Leveraging Edge Computing for Mobile Augmented Reality

Sarah M. Lehman and Chiu C. Tan

1 Introduction

Augmented reality (AR) is the insertion of virtual content into a view of the real world based on environmental context. The virtual content may take the form of characters, textures, or labels, and is dynamically updated based on the user's location, viewing direction, behavior, and other contextual data. The virtual content utilized in AR systems is traditionally visual in nature, but may be supplemented with auditory and haptic feedback as well.

AR systems differ from virtual reality (VR) and mixed reality (MR) systems in several key ways. Virtual reality systems seek to replace the physical world completely with virtual content, while augmented reality inserts virtual content into the real world. The range from augmented to mixed reality is a sliding scale; an application which simply places a virtual character on a tabletop would be considered augmented reality, whereas an application which identifies a plethora of grocery and kitchen items and displays possible recipes could be considered mixed reality. While the difference between AR and MR systems is one of degrees, for the purposes of this chapter, we will focus on augmented reality systems.

Augmented reality has made great impacts in recent years in both research and commercial domains, such as education [3, 10, 17], tourism [11, 13, 25, 56], entertainment [46, 58], healthcare [7, 20, 24, 27, 41], and manufacturing [19, 38, 57]. Contemporary platforms for AR systems are as varied as the systems themselves; smartphones and tablets are popular tools for AR, but alternative platforms such as

S. M. Lehman (✉) · C. C. Tan
Temple University, Philadelphia, PA, USA
e-mail: smlehman@temple.edu; cctan@temple.edu

W. Chang, J. Wu (eds.), *Fog/Edge Computing For Security, Privacy, and Applications*, Advances in Information Security 83,
https://doi.org/10.1007/978-3-030-57328-7_13

head-mounted displays (HMDs) like the Microsoft HoloLens,[1] and Google Glass,[2] and augmented windshields for smart vehicles[3] are also gaining popularity. This wider availability of low-cost AR-enabled hardware has encouraged market growth for AR systems with global profits of 6.1 billion USD in 2016. As the cost of hardware decreases and the availability of development tools increases, this profit is predicted to grow to an estimated 18.8 billion USD in 2020.[4]

As AR systems increase in prevalence and popularity, the security risks of such systems must also be considered. There are several points of an AR system that are vulnerable to different kinds of attacks. Attackers could target user authentication or input data collection, compromising how the system extracts and prepares data for processing. They could also target the actual processing of the data itself by constructing system inputs in such a way that forces the application to produce a result that is both unexpected to the developer and beneficial to the attacker. Finally, the attacker could target the output of the application, either to interfere with outputs from other applications or to compromise the user's interactions with their environment. These attacks can be mitigated when the AR system is moved to the edge, thanks to expanded processing resources and data storage capabilities on edge servers. However, when moving operations to the edge, an AR system must first and foremost be able to guarantee low end-to-end latency in order to preserve a high quality user experience. Understanding how user experience can be measured, which system metrics impact it, and to what degree, is key to designing an effective AR system. Ensuring that a system meets these requirements while also providing a high level of security is even more important.

The rest of the chapter is organized as follows. First, we provide an background of augmented reality research and development, introducing the AR processing pipeline. Next, we discuss the security implications for AR systems at large, including examples of specific vulnerabilities within each phase of the AR processing pipeline. Then, we give a brief overview of research efforts focused on moving AR systems to the network edge as well as the metrics for evaluating them, followed by an exploration of the security issues for AR at the edge. We wrap up with a discussion of open problems in this research space, and present our conclusions.

2 Background of AR Systems

The fundamental requirement for an augmented reality system is the ability to sense and respond to changes in the user's environment and behavior, typically via the

[1]https://www.microsoft.com/en-us/hololens.

[2]https://www.google.com/glass/start/.

[3]https://www.motorauthority.com/news/1120806_hyundai-turns-the-windshield-into-an-augmented-reality-nav-system.

[4]https://www.statista.com/statistics/591181/global-augmented-virtual-reality-market-size/.

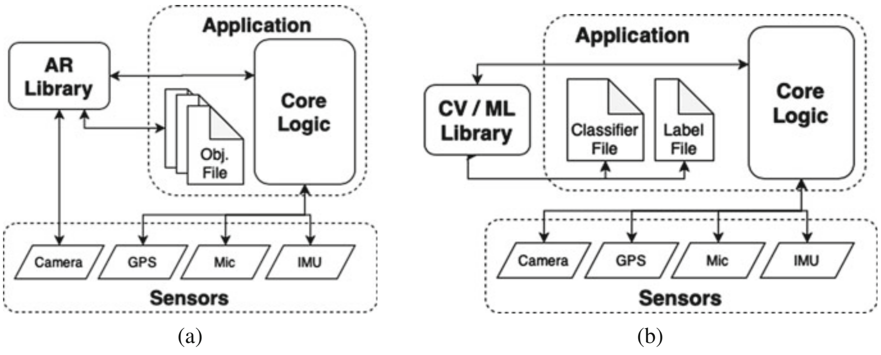


Fig. 1 App architectures when using (**a**) AR libraries or (**b**) computer vision/machine learning libraries

delivery of visual or auditory cues to the user. Because of this, AR systems rely on live data feeds from various sensors, such as the camera, microphone, GPS, and more. The exact implementation for processing these data feeds depends on the type of underlying library being utilized. For systems built with specialized AR libraries like ARCore[5] and Vuforia[6] (Fig. 1a), the library acts as an intermediary between the core application logic and the target device sensors (such as the camera). The developer is responsible for initializing the library instance with object files representing the targets to be recognized, and then subscribing to event listeners that will be raised when a target is located. This ease of integration comes with a price, however, as developers are limited to only those functions which the library publisher explicitly supports. On the other hand, developers using more general-purpose libraries such as OpenCV[7] and TensorFlow[8] (Fig. 1b) can exert more fine-grained control over the functionality of their systems, executing any operations for which they are able to collect data and train models. However, the downside is that these operations are remarkably resource-intensive, making them difficult to integrate efficiently into resource-constrained systems.

