| 0.6 | 0.6% | 0.0% | 0.0% | 8.8% | 1.2% | 9.4% |
| 0.8 | 0.0% | 0.0% | 0.6% | 1.2% | 16.9% | 10.6% |
| 1.0 | 0.0% | 0.0% | 0.0% | 0.0% | 1.9% | 19.4% |

taken from three AR apps: *SimpleAR*, *Interaction*, and *FeatheredPlanes*. They are part of the official open-source showcase apps[4] from Unity, and we chose these three because they are mostly related to the object placement feature in AR. In particular, *SimpleAR* automatically places a yellow semi-transparent plane on each of the detected surfaces. There are buttons on the screen to let the users pause or resume the AR experience. *Interaction* allows a user to place a 3D model on a detected surface and perform some interactions such as movement, rotation, and resizing. *FeatheredPlanes* allows a user to place a 3D model on a feathered plane, representing a detected surface with dotted mesh fading toward the edges.

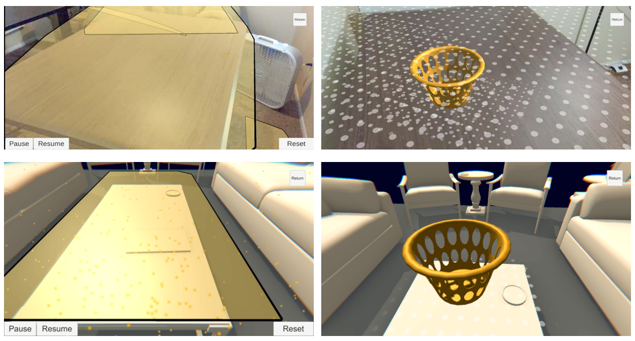


**Figure 9: Screenshots of *SimpleAR* (left) and FeatheredPlane (right) in real scenes (upper) and Unity Mars virtual scenes (lower)**

To perform the validation, we load the source code of each of the three apps into Unity and then manually test them in the virtual scenes of Unity Mars. To be consistent with prior evaluations, we tested each app on the same five virtual scenes: living room, dining room, backyard, office, and factory. The upper row of Figure 9 shows the screenshots of the apps *SimpleAR* (left) and *FeatheredPlanes* (right) in real scenes. During the testing process, we randomly take screenshots from different distances, horizontal angles, and vertical angles. Since unrealistic object placements can be sparse in reality, to make sure they are observable in the evaluation, we plant some unrealistic object placements when taking screenshots by randomly mutating the placed object's position. It should be noted that we mutate an object together with the detected surface it is placed on because objects can be placed on only detected surfaces in most AR frameworks, and object misplacement happens mainly due to imprecisely detected surfaces. So, for unrealistic object placement,

[4]https://github.com/Unity-Technologies/arfoundation-samples

**Table 11: Prediction on AR App Screenshots**

| App | MAE | MSE | RMSE | Accuracy | Accuracy (±) |
|---|---|---|---|---|---|
| *SimpleAR* | 0.093 | 0.027 | 0.163 | 73.3 | 86.7 |
| *Interaction* | 0.107 | 0.059 | 0.242 | 73.3 | 86.7 |
| Feathered | 0.080 | 0.024 | 0.155 | 66.7 | 93.3 |

**Table 12: The X-Y Table for Predict on AR App Screenshots**

| Label \ Pred. | 0.0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 |
|---|---|---|---|---|---|---|
| 0.0 | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% |
| 0.2 | 0.0% | 0.0% | 4.4% | 0.0% | 0.0% | 2.2% |
| 0.4 | 0.0% | 0.0% | 4.4% | 0.0% | 0.0% | 0.0% |
| 0.6 | 0.0% | 0.0% | 0.0% | 2.2% | 0.0% | 8.9% |
| 0.8 | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% | 15.6% |
| 1.0 | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% | 62.2% |

if the underlying plane is visible, it will be floating together with the object. The lower row of Figure 9 shows the screenshots of mutated object placements from *SimpleAR* and *FeatheredPlanes*.

From the three AR apps, we took 15 screenshots from each app to form a dataset of 45 screenshots[5]. Then, we uploaded all of them to Amazon Mechanical Turk for labeling (each screenshot was labeled by five turks), and applied our pre-trained prediction model (trained with the whole training set of 720 screenshots) to the dataset to acquire the predicted scores of each screenshot. Finally, we compared the labeled scores and predicted scores, and the results are presented in Table 11 and Table 12.

