ARCHIE: A User-Focused Framework for Testing Augmented Reality Applications in the Wild

Sarah M. Lehman* Temple University
Haibin Ling† Stony Brook University
Chiu C. Tan‡ Temple University

ABSTRACT

In this paper, we present ARCHIE, a framework for testing augmented reality applications in the wild. ARCHIE collects user feedback and system state data *in situ* to help developers identify and debug issues important to testers. It also supports testing of multiple application versions (called "profiles") in a single evaluation session, prioritizing those versions which the tester finds more appealing. To evaluate ARCHIE, we implemented four distinct test case applications and used these applications to examine the performance overhead and context switching cost of incorporating our framework into a pre-existing code base. With these, we demonstrate that ARCHIE provides no significant overhead for AR applications, and introduces at most 2% processing overhead when switching among large groups of testable profiles.

Index Terms: Software and its engineering—Software testing and debugging; Human-centered computing—User interface toolkits; Human-centered computing—Mixed / augmented reality

1 INTRODUCTION

Augmented reality (AR) applications are increasing in popularity, with applications moving beyond just gaming to fields such as healthcare [46, 59, 70], education [50, 82, 85], manufacturing [41, 61, 65], and so on. On mobile application marketplaces, AR apps for entertainment [12, 24, 27], measurement [3, 4], translation [11, 16], and more are reaching millions and hundreds of millions of downloads. While augmented reality systems have been around since the early 1990s [66], the current rise in popularity is due to two reasons. The first reason is the hardware able to support AR is relatively inexpensive and widely available. The market niche originally dominated by bulky and expensive head-mounted displays (HMDs) such as Google Glass [8], Microsoft Hololens [15], and Oculus Rift [19] is now being filled by conventional smartphones, particularly when paired with inexpensive headsets such as Google Cardboard [9]. Even older devices such as the Google Nexus 5, originally released in 2013 [14], are able to download an AR app as easily as a regular smartphone app.

The second reason is that AR-specific development kits (such as Google's ARCore for Android [10] and Apple's ARKit for iOS [2]) and integrated development environments (such as Unity3D [32] with Vuforia [36]) have made it easier for developers to build AR apps without specialized knowledge of machine learning, computer vision, or graphics. Developers who desire greater control over their systems can make use of computer vision libraries such as TensorFlow [29] or OpenCV [21] to create even more powerful applications.

However, the current AR software development landscape is lacking any software tools to *test and debug* AR apps. AR apps are different from conventional apps in that the augmented images and labels are generated and positioned based on the user's behavior and environment. For example, a navigation app like Google Maps AR [37] will need to determine the user's context from their position, facing direction, and visible entities in their environment to determine what directions to place on the screen and where. This makes AR applications very difficult to test in the wild due to high variability of user behavior and environmental factors. User movement speed, weather, lighting, presence and movement of bystanders, and many other factors all contribute to the performance of an AR system. However, traditional software engineering practices such as the creation and automation of unit tests are ill-equipped to capture and recreate these conditions, as the test writers would be unable to predict which combinations and interactions of conditions a user would encounter. Similarly, because these conditions are likely to occur in tandem, logic to handle one set of conditions is likely to interfere with the logic for another set, making regression testing difficult as well. Once the evaluations are over, even with detailed interviews or questionnaires filled out by the testers, it is impossible to know what underlying errors in the system caused the observable issue. For example, incorrect label placement in an AR navigation app could be due to a lag in the output generation or an error in the placement logic.

To help address these needs, we present **ARCHIE, the Augmented Reality Computer-Human Interaction Evaluator framework**, specifically designed to help AR application developers with system testing and the collection of feedback and debugging data. Our framework makes the following contributions:

- **Identifying performance and usability issues in the wild**. ARCHIE collects performance and usability feedback from testers using a smartwatch-based companion app. Feedback is gathered after specific system events to verify positive results (such as identifying an AR target), and after periods of inactivity to verify negative results (such as failing to identify a target). Testers are also able to identify any specific issues they observe (such as slow output, poor output placement, etc.)

- **Recreating and debugging run-time errors**. When testers submit poor performance or usability feedback, it is important to understand what underlying errors caused that poor feedback. ARCHIE aggregates environmental and system state data during run-time, and provides collections of that data for each instance of poor feedback submitted by the tester. Developers can then query that data, or feed it to the app back in the lab to identify and debug system errors.

- **Testing multiple implementations**. ARCHIE also allows developers to implement multiple logical modules, called *profiles*, which can be used to compare different functional options for an application, such as user interfaces or event recognition. ARCHIE instantiates these profiles at run-time, and interleaves them for the tester, collecting dedicated feedback for each one.

The rest of the paper is organized as follows: Section 2 provides an overview of current issues facing AR application testers and the evaluation methods they employ. Section 3 describes the design

*e-mail: smlehman@temple.edu
†e-mail: haibin.ling@stonybrook.edu
‡e-mail: cctan@temple.edu

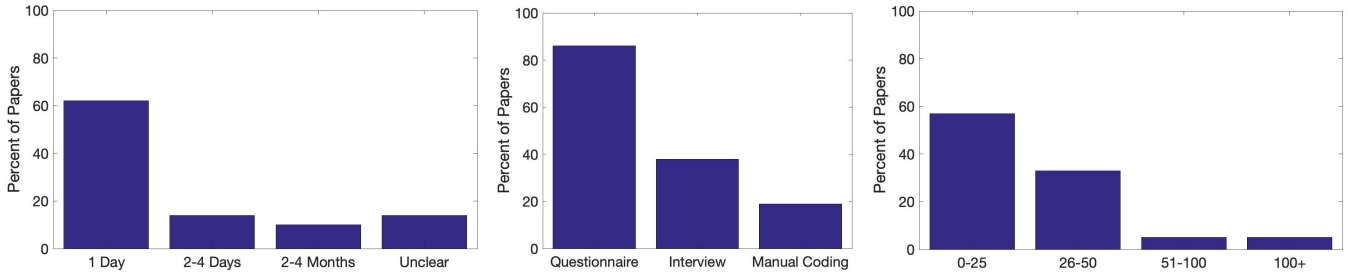
(a) Classifier error generates wrong warning text

(b) Slow processing, quick user movement prevents classification

(c) Poor label placement covers pertinent real-world content (car in intersection)

(d) Alert correctly placed but difficult to read due to poor color choice

Figure 1: Conditions that may negatively impact users' experience of "distracted pedestrian" AR application.



(a) Length of evaluation

(b) Evaluation method *

(c) Size of test group

Figure 2: Evaluation trends of surveyed AR systems. Figures marked with asterisks (*) show papers which fall into multiple categories.

of ARCHIE, and Section 4 describes how developers and testers interact with ARCHIE. Sections 5 and 6 evaluates our solution, and discusses how to extend ARCHIE to support virtual reality (VR) app testing respectively. Section 7 discusses the related work, and Section 8 concludes.

2 Observations of Real-world Testing Practices

An AR app has some common errors that are difficult to detect using conventional software engineering testing. To illustrate, consider an AR app for smartphones that seeks to warn distracted pedestrians when they might be walking into an intersection, such as the one shown in Figure 1.