The pipeline of operations for an AR system is split in three major phases, as demonstrated by Fig. 2. The first phase is the **input phase**, during which the AR device is responsible for collecting data from the on-board sensors, including visual data from the camera, audio from the microphone, and environmental data such as location, ambient light levels, and more. The second phase is the **transformation phase**, during which the data that has been collected is utilized in a computer vision or machine learning operation, such as object recognition. This phase includes preprocessing the data to get it ready for the operation, performing the operation, and

[5] https://developers.google.com/ar/.

[6] https://www.vuforia.com.

[7] https://opencv.org.

[8] https://www.tensorflow.org.

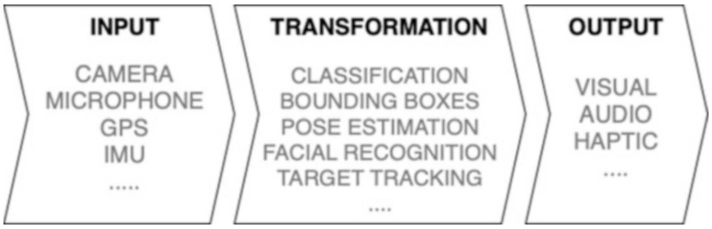


Fig. 2 AR processing pipeline

then aggregating the results as appropriate. The final phase is the **output phase**, during which the transformation results are used to generate and display virtual content, such as labels, sounds, or haptic feedback.

There are many types of computer vision and machine learning operations that can be utilized during the transformation phase of an AR system. **Classification** refers to the act of determining the subject of a given image. The drawing of **bounding boxes** refers to, not only identifying objects of interest within an image, but denoting the space they embody by drawing a box around it. **Pose estimation** looks at the posture of an entity within an image, such as a human subject, and recreates the posture of the underlying skeleton. **Face detection** identifies features of human faces within a given image in order to judge location, expression, gaze direction, and more. **Target tracking** refers to the monitoring of an actor's position as they move through a scene, including behind other objects or actors in that scene. All of these operations and more can be utilized by AR systems to identify users' context and place responsive content into the environment.

3 Security Risks of AR

Due to the richness of the data collected by AR applications, they are subject to a number of security and privacy vulnerabilities that non-AR applications do not face. These vulnerabilities stem from the always-on nature of the microphone and camera sensors in combination with atypical input methods, such as voice commands or physical gestures, and environmental or context data, such as GPS location and accelerometer readings.

Each phase of the AR pipeline is subject to different types of attack. Attacks during the input phase can target user authentication processes or collection of data from system sensors or remote servers. Attacks during the transformation phase can focus on how this collected data is pre-processed, manipulate or compromise the actual computer vision operation being performed, or contaminate the returned result. Finally, attacks during the output phase can target the content being generated or the placement and styling of that content in the system display. We will look at examples of different kinds of attacks in each of these phases, and wrap up with a quick discussion of other concerns in AR security beyond the processing pipeline.

3.1 Input Phase Attacks

3.1.1 User Authentication

AR systems rely on gestures and voice commands to maintain an immersive experience in a system without traditional input operations, such as typing on a keyboard or clicking with a mouse. Systems utilizing these non-traditional input methods rely on user authentication to identify valid inputs and filter out environmental noise, ensuring that subsequent system operations are germane to the current user. Proper user authentication can prevent both the creation of incorrect or unnecessary virtual content, as well as incorrect blocking of desired content.

For voice commands, a third-party attacker can control a victim's system through a number of methods. For instance, inaudible or unintelligible commands can be played through speakers; alternatively, a specific user's voice can be recorded or synthesized from recordings and then played back. The typical defense against attacks like this is called liveness detection, in which the presence of a valid human user is determined before executing a command. One particular solution presented in [52] suggested the use of contact microphones in AR headsets to compare waves travelling through the speaker's head with those received through the air at the headset microphone.

Physical gestures are also vulnerable in AR systems. Because gestures are readily visible by observers, simply making a given gesture is insufficient for user authentication. The exact manner in which the gesture is completed must be analyzed to distinguish between users. An example of this is in [53], in which the IMU sensors of an AR headset are used to monitor and analyze a user's walking patterns. The user's gait is then utilized as a unique identifier for authentication. Gestures can also be used when pairing devices among multiple users [18, 54], though extra steps must be taken to prevent man-in-the-middle attacks during the pairing process.

However, voice- and gesture-based authentication are only really viable with wearable sensors. For more pervasive AR environments, such as augmented windscreens in smart cars, the display is part of the environment and accessible to all persons in the immediate area. Therefore, performing authentication becomes a more delicate process than attempting to validate a single user in isolation.

3.1.2 Data Collection

After the AR system has authenticated the user, it can start collecting data. However, once data is collected by an application, any control that the user has over that data is gone. Therefore, limiting a malicious application's ability to collect data in the first place becomes a major concern. For example, a retail application which places virtual furniture in the user's home would have a justifiable need to recognize planes, but might also recognize text in the background without indicating as such to the user. The primary method of defending against this is to rely on the underlying operating system or commercial libraries to enforce data access limitations. For example, instead of allowing applications to subscribe directly to device sensors such as the camera and perform operations on the raw frame, the operating system could expose access to abstracted objects such as skeletons, faces, and hands [22]. In this way, applications have access to the objects and associated meta data that they truly need without accessing raw frames from the camera feed. Similarly, wrappers could be implemented for popular computer vision libraries such as OpenCV to enforce a customizable level of sanitation upon frames being processed by a given application [23].