From Table 11, we can observe that the results of PredART on real AR app screenshots are comparable to those of cross-object prediction in Table 7. This is understandable because we are also performing cross-object prediction (all the objects in the AR apps are unseen by the model) for AR app screenshots. From Table 12, we can see that for all score ranges, PredART is able to precisely predict the scores of most screenshots, and for most of the non-realistic object placement screenshots (score < 0.5), PredART can correctly predict their scores to be less than 0.5. It should be noted that when mutating objects with their planes, for consistency we used the same range of placement gaps as described in Section 3.1 for generating test sets. It seems that the placement gap range resulted in fewer non-realistic screenshots in the AR app testing dataset. This is probably caused by the visible planes (e.g., yellow planes, feathered planes) in our subject apps, which are mutated together with the objects. Since planes are much larger, the placement gap can be less observable. On the other hand, the more sparse the distribution of non-realistic object placements, the closer to the realistic testing case where placement errors are sparse and the data is more unbalanced. Nevertheless, our evaluation results show that PredART can effectively identify non-realistic screenshots in such scenarios.

## 4.6 Summary of Findings

In this subsection, we summarize our findings to answer our research questions as follows.

[5]The dataset is available at the project website.

- PredART is effective because it can achieve very high metric values in predicting user ratings of virtual objects' placement.
- PredART outperforms ResNet, the state-of-the-art image regression technique.
- Noises in scenes and objects' sizes all moderately affect prediction results.
- On cross-object prediction, PredART achieves acceptable metric values. Techniques making the results lean to the unrealistic side may help PredART in its practical usage.
- On screenshots from real AR apps tested on Unity Mars, PredArt is able to achieve results comparable to those from controlled testing scenarios.

## 4.7 Threats to Validity

The major external threat to our validity is that our evaluation only covers a small number of scenes and virtual objects [58] [20]. Due to the cost of labeling with multiple workers, we are not able to create a huge dataset as an initial study, but we cover all the different styles of scenes (small indoor scenes, large indoor scenes, and outdoor scenes) in Unity Mars, and objects of different sizes. We also studied the influence of these factors to indicate directions for future research. To further reduce the threats, we plan to enlarge our dataset to consider more test scenes and more types of virtual objects. The major internal threat to our validity is the potential mistakes made by labelling workers. Since we have five workers for each screenshot, the influence of one mistake is lessened. Furthermore, we performed outlier analysis to further confirm that the potential mistakes made by labeling workers are limited to a very small portion of the dataset. To further reduce this threat, we plan to consider using more workers for the labeling, and then study whether the labeling results stay stable when more workers are added. Using this approach, we can find out how many workers we need to achieve a stable rating of screenshots.

## 5 DISCUSSION

In this section, we discuss how our technique can help detect bugs in AR software in practice.

## 5.1 Limitations of PredART

PredART mainly has two limitations. The first limitation is that our approach is designed to be used in testing scenes (i.e., Unity Mars). If we want to apply PredART to physical phones and real-world scenes, the potential challenge is whether the feature values used in our model are still accessible. In particular, the screenshots can be easily acquired at runtime, and the horizontal angle and distance can be calculated from sensor information. But the vertical angle and misplacement could be hard to acquire at runtime. Therefore, we may need to adapt our feature set. Meanwhile, once migrated to real-world scenes, our approach has the potential to automatically make object placement more accurate. We can change our prediction goal from whether the object's placement is realistic or not to whether the object has a positive or negative misplacement. We plan to work on this direction in the following projects. The second limitation is that PredART handles only screenshots, which are more suitable for static virtual objects. To determine the realisticness of moving virtual objects, video clips may be needed and more features (e.g., the change of positions and angles) may also need to be considered.

### 5.2 Industrial Relevance

PredART can directly help AR software quality engineers in their testing tasks by providing automatic test oracles so that they do not need to keep watching test videos and hire multiple people to judge whether the placement of objects is realistic. The warnings raised by PredART may help AR developers find bugs in their application code. They may also help product managers predict user satisfaction with their product, determine whether the AR feature is mature enough to be released, and help software architects select more compatible AR underlying platforms.

## 6 RELATED WORK

### 6.1 VR and AR Application Testing

The authors are not aware many research effots on VR and AR Testing. Recently, Wang proposed VRTest [55], a test framework for VR software. It extracts information about virtual objects at runtime, and automatically guide the player camera towards these objects to interact with them. Besides the research effort on VR/AR software testing, there are also some research efforts on game testing. Wuji [63] is a technique that supports automatic testing of games based on evolutionary algorithms and reinforcement learning. It explores the game spaces and branches, as well as makes progress by passing stages. Zhao et al. [62] proposed an approach to enhance playing tactics in game testing by learning from player action sequences. Bergdahl et al. [13] proposed an approach to augment existing manually written test scripts with reinforcement learning. ARCHIE [31] is a framework to collect feedback from manual testers and system state to identify and debug issues. Scheibmeir et al. [52] present a framework that uses machine learning techniques to detect object presentation in the physical world. Compared with these efforts, our research focuses on the impact of virtual object placement errors, which is an important factor affecting the success of AR apps.