The first kind of error are **classification errors**, as seen in Figure 1a. These are errors that result in incorrect label or image to be displayed. The causes of classification error can be due to the mismatch between the training and testing data sets used in the lab, and the actual visual inputs encountered in the wild. Some contributors to classification errors include environmental conditions such as poor light and viewing angle, as well as intra-class variations (such as different breeds of the same animal or different models of the same car). Because AR systems are subject to user movement, classification errors can also occur when the tester is moving too quickly to get a clear image of the target (such as Figure 1b). As it is impossible to train a classifier for every variation of inputs that it may receive, or to control exactly the movement and behavior of the user, classification errors are very common for AR systems.

The second kind of error are **placement errors**. Placement errors occur when the system has correctly classified the visual input, but places the label or image output in an inappropriate location on the display. This kind of error has the potential to be dangerous to testers, as they can affect the tester's safety in the real world. For example, Figure 1c shows how the stop sign can be correctly identified, but the warning label blocks the tester's view of the vehicle in the intersection.

The third kind of error are **resource limitation errors**. These are errors that are caused by insufficient resources (e.g. CPU, RAM, etc.) in the underlying system. It is possible for resource limitation errors to contribute to a classification or placement error. For example, the classification error shown in Figure 1b may be due to a slow sampling rate from the device camera, or insufficient resources to process the input in time.

The fourth kind of error are **style errors**. Style errors occur when the system classifies input and generates output correctly, and places the output in an appropriate location, but presents the output in an inappropriate way. For example, in Figure 1d, the warning label for the stop sign is placed above the intersection, but the white text is unreadable against the bright sky.

Environment	System Function	Sources
Lab	Label display	[69, 79, 80]
	Label placement	[54, 63, 75]
	UI comparison	[44, 64]
	Interaction	[39, 43, 48, 60, 74]
	Avatars	[52, 78, 84]
Wild	Label display	[40, 58, 77]
	Interaction	[45, 86]

Table 1: System functions targeted by surveyed AR systems

2.1 Survey of Feedback Methods

To understand how AR researchers handle testing errors, we conducted a survey of recent AR related papers. We first identified the high level system function being tested. We then categorized papers by whether the project was tested in the lab or in the wild, the length of the evaluation, the size of the test group, and primary method of collecting tester feedback. The results are presented in Table 1 and Figure 2, and summarized into the following key points.

- **Head-mounted displays (HMDs) are a popular testing**

platform. The majority of papers surveyed (62%) tested their platform either solely on HMDs or supplementing the HMD with a smartphone. A comparatively small number (24%) [40, 45, 60, 64, 86] tested solely on smartphones, while 14% [39, 63, 80] tested on a custom platform.

- **Questionnaires and interviews are the primary methods for gathering feedback from testers.** Of the papers surveyed, 81% relied solely on questionnaires, interviews, or both to collect tester feedback. The remaining papers [39, 63, 77, 80] substituted or supplemented these methods with manual coding, which is the manual review and labeling of data collected by the system.

- **Most tests are conducted in a single sitting with a small group of testers.** The bulk of papers surveyed (62%) conducted all tester activities within a single day. Of those papers, 69% conducted their evaluations with 25 or fewer testers.

- **A non-trivial portion of papers explicitly asked testers to compare user interface (UI) options**. More than one third of papers surveyed [44, 54, 64, 74, 75, 77, 78, 84] gave testers multiple UI options to evaluate, and took feedback specifically on which UI version was preferred for the target scenario.

2.2 Limitations of Existing Methods

From this survey, it is clear that questionnaires, interviews, or a combination of both are the method of choice for eliciting feedback from system testers. Unfortunately, these methods suffer from a number of limitations. The first limitation is that, while questionnaires and interviews can be very helpful in identifying whether the issues discussed above (classifier, placement, resource limitation, style) occurred, they are limited in their ability to identify the issues' underlying cause. For example, a tester might indicate in her interview that the application failed to identify the stop sign, but would not necessarily be able to tell you *why*. Was the tester moving too quickly for the camera to get a clear picture, such as in Figure 1b? Was she viewing the target from too severe of an angle for the classifier to recognize it? Was the light too low? Was the classifier too slow to process the image, and was overridden by earlier or later inputs? Using questionnaires and interviews alone cannot help researchers answer these questions.

The second limitation is that questionnaires and interviews provide no support for recreating error scenarios upon returning to the lab. Subjective, written responses from testers provide no actionable data to use as input back into the system. Any data sets, system inputs, or generated outputs must be recreated from scratch in order to debug the issues reported by testers.

The third limitation is that, when applications are testing multiple GUIs or implementation versions, questionnaires and interviews cannot guarantee that testing conditions for each version are consistent. Specifically, if a tester claims to prefer Interface A over Interface B, it is important to verify that the tester's preference is due to the differences in interfaces only, and not because of additional factors. For example, if the tester only evaluated Interface A during the day and Interface B at night, it is possible that the preference was impacted by time of day. Utilizing questionnaires and interviews to compare application versions or UI alternatives cannot provide this crucial insight into the testing conditions for each option.

Based on our observations of current testing methods for AR systems, and the inability of these methods to handle the testing issues discussed above, we designed ARCHIE to (1) collect tester feedback *in situ* to assess system performance and usability, (2) collect system input and resource consumption data to assist developer debugging efforts, and (3) facilitate multi-UI testing to support a more consistent tester experience. ARCHIE's lightweight design allows it to provide these features without impacting the function of the application under test. The following sections discuss the design of ARCHIE in greater detail.

3 System Design

3.1 Framework Architecture

As shown in Figure 3, the ARCHIE framework consists of three primary components: the **AR application** as developed by the external research team, the **ARCHIE annotator** helper application, and the **ARCHIE core library**. The annotator helper application is a lightweight client responsible only for forwarding input requests to the evaluator, and returning his or her response to the main system. The core library contains the bulk of our framework functionality. Any application wishing to utilize the ARCHIE framework may also implement modules called **profiles** to represent a distinct UI design point for testers to evaluate. A profile could contain logic for input, output, or core event recognition (such as gesture recognition or object classification).

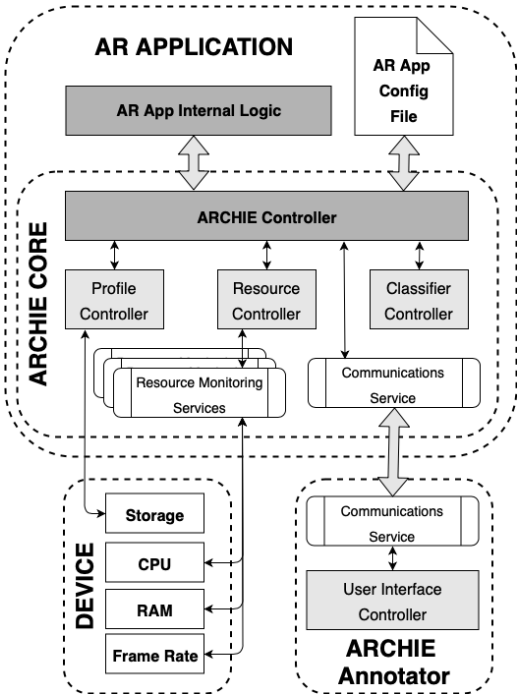


Figure 3: ARCHIE framework system architecture

An AR application-under-test (AUT) interactions with the core library by passing a **configuration file** on start-up to the ARCHIE Controller, the primary interface point between testable applications and the rest of the framework. The configuration file should minimally contain the list of input, output, and core event recognition profiles the framework is expected to utilize during the test instance. The ARCHIE Controller then instantiates these profiles, raising the appropriate events to connect to input sources, retrieve and process input data, and communicate the results to the user. Once the configuration file settings have been imported and the profile event handlers are initialized, the controller is responsible for managing the profile switching schedule, interfacing with the annotator application, and monitoring the appropriate device-level resources over the remainder of the test instance. When the evaluation is complete, ARCHIE generates user feedback and system state output logs for researchers to review.