There are also situations in which data access is based, not on the operation being performed, but on the user's environment and the objects therein. Data collection in this situation can be managed by incorporating access control policies directly into the user's environment. One potential approach is to modify a device's camera subsystem directly to blank out sections of a 2D plane encompassed by a particular boundary, or to build a trusted application into the OS to support the identification of 3D objects to avoid [47, 48]. Another approach would be to instrument individual spaces or objects within the environment to broadcast their own privacy preferences, either with physical markers or wireless signals [50]. Once a privacy policy has been received and the source has been verified, whether as a broadcast or from one of the device's own subsystems, it is the operating system's responsibility to enforce the necessary response logic, and sanitize the raw sensor data according to the received policy.

Unfortunately, collecting data from system sensors and nearby devices is not the source of information that must be secured. AR systems are also subject to side-channel attacks, in which externally observable data such as power consumption or network traffic patterns are used to infer private information about the user. For example, it is possible to infer a user's location based on patterns in network traffic when downloading AR data [39]. Additionally, end users typically have a poor understanding of what data is being collected and how it is being processed [12, 14], making them more likely to agree to privacy-invading permissions or to overlook suspicious activity from malicious applications.

However, the primary drawback to these solutions is that they require users to provide explicit privacy preferences, or for devices and environments to establish and broadcast individualized security policies. Additionally, application developers are expected to monitor their own resource consumption or network traffic patterns to combat side-channel attacks. There is no coordination between users, devices, or environments, and as such, privacy and security decisions must be made on a case-by-case basis.

3.2 Transformation Phase Attacks

Attacks during the transformation phase are largely focused on compromising the internal computer vision logic of the system, usually by modifying inputs either to produce an unexpected output or to prevent recognition entirely. Computer vision modules have long been known to suffer under less than ideal environmental conditions such as low light, inclement weather, or oblique viewing angles. However, research has also shown that image classification and other computer vision operations are also vulnerable to small changes in input images [15, 16, 32, 40]. These minor changes or "perturbations", so small they're practically imperceptible to humans, are interpreted by the system in such a way that the underlying vision module can return unexpected results, either preventing the desired result or producing a result more profitable to the attacker.

Additionally, system inputs can be manipulated to prevent recognition altogether. For example, certain patches can be attached to people's clothing in order to prevent surveillance systems from recognizing them as human [55]. Alternatively, clear bubble-like masks can be worn,[9] distorting the facial features enough to fool facial recognition software but still allow another person to infer facial expression. "Phantom" glasses also exist,[10] which reflect visible and infrared light back at cameras to block out the area of the face.

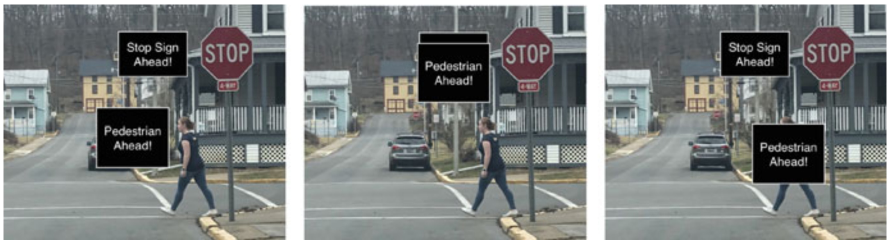
Each of these attacks requires the adversary to have some degree of knowledge of the underlying computer vision model in order to exploit it. Therefore, traditional solutions typically involve making some change to the underlying model that negates the attacker's knowledge of that model. However, updating computer vision components on distributed devices such as those found in AR systems can be expensive and unreliable, especially when the devices are resource-constrained.

3.3 Output Phase Attacks

Finally, attacks during the output phase manipulate the type and styling of virtual content being served to the user. Such attacks can be characterized in terms of *who* can display content, *what* content can be generated, *when* content will be displayed, and *where* content will be placed [28]. By compromising system output, attackers can distract their victims from the real-world, make them ill from motion sickness, or interfere with content from other trusted applications. As AR systems using head-mounted displays or smart windscreens (such as vehicles or helmets) become more

[9]https://www.businessinsider.com/clothes-accessories-that-outsmart-facial-recognition-tech-2019-10.

[10]https://mashable.com/review/review-reflectacles-phantom-anti-facial-recognition-technology-glasses-frames/.



(a) (b) (c)

Fig. 3 Examples of output attacks for AR systems. (**a**) Original. (**b**) Blocking other Apps. (**c**) Blocking real world

popular, dealing with output phase attacks also becomes more important in order to ensure user safety.

A malicious application can compromise the "who" and "what" aspects via a number of methods. Attacks made during the transformation phase can modify the result such that unintended content (or no content at all) is displayed. A malicious application running on a multi-program system can display its own content on top of the content from another application, effectively rendering that content obsolete (Fig. 3b). If the application developers themselves are the attackers, then the "when" and "where" aspects of system output can also be leveraged to attack the user. Timing of content can be managed so that it is distracting or irrelevant to the user's current context. Content can also placed poorly, either so close to the user's viewpoint so as to take up most or all of the view space, or locked in place relative to the user's head so that she cannot turn away from it, or placed over top more relevant real-world content such as a staircase or stop sign (Fig. 3c).