### 6.2 Test Oracle Generation

Test oracle generation has long been a bottleneck in automatic testing. A lot of earlier work proposed promising approaches but were not very effective due to the limitations of techniques at the time. These efforts have been summarized in two surveys in 2014 [11, 47]. Metamorphic testing takes advantage of the known correlations between test input changes and test oracle changes (e.g., feeding a subset of input should always result in an output that is a subset of the original output). It has been widely applied to the testing of scientific software [26] and security testing [16]. Donaldson et al. [18] propose to use metamorphic testing to test the compilation of shader scripts based on known correlations among rendering transformations. Langdon et al. [30] propose using deep learning to predict partial test oracles for mutation testing. Goffi et al. [21] propose mining texts from Java API Documents to generate test oracles on exceptional behaviors. Ceccato et al. [14] propose using machine learning to learn a model for legal SQL statements and then use it as an oracle in SQL injection testing. Mariani et al. [39] propose Augusto, which creates GUI test cases with oracles for common GUI operations such as log-in and CRUD (Creation, Reading, Update, and Deletion). Menghi et al. [42] proposed using simulink models to automatically create test oracles for continuous and uncertain output, although the specification process is still manual. Walsh et al. [54] propose to use the relationship among multiple layout outputs to automatically detect layout errors without using an oracle. Test migration techniques [12] [48] create test oracles by migrating oracles from existing tests. Chen et al. [15] propose GLIB to detect game GUI glitches by machine learning. GLIB uses a code-based data augmentation technique to automatically enlarge the training data. PredART is different from all of the above efforts because it targets a different application domain (i.e., AR applications) and a type of failure that heavily relies on human perception.

### 6.3 Studies on VR, AR, and Game Software

There have also been some empirical studies on VR software and video game software. Murphy-Hill et al. [44] presented results from a survey and interviews with video game developers to understand the major challenges between video game development and traditional software development. Later, Washburn et al. [56] performed an empirical study on the failure of game projects to find out what the major pitfalls in game projects are. Lin et al. [33] studied the characteristics of updates on the Steam platform to understand the priority of game updates in practice. Rodriguez and Wang [50] investigated the popularity trends and common project structures of open source virtual reality software projects. Li et al. [32] studied bug reports for web applications supporting extended reality to find out their commonalities. Molina et al. [43] studied code-asset dependency within VR software projects by extracting the direct association between VR objects and script files, the compositional relations between VR objects, and event triggering relations between scripts and VR objects. Pascarella et al. [46] studied open source video game projects to understand their characteristics and the difference between game and non-game development. Nusrat et al. [45] studied the major types of performance optimization in Unity-based virtual reality applications. Zhang et al. [61] studied possible solutions to detect potential privacy leaks in mobile augmented reality apps.

## 7 CONCLUSIONS

In this paper, we studied the feasibility of predicting users' perception of realisticness on virtual object placement with gaps. The predicted user rating can be used as an automatic test oracle in AR testing. We propose an approach called PredART which concatenates the output of convolutional neural networks and multi-layer perceptrons to better balance feature weights. Our empirical evaluation based on mechanical turk labeling shows that PredART is able to achieve an average accuracy of 85.0%, an MAE of 0.047, an MSE of 0.008, and an RMSE of 0.091. The evaluation also reveals that PredART outperforms the state-of-the-art image regression technique ResNet, and it is effective in cross-object prediction and prediction of screenshots of real AR apps tested in Unity Mars virtual scenes. In the future, we plan to (1) expand the training and evaluation set of our evaluation with more scenes and (2) study the predictability of realisticness from short video clips.

## ACKNOWLEDGMENTS

We would like to thank Dongfang Liu and Cheng Han from the Department of Computer Engineering at Rochester Institute of Technology for their kind help on the setup and configuration of computer vision tools. This research is supported in part by NSF awards 1736209, 1846467, 2007718, and 2221843.

## REFERENCES

[1] 2005. Unity. https://unity.com.

[2] 2018. Apple ARKit. https://developer.apple.com/augmented-reality/.

[3] 2018. Microsoft Hololens. https://www.microsoft.com/en-us/hololens.

[4] 2018. Oculus Rift. https://www.oculus.com/.