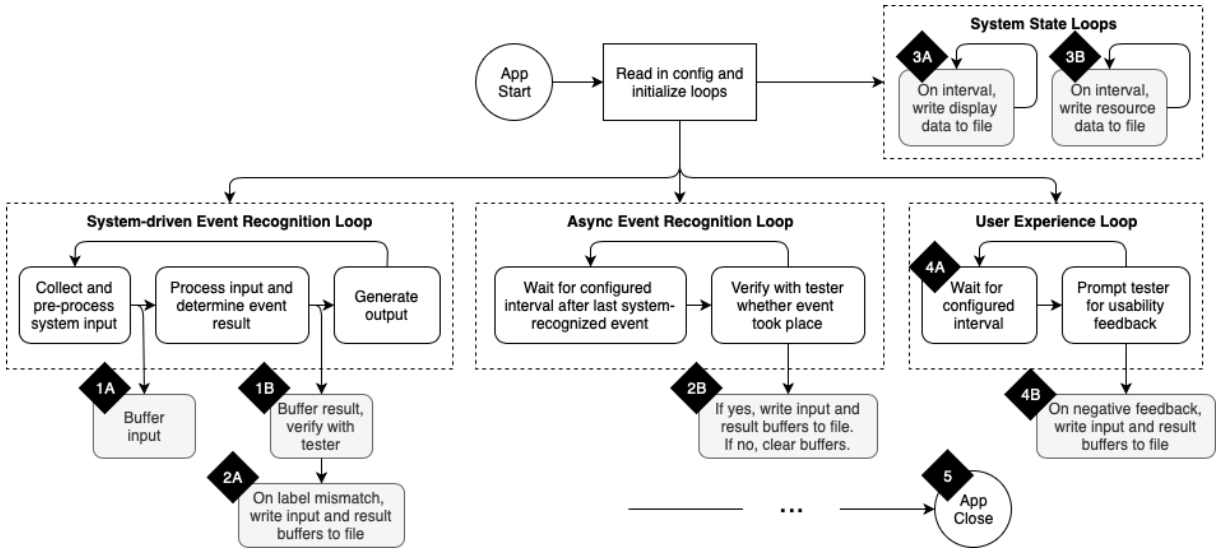


Figure 4: Runtime program loop with internal and user-driven data collection events.

Algorithm 1 Profile Scheduler

```
P, L_P, T, U, C ← configuration file
if t == 0 then
    for ρ in P do
        Q_t(ρ) = ℝ_t = α(ρ_t)
else
    while size(P) > L_P do
        for ρ in P do
            ℝ_t = α(ρ_t)
            Q_t(ρ) = Q_{t−1}(ρ) + (1/U)(ℝ_t − Q_{t−1}(ρ))
        ρ_min ← min(Q_t(P))
        ρ_max ← max(Q_t(P))
        if diff (ρ_max, ρ_min) ≥ C then
            remove(P, ρ_min)
        scheduleNextProfile(ρ_max)
    while executionTimeRemaining do
        roundRobin(P)
```

3.2 Profile Scheduling

The straightforward solution for choosing which profile to execute next would be to simply execute each profile in Round Robin order. However, this is not an effective use of a tester's time, as it could obligate her to spend a large portion of the trial interfacing with system features that may not be applicable or interesting to her. Instead, ARCHIE provides a customized scheduling algorithm that uses a tester's feedback data to prioritize the most liked profiles over time while still collecting the annotation and usability data in which researchers are interested.

To implement this algorithm, we utilized a customized solution for the Multi-Arm Bandit Problem as shown in Algorithm 1. We first define the reward function for a given data vector collected from profile (ρ) at instance (i) within time slice (t) as shown in Equation 1. This equation provides a numerical representation of how well the system's event recognition results matched the user's ground truth labels, plus the user's overall likability score at that given moment. Once all the instantaneous rewards have been calculated, the overall reward earned by the profile during the given time slice is calculated using Equation 2. It should be noted that both of these equations are simply default functions provided by ARCHIE, and that researchers wishing to use our framework may provide their own functions that are better tailored to their specific needs.

Param	Type	Description
P	Collection	The set of profiles, classifiers to be evaluated
L_P	Integer	P min-size constant
T	Integer	Time slice size constant
U	Float	Uncertainty constant
C	Float	Confidence threshold constant
t	Object	Current time slice of size T
ρ_t	Collection	Set of user response data collected while executing profile ρ over time slice t
$\mathbb{R}_t$	Integer	Reward received at time t
$Q_t(\rho)$	Float	Action-value function for profile ρ at time t

Table 2: Parameter list for profile scheduler algorithm. Values can be set in the configuration file for the application-under-test.

$$A(\rho_{ti}) = \begin{cases} 1 + usrScore_{ti} & sysResp_{ti} = usrResp_{ti} \\ usrScore_{ti} & sysResp_{ti} \neq usrResp_{ti} \end{cases} \quad (1)$$

$$\alpha(\rho_t) = \sum_{i=0}^{T} A(\rho_{ti}) \quad (2)$$

We then customize the traditional upper-confidence-bound solution of the Multi-Arm Bandit Problem, populating the parameters in Table 2 using entries retrieved from the application's configuration file, such as those shown in Figure 5a. The algorithm starts with the entire collection of profiles (P) as listed in the configuration file and calculates the action-value function (Q_t) for each profile (ρ) at the end of a given time slice (t). Since there is no guarantee that a user will always give the same feedback for a given profile, the action-value function is calculated in a non-stationary manner, giving greater weight to the most recent reward result for that profile. If the difference between the minimum and maximum action-values for a given time slice falls rises above a developer-defined threshold (C), the profile that earned the minimum score will be eliminated

Data Group	Fields	Collect	Commit
Event Recognition: Input	Timestamp; Preprocessed feature sets	1A	2A, 2B
Event Recognition: Results	Timestamp; Result; Preprocessing time; Processing time	1B	2A, 2B
System Stats: Display	Timestamp; Frame count; Total frame time; Average frame time	3A	3A, 5
System Stats: Resources	Timestamp; CPU consumption; RAM consumption; Process info	3B	3B, 5
UX Feedback	Timestamp; UX feature; User response	4A	4B

Table 3: Data collected by ARCHIE framework at various "collect" and "commit" points shown in Figure 4

```
"profile_scheduler" : {
        "lp" : 3, "u" : 0.5, "c" : 10,
        "time_slice" : {
                "denomination" : "Hours",
                "interval" : 1
        } }
```

(a)

```
"async_anno_settings" : {
        "denomination" : "Hours",
        "interval" : 1,
        "default_label" : "Stop Sign" }
```

(b)

Figure 5: Examples of configuration file settings to control (A) properties of the profile scheduling algorithm, and (B) asynchronous annotation data requests to the evaluator.

from the collection. Once the count of profiles remaining in the collection reaches a developer-defined limit (L_P), ARCHIE will execute the remaining profiles in Round Robin fashion.