Some researchers have proposed OS-level solutions to manage output security in AR applications. In [29, 30], the authors designed an AR-specific middle layer between installed applications and the underlying system sensors and drivers. This middle layer, called Arya, would be responsible for intercepting raw sensor data, translating it to high level recognizer objects, and applying security policies to any resulting application output. Supported policy logic includes detection of and subsequent remediating logic for a variety of security and safety-violating situations, such as content that moves too quickly, takes up too much of the user's view, blocks road signs or pedestrians while driving, and more. Application output found to be in violation of a policy can either be modified to satisfy the policy, or blocked altogether.

However, these defenses suffer from a number of limitations. While the concept of an AR-specific operating system is not new, it has yet to gain sufficient traction for commercial and research platforms. Thus, any device running solutions such as these will be doing so on top of more commercially available operating systems, and in doing so, incurring non-trivial amounts of processing overhead. Additionally, output policies must also be explicitly written and applied on a frame-by-frame basis for pairwise combinations of apps and users. Some work has been done to pave the way for securely sharing output in multi-user systems [31, 51], but the focus of such work has been on pairs of interacting users rather than larger groups.

The limitations described above can be resolved by moving portions of the AR system to the edge. In the following sections, we will describe the nature of AR systems at the edge, the impact that moving to the edge has on AR system requirements, and finally, how the edge can help resolve these security concerns.

4 AR Security at the Edge

The architecture of an edge-enabled AR system consists of several actors, as displayed in Fig. 4. The first is the **AR device**, which may be a smartphone, head-mounted display (HMD), or other "smart" display, such as a vehicular windshield or helmet faceplate. The AR device connects to the network through a **wireless access point** or base station, which will then forward the data from the device to a **server**, either there on the edge or deeper within the network.

Figure 5 shows the updated AR pipeline when offloading operations to the edge, the most commonly offloaded operations being those in the transformation phase. First, the input phase now includes any necessary pre-processing of the collected data, such as downsampling an image or video clip. The AR device then forwards the data to its associated wireless access point, which in turn forwards the data to the edge server. The transformation phase then begins, wherein the server performs the necessary computer vision or machine learning operations on the data, such as object classification or pose estimation. The results of the operation are then sent back to the wireless access point, which forwards them to the AR device. Finally, in the output phase, the AR device processes and responds to the results of the offloaded operation, usually by providing a visual cue to the user, such as displaying a particular label.

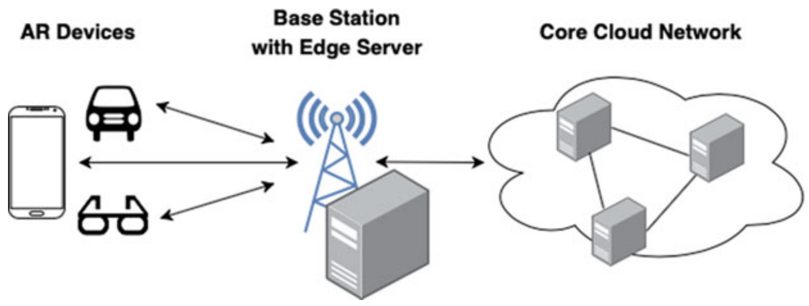


Fig. 4 Example architecture of edge-enabled AR system

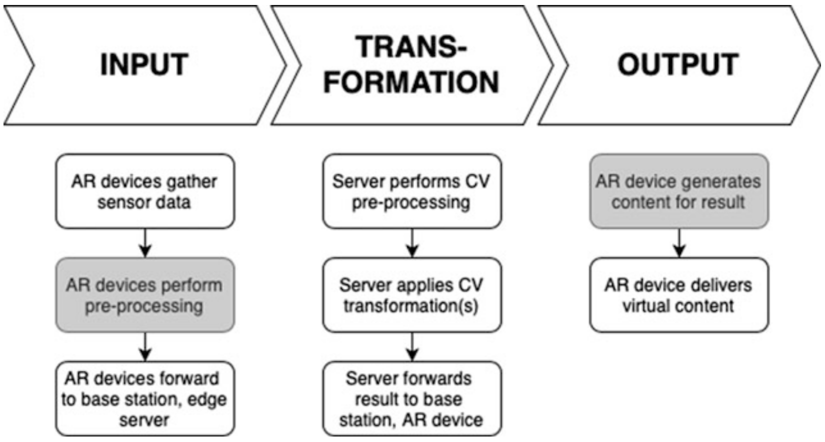

INPUT
TRANS-FORMATION
OUTPUT
AR devices gather sensor data
AR devices perform pre-processing
AR devices forward to base station, edge server
Server performs CV pre-processing
Server applies CV transformation(s)
Server forwards result to base station, AR device
AR device generates content for result
AR device delivers virtual content

Fig. 5 Modified AR pipeline when offloading to edge server. Highlighted steps are optional, depending on division of labor between edge server and local AR device

4.1 System Requirements for AR at the Edge

Once an AR system has been moved to the edge, its system requirements change slightly. Having access to additional storage and compute resources on the edge frees up local resources on the AR device, but also introduces latency while waiting for offloaded operations to return their results. It is up to the system designers to decide which operations to offload, how often, and with what data.

As discussed above, the transformation phase is the portion of the pipeline that is traditionally offloaded to the edge. This is because the computer vision or machine learning operation of choice can more or less be treated as a black box, providing results for a given set of inputs independent of anything else happening on the AR device. However, the exact data exchanged before and after offloading differs based on the type of operation being executed. Some systems may offload video clips, individual frames, or simply sets of features extracted from the local data stream. The results that the server returns could be labels, coordinates, or even encoded object files representing the content to be displayed. The degree to which data is pre-processed before offloading, or the virtual content is generated after receiving a result (designated as highlighted steps in Fig. 5) are variable depending on the system being implemented and the availability of resources on the AR device itself.