[5] 2019. XR Plug-in Framework. https://docs.unity3d.com/Manual/XRPluginArchitecture.html.

[6] 2020. Unity MARS. https://unity.com/products/unity-mars.

[7] 2021. Google AR Core. https://developers.google.com/ar.

[8] Charu C Aggarwal. 2017. An introduction to outlier analysis. In *Outlier analysis*. Springer, 1–34.

[9] Amazon. 2005. Amazon Mechanical Turk. https://www.mturk.com.

[10] Ronald T Azuma. 1997. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments* 6, 4 (1997), 355–385.

[11] 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

[12] Farnaz Behrang and Alessandro Orso. 2019. Test migration between mobile apps with similar functionality. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 54–65.

[13] Joakim Bergdahl, Camilo Gordillo, Konrad Tollmar, and Linus Gisslén. 2020. Augmenting automated game testing with deep reinforcement learning. In *2020 IEEE Conference on Games (CoG)*. IEEE, 600–603.

[14] Mariano Ceccato, Cu D. Nguyen, Dennis Appelt, and Lionel C. Briand. 2016. SOFIA: An automated security oracle for black-box testing of SQL-injection vulnerabilities. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 167–177.

[15] Ke Chen, Yufei Li, Yingfeng Chen, Changjie Fan, Zhipeng Hu, and Wei Yang. 2021. Glib: towards automated test oracle for graphically-rich applications. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1093–1104.

[16] Tsong Yueh Chen, Fei-Ching Kuo, Wenjuan Ma, Willy Susilo, Dave Towey, Jeffrey Voas, and Zhi Quan Zhou. 2016. Metamorphic Testing for Cybersecurity. *Computer* 49, 6 (2016), 48–55.

[17] 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.

[18] Alastair F. Donaldson, Hugues Evrard, Andrei Lascu, and Paul Thomson. 2017. Automated Testing of Graphics Shader Compilers. *Proc. ACM Program. Lang.* OOPSLA, Article 93 (oct 2017).

[19] 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 surgery. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 934–943.

[20] Smita Ghaisas, Preethu Rose, Maya Daneva, Klaas Sikkel, and Roel J Wieringa. 2013. Generalizing by similarity: Lessons learnt from industrial case studies. In *2013 1st International Workshop on Conducting Empirical Studies in Industry (CESI)*. IEEE, 37–42.

[21] Alberto Goffi, Alessandra Gorla, Michael D. Ernst, and Mauro Pezzè. 2016. Automatic Generation of Oracles for Exceptional Behaviors. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. 213–224.

[22] Google. 2021. Google ARCore. https://developers.google.com/ar/discover/concepts.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. https://doi.org/10.48550/ARXIV.1512.03385

[24] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. 2004. Design science in information systems research. *MIS quarterly* (2004), 75–105.

[25] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[26] Upulee Kanewala, James M Bieman, and Asa Ben-Hur. 2016. Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels. *Software testing, verification and reliability* 26, 3 (2016), 245–269.

[27] Hirokazu Kato and Mark Billinghurst. 1999. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*. IEEE, 85–94.

[28] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[29] Sang Min Ko, Won Suk Chang, and Yong Gu Ji. 2013. Usability principles for augmented reality applications in a smartphone environment. *International journal of human-computer interaction* 29, 8 (2013), 501–515.

[30] William B. Langdon, Shin Yoo, and Mark Harman. 2017. Inferring Automatic Test Oracles. In *2017 IEEE/ACM 10th International Workshop on Search-Based Software Testing (SBST)*. 5–6.

[31] Sarah M. Lehman, Haibin Ling, and Chiu C. Tan. 2020. ARCHIE: A User-Focused Framework for Testing Augmented Reality Applications in the Wild. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE. https://doi.org/10.1109/vr46266.2020.00013

[32] Shuqing Li, Yechang Wu, Yi Liu, Dinghua Wang, Ming Wen, Yida Tao, Yulei Sui, and Yepang Liu. 2020. An exploratory study of bugs in extended reality applications on the web. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 172–183.

[33] Dayi Lin, Cor-Paul Bezemer, and Ahmed E Hassan. 2017. Studying the urgent updates of popular games on the Steam platform. *Empirical Software Engineering* 22, 4 (2017), 2095–2126.

[34] Dongfang Liu, Yiming Cui, Wenbo Tan, and Yingjie Chen. 2021. Sg-net: Spatial granularity network for one-stage video instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9816–9825.

[35] 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 *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 58. SAGE Publications Sage CA: Los Angeles, CA, 1681–1685.