The goal of this customized algorithm is to prioritize those profiles in which the tester shows the greatest interest, while simultaneously suppressing those profiles which the tester finds inferior. If a profile's reward is low enough that it falls below the acceptability threshold, then it is removed entirely and the tester does not have to interact with it any more. Furthermore, the algorithm does not require the system to converge on a single *best* answer. By providing a profile collection limit, the developer is able to designate the point at which the system will stop culling the collection and continue with Round Robin instead. By combining developer-provided parameters with user-driven reward functions, this scheduling algorithm helps ensure that testers are only presented with system options in which they are interested, and none in which they are not.

3.3 System State Data Collection

Based on the lessons learned in Section 2, we know how important it is to provide some way of identifying and debugging unintended event recognition behavior while conducting in-the-wild tests. ARCHIE helps researchers by buffering and storing collections of system state and user response data, clustered according to events in which researchers are interested, such as event recognition results which do not agree with the user's ground truth labels, or poor usability feedback.

Figure 4 demonstrates the on-going process of collecting application and system state data during a trial using the ARCHIE framework. Data samples extracted from the event recognition process are collected whenever a given phase of the pipeline is complete, while asynchronous user-focused data, such as usability feedback data, is collected according to the schedule set up by the researchers in the application's configuration file. The individual data points that are collected for each data group and the types of events that can trigger committing the buffer contents to file are described in Table 3. While each data group supports only a single event to trigger the collection of their respective data samples, any one of multiple commit events may trigger writing the buffered data to file. Once the data in the buffer has been committed, its contents are cleared and a new collection instance is started.

4 Testing with ARCHIE

4.1 Collecting Tester Feedback

Once the system has been initialized by the developer with the profiles that should be evaluated, the system can be handed off to testers. The ARCHIE framework requests input from the tester (1) to confirm when an event of interest is identified and (2) on configurable asynchronous schedules to collect general usability feedback and to see if any events of interest have been missed. (An example of a configuration file entry for asynchronous data collection is shown in Figure 5b.) An input request starts by playing an alert on the annotator device (such as a smartwatch, as shown in Figure 6), and presenting the question set by the developer (such as "Have you seen a stop sign in the last 10 minutes?" or "How are you liking the app so far?"). The user can toggle between positive and negative responses by tapping the center of the screen (Figure 6a). If the developer has included a request for additional input (such as rating how much the app is liked or disliked), the tester can drag her finger around the edge of the screen to provide her input (Figure 6b). Finally, the tester can select any applicable issues she encountered by checking them off from a grid of options, as shown in Figure 6c.

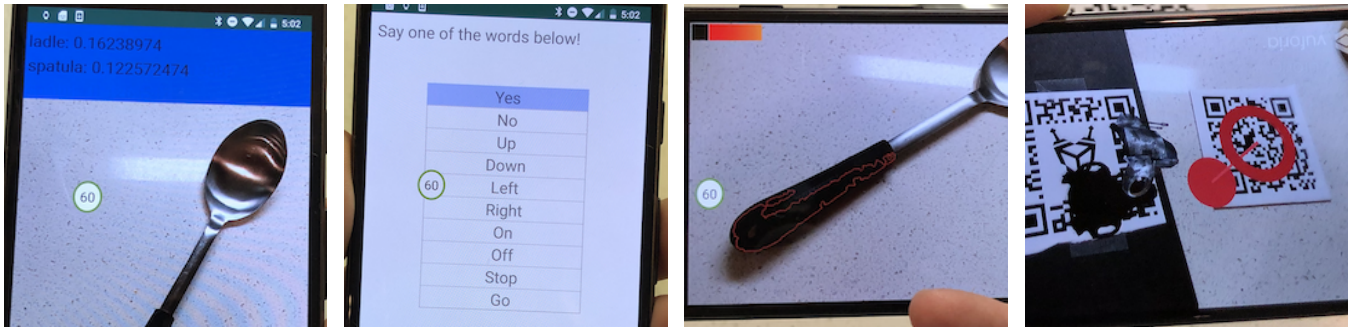
4.2 Debugging with Output Files

Once a tester has provided system and usability feedback, the output files generated by the ARCHIE framework can be used to recreate and debug the original scenario in order to determine the underlying cause. Consider again the system from Figure 1. The tester has completed the evaluation, and reported back to the research team that the system was a little slow in providing warnings of stop signs, and sometimes missed the stop signs altogether. To determine the underlying bug, the developers retrieve the data logs written by ARCHIE to the device file system. The data files (as described in Table 3) are organized into timestamped directories according to system event, such as an asynchronous event query ("async"), a system-generated event recognition ("system"), or a UX feedback request ("ux"). The directory name is then appended with the user's response (such as "falseNeg", "falsePos", or "slow"), and the appropriate timestamp. Inside each directory are the relevant files associated with that event, such as the inputs provided to the system and results that the system generated, as well as system display and resource consumption logs.

From these log files, it is possible to determine what unique combination of internal and external conditions caused the system to generate false positive or negative classifications, or contributed to slow performance. Input frames can be fed back into the system in the lab to determine where a classification error occurred or where an inappropriate label was placed. Reviewing the system log can help identify whether a spike in RAM and CPU consumption was anomalous or consistent with the rest of the trial. Exploring the results and system display logs in tandem can help determine whether the time an augmentation takes to display is due to the application's processing time or lag in the device as a whole.

(a) (b) (c)

Figure 6: Tester using ARCHIE annotator to provide feedback: (A) tapping to toggle between affirmative and negative reactions, (B) scrolling along the edge of the watch to rate her experience, (C) selecting issues she experienced from a grid of options.



(a) TF Classify (b) TF Speech (c) Color Blob Detector (d) DroneTracker

Figure 7: Sample screenshots of the ARCHIE evaluation candidate applications

5 Evaluations

We implemented and evaluated ARCHIE using a Google Pixel 2 running Android 8.1.0. We elected to test our framework using a smartphone because it could also double as an HMD when placed in an appropriate head mount. For testing purposes, we are primarily interested in the performance impacts when incorporating ARCHIE into different types of AR applications, to ensure that any negative feedback a tester may provide is due to the performance of the application itself, and not the influence of ARCHIE.

The design for the ARCHIE framework supports the collection of camera frames, video clips, audio clips, GPS and IMU data as sources of input for the data logs. Currently, only support for camera frame collection has been implemented because it was sufficient to meet the needs of our evaluations. The framework design also supports the collection of output camera frames containing the final augmented view displayed the user, but was not implemented due its library-specific (ARKit, Vuforia, etc.) nature. We anticipate adding this functionality to the framework in the future when more implementation-agnostic API support becomes available.