4.2 Quality of Augmentation at the Edge

When moving an AR system to the edge, it is important to keep in mind how the new system architecture will affect user experience, as the ultimate success of an AR system is tightly correlated with user experience and perception. It is not enough for the system to perform with low resource consumption or for the machine learning

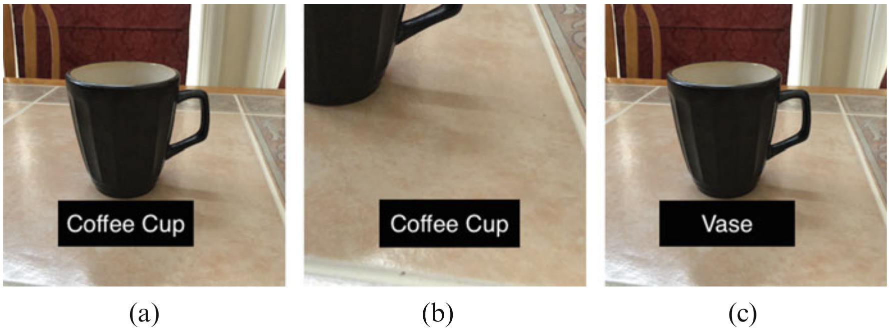


Fig. 6 Examples of poor user experience in AR systems. (**a**) Original. (**b**) Placement error (lag). (**c**) Recognition error

components to achieve high precision and recall. The system must combine these features into a sum greater than its parts; it must not only correctly identify context, but also produce the correct output in the correct manner in a timely fashion, a characteristic that can be broadly summed up by the term **quality of augmentation**, or QoA.

Low QoA has direct impacts, not only to the general appeal of a system, but also to the physiological responses of users. Motion sickness for AR and VR systems, also called "simulator sickness", occurs when a user experiences different stimuli from the system than she expects from the real world, such as recognition errors or delays in label placement, as shown in Fig. 6. A user with simulator sickness may experience disorientation, nausea, or vomiting. AR system designers must be very careful, therefore, to ensure that users of their systems experience a high QoA.

4.2.1 Metrics for QoA

As important as QoA is to AR systems, it is notoriously difficult to define. While QoA is supported by traditional system metrics such as battery drain or RAM and CPU consumption, these data points cannot guarantee how users will ultimately respond. Therefore, many system designers also choose to supplement traditional system testing with more qualitative testing in the form of user studies. From gauging simulator sickness [26] to focal length [8] to facilitating device pairing in multi-user systems [54], gathering feedback directly from users in the form of questionnaires and interviews has been the traditional method of gauging the quality of a system's user experience. However, due to their open-ended nature, user studies are limited in their ability to provide quantitative system metrics, particularly when evaluating systems in the wild rather than the lab [33].

Despite the nebulous nature of QoA, there are certain measurements that an AR system designer can make to approximate it, such as measuring **how quickly content is being generated**. The speed at which the system collects data, processes it, and displays the resulting output can be measured in a number of ways, such as

Table 1 Symbol definitions for QoA metrics

Symbol	Description
p	Precision; percentage of true positive predictions out of all positive predictions made
r	Recall; percentage of true positive predictions made out of all ground truth positives
s_i	Side length (in pixels) of the ith video frame with area s^2
ψ	Modeled complexity function (convex w.r.t. s_i^2)
ξ	Modeled accuracy function (concave w.r.t s_i^2)
c_i	Computational complexity of calculating the ith frame
a_i	Analytical accuracy from calculating the ith frame

end-to-end latency [9, 35–37, 44, 45, 49], and the number of *frames displayed per second* (FPS) [29, 48]. Managing these values ensures that the system can respond in real-time to changes in the user's behavior and environment, and are discussed in greater detail in Sect. 4.3.

An AR system designer can also approximate QoA by measuring the **accuracy of generated content**, typically in terms of intersection-over-union (IOU) [35, 45] and mean average precision (mAP) [36, 37]. IOU is utilized often in computer vision operations that identify specific areas of an image, such as drawing of bounding boxes. IOU is calculated as the percentage of area shared between a ground truth box and the predicted box, divided by the total area of those two boxes. A higher IOU means that the predicted box overlaps and shares more area with the ground truth box. Likewise, mAP is a common metric for computer vision operations that focus on classification, that is, identifying the subject of an image. mAP relies on the concepts of *precision*, or the rate of true positive predictions out of all predictions made (Eq. 1), and *recall*, or the rate of true positive predictions out of all ground truth positive matches (Eq. 2) (Table 1). mAP calculates the average area-under-curve when plotting precision and recall for a set of classifications. A higher mAP means that the system was able, not only to make predictions correctly, but to identify a high percentage of available items.

$$p = \frac{TP}{TP + FP} \tag{1}$$

$$r = \frac{TP}{TP + FN} \tag{2}$$

4.2.2 Trade-Offs for QoA

To support better quality of augmentation, AR system designers can made certain trade-offs in the **complexity** and subsequent **accuracy** of the internal computer vision operations, traditionally the most resource-intensive portions of an AR system. Figure 6 demonstrates how imbalanced complexity and accuracy can impact user experience. Figure 6b shows a system which, while accurate, takes so long to compute a label that the label placement is no longer accurate to the user's context. Figure 6c shows a system which quickly computes and places a label, but is highly inaccurate.

To understand the relationship of accuracy and complexity, the authors in [37] modeled the amount of time required to perform classification on input frames of varying sizes. They found that, for a given frame with resolution s^2, the time to complete the operation increases more quickly as the input resolution grows, while increases in accuracy slow and then plateau. They therefore proposed Eqs. 3 and 4, which model computational complexity and accuracy as convex and concave functions respectively with respect to the resolution of the ith video frame (Table 1).