[36] Unity Technologies Ltd. 2020. Unity MARS (Mixed and Augmented Reality Subsystem). https://blog.unity.com/technology/introducing-unity-mars-a-first-of-its-kind-solution-for-intelligent-ar.

[37] Unity Technologies Ltd. 2021. What's new in Unity Mars. https://blog.unity.com/technology/whats-new-in-unity-mars.

[38] Michael R Lyu, Irwin King, TT Wong, Edward Yau, and PW Chan. 2005. Arcade: Augmented reality computing arena for digital entertainment. In *2005 IEEE Aerospace Conference*. IEEE, 1–9.

[39] Leonardo Mariani, Mauro Pezzè, and Daniele Zuddas. 2018. Augusto: Exploiting Popular Functionalities for the Generation of Semantic GUI Tests with Oracles. In *Proceedings of the 40th International Conference on Software Engineering*. 280–290.

[40] Winter Mason and Sidharth Suri. 2011. How to use mechanical turk for cognitive science research. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 33.

[41] Zeljko Medenica, Andrew L Kun, Tim Paek, and Oskar Palinko. 2011. Augmented reality vs. street views: a driving simulator study comparing two emerging navigation aids. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*. 265–274.

[42] Claudio Menghi, Shiva Nejati, Khouloud Gaaloul, and Lionel C. Briand. 2019. Generating Automated and Online Test Oracles for Simulink Models with Continuous and Uncertain Behaviors. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 27–38.

[43] Jacinto Molina, Xue Qin, and Xiaoyin Wang. 2021. Automatic extraction of code dependency in virtual reality software. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 381–385.

[44] Emerson Murphy-Hill, Thomas Zimmermann, and Nachiappan Nagappan. 2014. Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development?. In *Proceedings of the 36th International Conference on Software Engineering*. 1–11.

[45] Fariha Nusrat, Foyzul Hassan, Hao Zhong, and Xiaoyin Wang. 2021. 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)*. 473–485.

[46] Luca Pascarella, Fabio Palomba, Massimiliano Di Penta, and Alberto Bacchelli. 2018. How is video game development different from software development in open source?. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 392–402.

[47] Mauro Pezzè and Cheng Zhang. 2014. Chapter One - Automated Test Oracles: A Survey. Advances in Computers, Vol. 95. Elsevier, 1–48.

[48] Xue Qin, Hao Zhong, and Xiaoyin Wang. 2019. Testmig: Migrating gui test cases from ios to android. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 284–295.

[49] Iulian Radu. 2012. Why should my students use AR? A comparative review of the educational impacts of augmented-reality. In *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 313–314.

[50] Irving Rodriguez and Xiaoyin Wang. 2017. An empirical study of open source virtual reality software projects. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 474–475.

[51] Franziska Roesner, Tadayoshi Kohno, and David Molnar. 2014. Security and privacy for augmented reality systems. *Commun. ACM* 57, 4 (2014), 88–96.

[52] Jim Scheibmeir and Yashwant K. Malaiya. 2019. Quality Model for Testing Augmented Reality Applications. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*. 0219–0226. https://doi.org/10.1109/UEMCON47517.2019.8992974

[53] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[54] Thomas A. Walsh, Gregory M. Kapfhammer, and Phil McMinn. 2017. Automated Layout Failure Detection for Responsive Web Pages without an Explicit Oracle. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 192–202.

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

[56] Michael Washburn, Pavithra Sathiyanarayanan, Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. 2016. What Went Right and What Went Wrong: An Analysis of 155 Postmortems from Game Development. In *Proceedings of the 38th International Conference on Software Engineering Companion* (Austin, Texas) *(ICSE '16)*. 280–289.

[57] Sanford Weisberg. 2005. *Applied linear regression*. Vol. 528. John Wiley & Sons.

[58] Roel Wieringa and Maya Daneva. 2015. Six strategies for generalizing software engineering theories. *Science of computer programming* 101 (2015), 136–152.

[59] Roel J Wieringa. 2014. *Design science methodology for information systems and software engineering*. Springer.

[60] 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 time. In *International Conference on Machine Learning*. PMLR, 23965–23998.

[61] Xueling Zhang, Rocky Slavin, Xiaoyin Wang, and Jianwei Niu. 2019. Privacy Assurance for Android Augmented Reality Apps. In *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 114–1141.

[62] Yan Zhao, Weihao Zhang, Enyi Tang, Haipeng Cai, Xi Guo, and Na Meng. 2021. A Lightweight Approach of Human-Like Playtesting. *arXiv preprint arXiv:2102.13026* (2021).

[63] 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 learning. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 772–784.