5.1 Evaluation Candidates

We used the following four case study applications to evaluate ARCHIE: TF Classify, TF Speech, Color Blob Detector, and DroneTracker. These applications were chosen to showcase how our framework integrates with three of the most popular computer vision support libraries in research and industry: TensorFlow [29], OpenCV [21], and Vuforia [36]. Each application was either taken from the publicly available sample applications published by the library developers or created as part of a tutorial, and represents a range of sensor and UI interactions.

"TF Classify" [30] (shortened to "TF-C", shown in Figure 7a) is an image classification application distributed and maintained by the TensorFlow development team which utilizes preview frames from the device's camera, categorizes the primary subject of the image, and populates a text box on the phone screen with the results in real-time. The camera preview is not manipulated directly; only the content of the text box is updated.

"TF Speech" [30] (shortened to "TF-S", shown in Figure 7b) is another application from TensorFlow which focuses on audio processing rather than image processing. Instead of using the camera, it captures audio snippets from the microphone and parses them in order to identify words it recognizes. The list of known words is displayed on the application screen, and when a recognized word is heard, the application will briefly highlight the corresponding box for that word on the screen.

"Color Blob Detector" [20] (shortened to "CBD", shown in Figure 7c) is an application distributed by the OpenCV development team which performs active color processing and boundary detection on a live camera preview. To detect color blobs, the user must press the device screen to select an area of the camera preview that he or she wishes to match. The application then segments that portion of the preview frame and identifies the primary color. For all subsequent preview frames until the user selects a new color blob, boundaries of areas within the preview that match the same primary color are detected and highlighted in real-time.

"DroneTracker" (shortened to "DT", shown in Figure 7d) is an application that we obtained from [13], as there are no formally recognized sample applications available for Vuforia. The DroneTracker app consists of a dual-target tracking system which displays

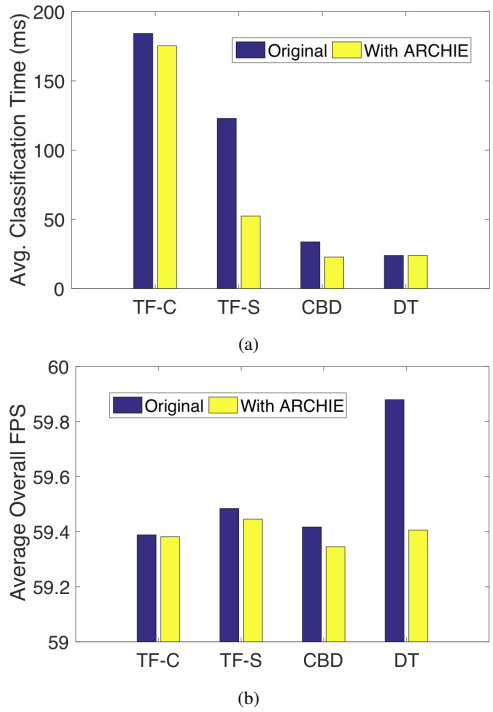


Figure 8: Impacts to (A) app-level classification time and (B) device-level FPS per case study application before and after incorporating ARCHIE framework

a small sci-fi drone that responds to and follows a target moved by the user around the screen in real-time.

5.2 Overhead of Incorporating ARCHIE

Because each of the evaluation candidate applications differs in nature from its fellows, each one had to be tested differently. TF-C was placed on the edge of a desk, pointed downward at a rug with a design detailed enough to be confused between two of the classifier's known classes. TF-S was laid on the desk next to an active radio. CBD was placed on the edge of the desk and pointed toward the rug as well, with a single screen touch by the tester to active the color blob boundary recognition logic. DT was placed on the edge of a stack of books, overlooking the two targets. Each application was then allowed to run for five trials of five minutes each, while ARCHIE collected the appropriate system state and application output data.

Figures 8 and 9 reflect the impact to UI and system performance respectively when incorporating the ARCHIE framework into an existing application. While examining impacts to CPU and RAM consumption is expected, we also included the UI metrics of display FPS and time to classify in order to demonstrate how ARCHIE might impact the user experience. If refactoring an application resulted in lowered FPS or a dramatically increased classification time, then developers could expect users to experience a decreased quality of overall system performance due to delayed or inaccurate system responses. However, no ARCHIE case studies showed significant impacts to FPS or classification time.

Of the performance metrics evaluated, the only case in which performance showed a significant hit was when ARCHIE was added to the TF-Speech app. In this case, user-space CPU utilization went up from about 28% to about 40%. We believe that the reason for this is due to the extremely simple nature of the original app - meaning that adding ARCHIE introduced a non-trivial amount of additional processing. The other apps utilize the camera and some

	System Function	Tester Resp. Method(s)	Num. of Eval. Sess.
HMD	Display	Questionnaire + Interview	1 [51]
		Unclear [73]	
		Questionnaire	3 [47]
	UI comparison	Questionnaire + Interview	1 [76]
	Interaction	Questionnaire + Interview	1 [57, 81]
		Questionnaire	Unclear [49]
		Interview	1 [68]
	Avatar	Questionnaire	1 [55]
Smart phone	Interaction	Manual coding	3 [38]
Both	Display	Interview	1 [53]
	Interaction	Questionnaire	1 [71]

Table 4: Expanded survey of contemporary VR systems

degree of image processing (operations which already require a good deal of resource consumption), such that introducing the additional processing for ARCHIE was negligible.

5.3 Cost of Switching Profiles

Providing context-switching ability for a collection of profiles and classifiers is a key contribution of the ARCHIE framework. However, this feature provides no benefit if the cost of context-switching is high. Therefore, we ran a series of tests evaluating the CPU and RAM consumption overhead of context-switching for collections of profiles ranging in size from one to ten, as reflected in Figure 10. TF-C was used for this evaluation. Five configurations were created with various numbers of profiles (one, three, five, seven and ten) for ARCHIE to manage, with each profile representing a distinct instance of the core image recognition classifier. Each profile was allowed to execute for one minute before switching to the next; each configuration was executed five times for 25 minutes per execution.

During testing, however, the camera interface would occasionally freeze, causing a significant drop in resource consumption until the next context switch, at which point the camera preview would refresh and processing began again. Over all of the trials for all configurations, only 5% of the records were affected and removed. The results in Figure 10 reflect the aggregated results for the remaining 95%. At most, resource consumption fluctuates by 2% when considered across five trials of 25 minutes each for each of the five test cases. This leads us to conclude that ARCHIE provides no significant overhead when switching between profiles during evaluations.

6 Additional Discussion

Even though ARCHIE was originally designed with AR applications in mind, it could easily be adapted to work with virtual reality (VR) applications. We expanded our survey from Section 2 to examine VR systems as well (results in Table 4), and observed the following: (1) all evaluations were conducted in the lab on some combination of smartphones and HMDs, (2) all papers but one [38] used questionnaires and interviews to collect user feedback, and (3) three papers [51, 76, 81] conducted their own explicit UI comparisons. Based on this, we believe ARCHIE has real potential for use in testing VR applications as they move out of the lab and into the wild. To accommodate this shift, we would need support for new profiles to monitor motion sickness, and data collection from new sensors such as galvanic skin response.