This means that it is the developers' responsibility to explore the impacts of varyingly complex and accurate models to the performance of their AR systems. For example, a simpler object detection model might be preferred for a system with strict latency requirements, while a more complex bounding box model might be appropriate for a system that requires highly accurate results but can tolerate some additional latency. Ultimately, it is up to the AR system designers to make these trade-off decisions relative to the needs of their specific systems.

$$c_i = \psi(s_i^2) \tag{3}$$

$$a_i = \xi(s_i^2) \tag{4}$$

4.3 Impacts to Service Latency

The principal problem for AR systems trying to maintain QoA at the edge is service latency. Augmented reality systems have strict latency restrictions in order to maintain an immersive user experience; indeed, experts have calculated a maximum allowable latency of only 100ms for AR operations [5, 6]. This means that the AR system has less than a tenth of a second from the point of data collection to process that data, calculate a result, and display virtual content according to that result (Table 2).

4.3.1 Calculating Service Latency

For an edge-enabled AR system, **service latency** can be defined as the combination of transmission latency, or the time it takes to transmit data from the AR device to the server, and computational latency, or the time required to complete the offloaded operation. This is reflected in Eq. 5.

Transmission latency only applies to systems with offloaded operations, and is directly influenced by the amount of data being transported. (Because we are

Table 2 Symbol definitions for latency metrics

Symbol	Description
L_s	Service latency
L_i^t	Transmission latency for the *i*th frame
L_i^c	Computational latency for the *i*th frame on a shared, multi-user server
β	Number of bits per pixel
R_i	Average wireless data rate for the *i*th user
N	Collection of possible edge servers
M	Collection of users
$\sigma_{i,n}$	Binary flag representing whether the *i*th user has been assigned to the *n*th server
c_i	Computational complexity of processing the *i*th frame
f_n	Set of available computational resources on the *n*th server

assuming the target server to be on the edge, we do not consider the impact of current traffic loads within the core network to transmission time.) Equation 6 demonstrates how transmission latency can be calculated from a system level (rather than wireless link performance level) where β is the number of bits per pixel, s_i^2 is the number of pixels in the *i*th frame with a side length of s, and R_i is the *i*th user's average wireless data rate [37].

Computational latency is heavily influenced, not only by the amount of data, but also the complexity of the operation being performed and the amount of resources available on the server when sharing that server with other users. Equation 7 demonstrates how this type of latency can be calculated where c_i is the computational complexity of the *i*th task, and where, for the *n*th server, $\sigma_{i,n}$ is a binary variable representing whether the *i*th user is assigned to this server, f_n is the total set of available computational resources, and $\frac{f_n}{\sum_{m\in M}\sigma_{m,n}}$ is the amount of resources allocated to each user [37].

$$L_s = L_t + L_c \tag{5}$$

$$L_i^t = \frac{\beta s_i^2}{R_i} \tag{6}$$

$$L_i^c = \sum_{n\in N} \sigma_{i,n} \frac{c_i}{f_n} \sum_{m\in M} \sigma_{m,n} \tag{7}$$

4.3.2 Trade-Offs to Manage Latency

The degree to which service latency impacts a given AR system depends on the nature of the system and the portions of the AR pipeline being offloaded (shown in Fig. 5). For resource constrained devices such as smartphones and HMDs, the operations to be performed might be too resource-intensive to complete locally. For other devices such as "smart" vehicles with augmented windscreens, the operations to be performed might require more data than an individual device can collect on its own. Therefore, AR system designers must selectively offload certain operations either to preserve local resources or to aggregate and process larger data sets. Regardless of the purpose of the offload, the system must still respect the appropriate latency restrictions to maintain QoA for users.

While network traffic conditions and server workloads are not always under one's control, the amount of data being transmitted and the operation being performed *are*. To explore the implication of this, the authors in [37] modeled the impacts to latency and accuracy when offloading object classification operations for models of various sizes. They observed that *decreasing* either the resolution of the image being transmitted or the complexity of the model being executed dramatically decreased latency, while incurring only a minor cost to accuracy [36]. When *increasing* the resolution of the image being transmitted, they observed that computational latency increased at a gently exponential rate while accuracy tapered off after an initial increase. This led the authors to conclude that latency and accuracy of AR operations can be dynamically managed in response to changing network conditions, assuming the client application has a variety of input image sizes and computational models to choose from.

However, decreasing the frame size or complexity of the computational model has a downside, since both of these factors directly impact the system's quality of augmentation. Indiscriminately decreasing frame size or computational model complexity may decrease latency, but it will also negatively impact the accuracy of computation results. However, by selectively modifying these parameters to perform controlled approximate augmentation, clients can strike a balance between managing latency and preserving QoA.

Assuming that service latency is appropriately managed, moving an AR system to the edge gives that system access to increased computational resources, larger data sets aggregated from multiple devices, and the ability to make collaborative decisions between neighboring devices. These features can provide a number of security benefits to the various stages of the AR processing pipeline. The following sections discuss these implications in greater detail.

4.4 *Securing AR Systems Using the Edge*

4.4.1 Input Phase Defenses

User Authentication Recall from Sect. 3.1.1 that one of the major vulnerabilities for the input phase of AR systems is authenticating the system user. Typical user authentication solutions, such as those discussed previously, assume that a user authenticates herself with a partner using wearable sensors and coordinating protocols. An example of this is the headset-pairing protocol proposed in [54], which suggests the use of gestures to create a secure communications channel directly between devices. The devices wirelessly exchange public keys, and using those keys, display abstract shapes in the air to be traced by the corresponding partner. If each user observes their partner to trace the same shape, then the pairing is successful.