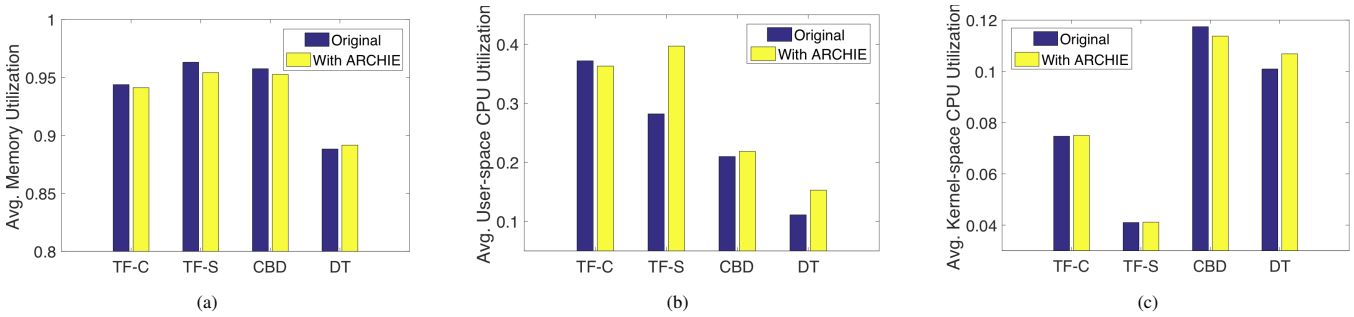


Figure 9: Consumption of (A) memory, (B) user-space CPU, and (C) kernel-space CPU before and after incorporating ARCHIE framework into application code base. Results averaged over five trials each.

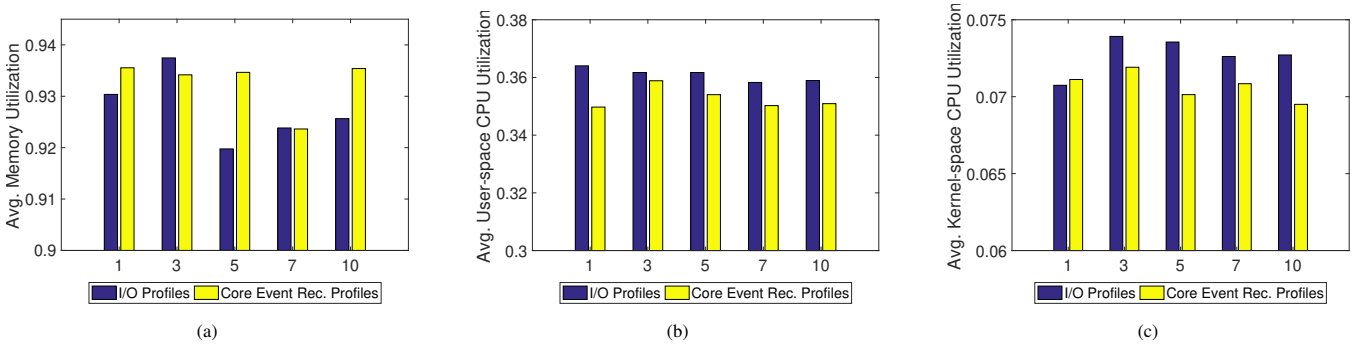


Figure 10: Impacts to consumption of (A) memory, (B) user-space CPU, and (C) kernel-space CPU when switching between differently sized collections of profiles at run-time. Results averaged over five trials each.

7 Related Work

Many tools and frameworks exist to help developers test their mobile AR systems. These testing platforms serve a number of goals, many of which can be supplemented with ARCHIE.

Commercial testing tools. The increase in commercially available AR hardware and development environments brings with it an increase in commercial testing tools as well. Oculus provides official guidelines for performance optimization [35] as well as an array of performance monitoring and debugging tools [6, 18, 23, 25] for applications built for the Rift headset. Microsoft provides its own official guidelines for testing applications built for the HoloLens headset [31], in addition to a selection of emulators and simulators [22, 33, 34] for testing applications with pre-determined inputs. Similarly, Apple's latest version of ARKit ships with a testing tool called "Reality Composer" [28], which allows developers to record and replay system inputs for testing. There are also many general-purpose tools available for graphics and systems-level profiling, such as NVIDIA's "Nsight" suite of tools [17], the "apitrace" tool for OpenGL [1], Android's system trace CLI tool [5], and Windows profilers for system events [7] and DirectX applications [26]. While all of these tools can be helpful during development and initial testing stages, they are generally suitable only for in-lab testing, as their operations are quite resource-intensive. ARCHIE can supplement these efforts by providing additional lightweight support when an application has passed code quality testing in the lab and is ready for user evaluations in the wild.

Quality of Experience (QoE) testing. In addition to commercial tools, there has been research aimed at quantifying the quality of experience (QoE) that a tester might have while using a given application. Papers such as [42, 62, 83] propose frameworks to identify potential system bottlenecks that could adversely affect user experience. However, what these systems provide is raw data output; it is up to the researcher to determine the point at which the perceived latency for an application operation becomes insufferable to the user. ARCHIE can supplement these testing frameworks by providing temporally-aligned user feedback data to identify exactly when changes in system performance start to impact the user.

Cross-platform testing. Other general-purpose testing tools focus on supporting heterogeneous hardware platforms [72], software platforms [67], and embedded devices [56]. However, these testing platforms do not address the user experience component of the testing process. ARCHIE can supplement cross-platform testing frameworks such as these by presenting users with different implementations of a given interface on a given device in order to identify whether any issues or preferences exist from a user's perspective when porting applications to new environments.

8 Conclusions

In this paper, we presented ARCHIE, the Augmented Reality Computer-Human Interaction Evaluator framework. We demonstrated the prevalence and limitations of using only questionnaires and interviews for testing augmented reality applications in the wild, and showed how our feedback framework can assist developers in identifying and debugging UX issues with negligible impact to system performance. In the future, we would like to expand ARCHIE to include more formal usability testing.

Acknowledgments

This research was supported in part by NSF grants 1618398, 1814745, and 2002434.

References

[1] apitrace. https://apitrace.github.io. Accessed: 2019-10-22.

[2] Apple developer — arkit. https://developer.apple.com/arkit/. Accessed: 2018-09-04.

[3] Ar ruler app - tape measure and camera to plan. https://play.google.com/store/apps/details?id=com.grymala.aruler. Accessed: 2019-10-15.

[4] Camtoplan - ar measurement / tape measure. https://play.google.com/store/apps/details?id=com.tasmanic.camtoplan. Accessed: 2019-10-15.

[5] Capture a system trace on the command line — android developers. https://developer.android.com/studio/profile/systrace/commandline. Accessed: 2019-10-22.

[6] Compositor mirror — oculus documentation. https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-compositor-mirror/. Accessed: 2019-10-22.

[7] Event tracing - windows applications — microsoft docs. https://docs.microsoft.com/en-us/windows/win32/etw/event-tracing-portal. Accessed: 2019-10-22.

[8] Glass. https://www.x.company/glass/. Accessed: 2019-10-15.

[9] Google cardboard - google vr. https://arvr.google.com/cardboard/. Accessed: 2019-10-15.

[10] Google developers — arcore. https://developers.google.com/ar/. Accessed: 2018-09-04.

[11] Google translate. https://play.google.com/store/apps/details?id=com.google.android.apps.translatehl=en. Accessed: 2019-10-15.

[12] Ingress prime. https://play.google.com/store/apps/details?id=com.nianticproject.ingress. Accessed: 2018-09-04.

[13] Knights of unity — vuforia ar tutorial. http://blog.theknightsofunity.com/unity-vuforia-guide/. Accessed: 2018-09-20.

[14] Lg nexus 5 — newegg. https://www.newegg.com/p/23B-000F-00110. Accessed: 2019-10-15.

[15] Microsoft hololens — mixed reality for business. https://www.microsoft.com/en-us/hololens. Accessed: 2019-10-15.

[16] Mondly. https://play.google.com/store/apps/details?id=com.atistudios.mondly.languages. Accessed: 2019-10-15.

[17] Nvidia nsight systems — nvidia developer. https://developer.nvidia.com/nsight-systems. Accessed: 2019-10-22.

[18] Oculus debug tool — oculus documentation. https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-debug-tool/. Accessed: 2019-10-22.

[19] Oculus rift. https://www.oculus.com/rift. Accessed: 2019-10-15.

[20] Opencv - android - color blob detection. https://github.com/opencv/opencv/tree/master/ samples/android/color-blob-detection. Accessed: 2018-09-20.

[21] Opencv library. https://opencv.org. Accessed: 2018-09-26.

[22] Perception simulation - mixed reality — microsoft docs. https://docs.microsoft.com/en-us/windows/mixed-reality/perception-simulation. Accessed: 2019-10-22.

[23] Performance heads-up display — oculus documentation. https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-hud/. Accessed: 2019-10-22.

[24] Pokemon go. www.pokemon.com/us/pokemon-video-games/pokemon-go/. Accessed: 2018-09-04.

[25] Profiler tool reference — unreal engine documentation. https://docs.unrealengine.com/en-US/Engine/Performance/Profiler/index.html. Accessed: 2019-10-22.

[26] Profiling directx apps - windows applications — microsoft docs. https://docs.microsoft.com/en-us/windows/win32/direct2d/profiling-directx-applications. Accessed: 2019-10-22.

[27] Quiver - 3d coloring app. https://play.google.com/store/apps/details?id=com.puteko.colarmix. Accessed: 2018-09-04.

[28] Reality composer - augmented reality - apple developer. https://developer.apple.com/augmented-reality/reality-composer/. Accessed: 2019-10-22.

[29] Tensorflow — open-source machine learning framework. https://www.tensorflow.org. Accessed: 2018-09-04.

[30] Tensorflow android camera demo. https://github.com/tensorflow/tensorflow/tree/ master/tensorflow/examples/android. Accessed: 2018-09-20.

[31] Testing your app on hololens - mixed reality — microsoft docs. https://docs.microsoft.com/en-us/windows/mixed-reality/testing-your-app-on-hololens. Accessed: 2019-10-22.

[32] Unity3d game engine. www.unity3d.com. Accessed: 2018-09-04.

[33] Using the hololens emulator - mixed reality — microsoft docs. https://docs.microsoft.com/en-us/windows/mixed-reality/using-the-hololens-emulator. Accessed: 2019-10-22.

[34] Using the windows mixed reality simulator - mixed reality — microsoft docs. https://docs.microsoft.com/en-us/windows/mixed-reality/using-the-windows-mixed-reality-simulator. Accessed: 2019-10-22.

[35] Vr performance optimization guide — oculus documentation. https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-performance-opt-guide/. Accessed: 2019-10-22.

[36] Vuforia — augmented reality framework. https://www.vuforia.com. Accessed: 2018-09-04.

[37] What is google maps ar navigation and how do you use it? https://www.pocket-lint.com/apps/news/google/147956-what-is-google-maps-ar-navigation-and-how-do-you-use-it. Accessed: 2019-10-15.

[38] K. Ahuja, R. Islam, V. Parashar, K. Dey, C. Harrison, and M. Goel. Eyespyvr: Interactive eye sensing using off-the-shelf, smartphone-based vr headsets. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(2):57, 2018.

[39] F. Bork, R. Barmaki, U. Eck, K. Yu, C. Sandor, and N. Navab. Empirical study of non-reversing magic mirrors for augmented reality anatomy learning. In *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 169–176. IEEE, 2017.

[40] C. Cao, Z. Li, P. Zhou, and M. Li. Amateur: Augmented reality based vehicle navigation system. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):155, 2018.

[41] T. P. Caudell and D. W. Mizell. Augmented reality: An application of heads-up display technology to manual manufacturing processes. In *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on*, vol. 2, pp. 659–669. IEEE, 1992.

[42] Q. A. Chen, H. Luo, S. Rosen, Z. M. Mao, K. Iyer, J. Hui, K. Sontineni, and K. Lau. Qoe doctor: Diagnosing mobile app qoe with automated ui control and cross-layer analysis. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pp. 151–164. ACM, 2014.

[43] M. A. Cidota, P. J. Bank, P. Ouwehand, and S. G. Lukosch. Assessing upper extremity motor dysfunction using an augmented reality game. In *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 144–154. IEEE, 2017.

[44] C. Diaz, M. Walker, D. A. Szafir, and D. Szafir. Designing for depth perceptions in augmented reality. In *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 111–122. IEEE, 2017.

[45] D. Harborth and S. Pape. Exploring the hype: Investigating technology acceptance factors of pokémon go. In *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 155–168. IEEE, 2017.

[46] M. K. Holden. Virtual environments for motor rehabilitation. *Cyberpsychology & behavior*, 8(3):187–211, 2005.

[47] O. Hurd, S. Kurniawan, and M. Teodorescu. Virtual reality video game paired with physical monocular blurring as accessible therapy for amblyopia. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 492–499. IEEE, 2019.

[48] J. Jung, H. Lee, J. Choi, A. Nanda, U. Gruenefeld, T. Stratmann, and W. Heuten. Ensuring safety in augmented reality from trade-off between immersion and situation awareness. In *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 70–79. IEEE, 2018.

[49] F. Kern, C. Winter, D. Gall, I. Käthner, P. Pauli, and M. E. Latoschik. Immersive virtual reality and gamification within procedurally generated environments to increase motivation during gait rehabilitation. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 500–509. IEEE, 2019.

[50] M. Kesim and Y. Ozarslan. Augmented reality in education: current technologies and the potential for education. *Procedia-Social and Behavioral Sciences*, 47:297–302, 2012.

[51] T. Keskinen, V. Mäkelä, P. Kallioniemi, J. Hakulinen, J. Karhu, K. Ronkainen, J. Mäkelä, and M. Turunen. The effect of camera height, actor behavior, and viewer position on the user experience of

360 videos, 2019.

[52] K. Kim, L. Boelling, S. Haesler, J. Bailenson, G. Bruder, and G. F. Welch. Does a digital assistant need a body? the influence of visual embodiment and social behavior on the perception of intelligent virtual agents in ar. In *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 105–114. IEEE, 2018.