However, approaches like these are limited when attempting to authenticate individual users within a larger, co-located group. Multiple users within the same physical space can create authentication interference, such as microphones collecting overlapping voice commands, or cameras collecting conflicting gestures. There is also logical ambiguity regarding connections to make between users; the system must be able to support a wide variety of communication permissions within the group, such as exclusive partner pairs, groups open to accepting new members, and closed groups with no option for outsiders to join in. The edge can help with these authentication problems by becoming the intermediary for user authentication in group settings, as demonstrated by the sample system in Fig. 7. The edge server would take responsibility for managing repositories of public keys and group membership information, mediating connections with new group members, and supplying short-term keys for use in direct communication among the group.

Data Collection Moving an AR system to the edge can also help control the collection of environmental data in order to preserve user and bystander privacy, as discussed in Sect. 3.1.2. In particular, edge servers are a highly advantageous place to aggregate privacy preference data from multiple users in the same geographic area, and to generate privacy policies dynamically based on environmental context in that area. One particular solution suggests the use of specific privacy markers to prevent recording of restricted 2D surfaces and 3D objects [48]. The original implementation for this solution required the authors to update their device's

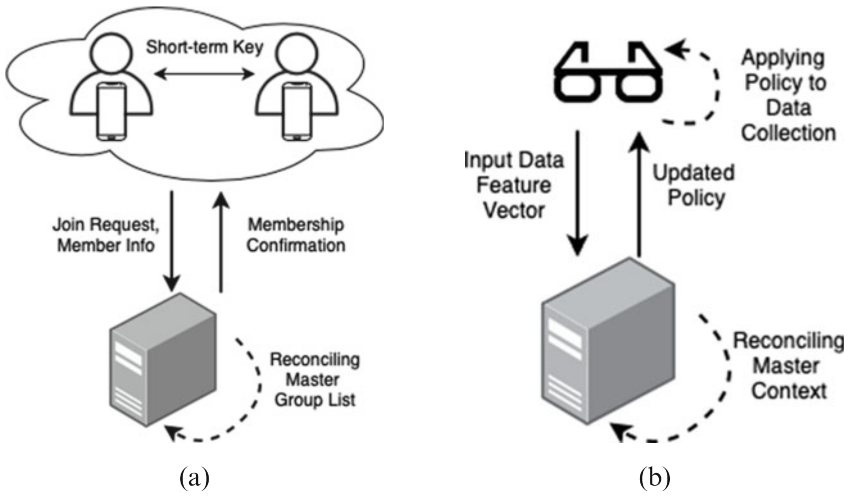

Short-term Key
Join Request, Member Info
Membership Confirmation
Reconciling Master Group List
(a)
Applying Policy to Data Collection
Input Data Feature Vector
Updated Policy
Reconciling Master Context
(b)

Fig. 7 Potential uses for edge servers in (**a**) output policy generation and (**b**) display reconciliation between multiple users

hardware access layer (HAL) in order to intercept frames from the camera. This approach allowed them to sanitize each frame before applications running at less privileged layers could access them, but also incurred a significant performance hit. It is possible that, instead of modifying the device's HAL, an intermediate layer could have been created to offload the frame sanitation operation to a trusted edge server. Doing so would also allow for the aggregation of privacy markers submitted by multiple users, or for a system administrator to update the types of markers that the system can recognize without having to push an update to individual devices.

Instead of markers, systems can also leverage user- or administrator-provided policies to manage data collection. Systems such as I-Pic [1] require devices to broadcast privacy policies over BLE to prevent image capture in undesirable contexts. I-Pic assumes that each user-device pair is proactive in learning and following the privacy preferences of those nearby. Therefore, each device is responsible for storing its own local cache of privacy policies for users in the immediate area, and enforcing those policies on its captured images. Other solutions such as PrivacyManager [34] utilize data gathered from the environment (such as location, time of day, or ambient light and sound levels) to determine whether it is appropriate for a camera to record. Unlike I-Pic, PrivacyManager assumes that each user-device pair is *untrusted*, and so relies on a central authority such as a system administrator to generate and distribute policies. For both of these systems, moving data storage and policy generation to the edge is highly beneficial. Devices using I-Pic could store their privacy preferences on the edge server, without having to keep local copies. Devices using PrivacyManager could aggregate contextual information at the server, and use it to dynamically update and distribute policies.

4.4.2 Transformation Phase Defenses

Defenses against attacks conducted during the transformation phase of the AR pipeline borrow heavily from advancements in adversarial machine learning. Traditional defenses in this area fall into three broad categories, as described in [4]. The first option is using **modified input for training and testing**. This is considered to be a "brute force" strategy, employing approaches such as compressing the input images, adding random padding, and even training with actual adversarial examples. While these kinds of defenses can help with adversarial inputs, they can also hinder correct processing of valid inputs, since any changes to the input images would be applied regardless of the validity of the image in question. The second possibility is **adding onto the computational model**, usually in the form of a secondary network trained especially either to generate new adversarial inputs or reverse perturbations applied to existing ones. Unfortunately, the impact to service latency incurred by adding secondary networks to a computational model makes this option less attractive for AR systems.

The third option for defense is **modifying the computational model** itself, such as updating weights or loss functions within the network to reflect adversarial inputs, inserting a masking layer to the network before the classification layer, or adding a sub-network trained specifically on classification inputs. One notable example of this kind of defense is called *defensive distillation* [42]. This approach builds on the idea of "distilling" a neural network [21], or using the probability vectors generated by a pre-trained network to train a second, smaller network. This second network can then be executed on a smaller, more resource-constrained device, which is an ideal feature for AR systems. Defensive distillation uses these probability vectors, not to compress the network into a smaller architecture, but to train a new instance of the same network architecture. Training this new network instance, not on discrete labeled inputs, but on probability vectors for a group of labels, gives the network a richer understanding of the trained data sets, making it more robust and resilient to adversarial inputs.