[53] W. Kim, K. T. W. Choo, Y. Lee, A. Misra, and R. K. Balan. Empath-d: Vr-based empathetic app design for accessibility. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 123–135. ACM, 2018.

[54] E. M. Klose, N. A. Mack, J. Hegenberg, and L. Schmidt. Text presentation for augmented reality applications in dual-task situations. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 636–644. IEEE, 2019.

[55] G. A. Lee, T. Teo, S. Kim, and M. Billinghurst. A user study on mr remote collaboration using live 360 video. In *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 153–164. IEEE, 2018.

[56] Y.-D. Lin, E. T.-H. Chu, S.-C. Yu, and Y.-C. Lai. Improving the accuracy of automated gui testing for embedded systems. *IEEE software*, 31(1):39–45, 2014.

[57] F. Lu, D. Yu, H.-N. Liang, W. Chen, K. Papangelis, and N. M. Ali. Evaluating engagement level and analytical support of interactive visualizations in virtual reality environments. In *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 143–152. IEEE, 2018.

[58] R. McNaney, I. Poliakov, J. Vines, M. Balaam, P. Zhang, and P. Olivier. Lapp: a speech loudness application for people with parkinson's on google glass. In *Proceedings of the 33rd annual ACM conference on Human Factors in Computing Systems*, pp. 497–500. ACM, 2015.

[59] A. S. Merians, D. Jack, R. Boian, M. Tremaine, G. C. Burdea, S. V. Adamovich, M. Recce, and H. Poizner. Virtual reality–augmented rehabilitation for patients following stroke. *Physical therapy*, 82(9):898–915, 2002.

[60] J. Müller, T. Langlotz, and H. Regenbrecht. Panovc: Pervasive telepresence using mobile phones. In *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 1–10. IEEE, 2016.

[61] A. Y. Nee, S. Ong, G. Chryssolouris, and D. Mourtzis. Augmented reality applications in design and manufacturing. *CIRP Annals-manufacturing technology*, 61(2):657–679, 2012.

[62] A. Nikravesh, H. Yao, S. Xu, D. Choffnes, and Z. M. Mao. Mobilyzer: An open platform for controllable mobile network measurements. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 389–404. ACM, 2015.

[63] Y. Park, S. Yun, and K.-H. Kim. When iot met augmented reality: Visualizing the source of the wireless signal in ar view. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 117–129. ACM, 2019.

[64] S. Prakash, A. Bahremand, L. D. Nguyen, and R. LiKamWa. Gleam: An illumination estimation framework for real-time photorealistic augmented reality on mobile devices. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 142–154. ACM, 2019.

[65] H. Regenbrecht, G. Baratoff, and W. Wilke. Augmented reality projects in the automotive and aerospace industries. *IEEE Computer Graphics and Applications*, 25(6):48–56, 2005.

[66] L. B. Rosenberg. The use of virtual fixtures as perceptual overlays to enhance operator performance in remote environments. Technical report, Stanford Univ Ca Center for Design Research, 1992.

[67] S. Roy Choudhary, M. R. Prasad, and A. Orso. X-pert: a web application testing tool for cross-browser inconsistency detection. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pp. 417–420. ACM, 2014.

[68] L. Shapira, J. Amores, and X. Benavides. Tactilevr: integrating physical toys into learn and play virtual reality experiences. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 100–106. IEEE, 2016.

[69] J. Shu, S. Kosta, R. Zheng, and P. Hui. Talk2me: A framework for device-to-device augmented reality social network. In *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 1–10. IEEE, 2018.

[70] J. H. Shuhaiber. Augmented reality in surgery. *Archives of surgery*, 139(2):170–174, 2004.

[71] M. Speicher, S. Cucerca, and A. Krüger. Vrshop: A mobile interactive virtual reality shopping environment combining the benefits of on-and offline shopping. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):102, 2017.

[72] C. Tao and J. Gao. Modeling mobile application test platform and environment: testing criteria and complexity analysis. In *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, pp. 28–33. ACM, 2014.

[73] J.-D. Taupiac, N. Rodriguez, O. Strauss, and M. Rabier. Ad-hoc study on soldiers calibration procedure in virtual reality. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces-proceedings*, 2019.

[74] C. Templeman, F. J. Ordóñez, A. Symes, and D. Roggen. Exploring glass as a novel method for hands-free data entry in flexible cystoscopy. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 581–592. ACM, 2016.

[75] C. Trepkowski, D. Eibich, J. Maiero, A. Marquardt, E. Kruijff, and S. Feiner. The effect of narrow field of view and information density on visual search performance in augmented reality. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 575–584. IEEE, 2019.

[76] K. Vasylevska, H. Yoo, T. Akhavan, and H. Kaufmann. Towards eye-friendly vr: How bright should it be? In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 566–574. IEEE, 2019.

[77] C. Voss, P. Washington, N. Haber, A. Kline, J. Daniels, A. Fazel, T. De, B. McCarthy, C. Feinstein, T. Winograd, et al. Superpower glass: delivering unobtrusive real-time social cues in wearable systems. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pp. 1218–1226. ACM, 2016.

[78] M. E. Walker, D. Szafir, and I. Rae. The influence of size in augmented reality telepresence avatars. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 538–546. IEEE, 2019.

[79] S. Werrlich, A. Daniel, A. Ginger, P.-A. Nguyen, and G. Notni. Comparing hmd-based and paper-based training. In *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 134–142. IEEE, 2018.

[80] C. A. Wiesner, M. Ruf, D. Sirim, and G. Klinker. 3d-frc: Depiction of the future road course in the head-up-display. In *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 136–143. IEEE, 2017.

[81] M. Wirth, S. Gradl, J. Sembdner, S. Kuhrt, and B. M. Eskofier. Evaluation of interaction techniques for a virtual reality reading room in diagnostic radiology. In *The 31st Annual ACM Symposium on User Interface Software and Technology*, pp. 867–876. ACM, 2018.

[82] H.-K. Wu, S. W.-Y. Lee, H.-Y. Chang, and J.-C. Liang. Current status, opportunities and challenges of augmented reality in education. *Computers & education*, 62:41–49, 2013.

[83] L. Xue, C. Qian, and X. Luo. Androidperf: A cross-layer profiling system for android applications. In *Quality of Service (IWQoS), 2015 IEEE 23rd International Symposium on*, pp. 115–124. IEEE, 2015.

[84] B. Yoon, H.-i. Kim, G. A. Lee, M. Billinqhurst, and W. Woo. The effect of avatar appearance on social presence in an augmented reality remote collaboration. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 547–556. IEEE, 2019.

[85] S. C.-Y. Yuen, G. Yaoyuneyong, and E. Johnson. Augmented reality: An overview and five directions for ar in education. *Journal of Educational Technology Development and Exchange (JETDE)*, 4(1):11, 2011.

[86] J. Zhang, A. Ogan, T.-C. Liu, Y.-T. Sung, and K.-E. Chang. The influence of using augmented reality on textbook support for learners of different learning styles. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 107–114. IEEE, 2016.