The approaches discussed here are intended to be applied to the computer vision model before the parent application is deployed for use in the real world. However, utilizing the edge for AR systems makes these defenses, particularly modifications to an active computational model, much more feasible. Not only do edge servers provide additional processing and storage resources to AR devices, but they also provide a single point of deployment for an AR system's computational model. When changes to a model must be made, or a model must be replaced completely, it is simpler and faster to deploy the updated model to an edge server than to push that model to every subscribing AR device.

4.4.3 Output Phase Defenses

Recall from Sect. 3.3 that output attacks against AR systems are described in terms of who generates the content, what content is being generated, and when and where that content will be displayed. In these situations, the malicious entity may be the current application, a background application, or another user. Regardless of the source, the appropriateness of any given system output is relative to the user's current context, as output generated in an empty room will pose no threat while the same output on a crowded street may obscure crucial real-world content. Solutions like Arya [29, 30] utilize output control policies imposed at the OS-level to restrict the conditions under which output can be displayed. However, this system makes several simplifying assumptions, including manual creation of output policies and trustworthiness of fellow system users. There is no consideration for dynamically changing policies or for untrusted fellow users.

Moving output policy creation to the edge allows a system to aggregate and leverage data collected by many devices in order to determine the appropriate conditions to write into the policy. Reference [2] builds on Arya to do this, leveraging reinforcement learning on edge nodes to generate output policies dynamically. However, this solution relies on simulations of application use in order to generate policy rules, and provides no discussion on how to verify or improve the performance of that policy in the real-world. An alternative to this solution might be similar to Fig. 7a, which requires applications to register their output preferences with a designated edge server, which consolidates and assigns display parameters to each one. Alternatively, the same application being executed on multiple devices could maintain a cache of context data at the edge, such as weather conditions or network loads, that would subsequently impact how to display data for each subscribing user.

Regardless of how the output policy is generated, additional consideration needs to be paid when dealing with multi-user systems. In [31], the authors conducted a series of user studies investigating how subjects in a multi-user AR system interact with the environment and each other. They observed that users were quick to generate or manipulate content to negatively impact their partners, including modifying each other's appearance, setting up virtual barriers to block real world content, and hiding behind virtual content to obscure their own actions. Similarly, in [43], the authors observed a strong sense of ownership between users and the virtual content within their immediate space, along with adverse responses to other users modifying that content or space. In [51], the authors present several case studies of both co-located and geographically dispersed users, sharing content using both opt-out and opt-in defaults. Because these interactions are happening in real-time, this presents a unique opportunity for edge-computing to act as an intermediary between devices, as demonstrated in Fig. 7b. Instead of devices pairing up and sending content directly to each other, they can send their intended actions to an edge server, which reconciles the master view of the shared content, and delivers a sanitized view to each subscribing user. The physical proximity of an edge server to the users provides greatly improved latency over a central cloud server, allowing the system to maintain real-time reactions.

5 Analysis and Discussion

Today's commercial libraries for augmented reality do not currently provide explicit support for moving an AR system to the network edge. Some libraries are beginning to offer remote support in specific areas such as cloud-synced geo-located "anchors" for multi-user AR systems,[11] or leveraging cloud-based resources to execute more intensive versions of local operations.[12] However, the developer has no control over *where* these data are being stored or operations are being executed, which means that any system utilizing these functions is subject to all of the latency costs imposed by the cloud.

To explore what kind of latency costs a developer might expect when adopting offloaded operations with these commercial platforms, we conducted a series of tests comparing service latency when performing a given operation both locally and remotely. Because there are no commercially available AR libraries that explicitly support the edge, these offloaded operations are instead being executed on the

[11] https://developers.google.com/ar/develop/java/cloud-anchors/overview-android.

[12] https://firebase.google.com/docs/ml-kit/android/label-images.

cloud. However, should these library publishers offer edge support in the future, the performance of such offloaded operations can only improve. Leveraging the results of our experiments and the current state of commercial library offers, we conclude this section with a list of open problems for AR research and development at the network edge.

5.1 Experimental Results Comparing Local and Remote Processing

To conduct our experiments, we focused on two common computer vision tasks in augmented reality: text recognition and image labeling. For each computer vision task, we developed two "flavors" of a smartphone-based AR application - one which performed the task locally, and one which performed the task remotely. We then piped in frames from a pre-recorded video, so that each application would be operating on the same inputs. The CV module would operate on a single frame at a time; once the current frame completed, a new frame from the video stream would be requested, meaning slower modules received frames more infrequently. We conducted our experiments using a Samsung Galaxy GS9 smartphone running Android 10. The logic for both the local and remote computer vision operations were implemented using Firebase's Machine Learning Toolkit (ML Kit).[13]

We explored the impact of offloading a given operation based on the complexity and quality of the input video frame. For each application, we designed a collection of 20 second-long video clips with increasing complexity (volume of text, number of subjects shown on screen respectively) and quality of the video. Details of the video files used for the text recognition and image labeling apps can be found in Tables 3 and 4 respectively.

As each video frame was fed into the app's CV module, we measured the amount of time the system took to process the frame in milliseconds. Recall that some experts have calculated a maximum allowable latency of only 100ms for truly dynamic and immersive AR operations [5, 6]. With this in mind, we examined the time required to perform each selected operation (text recognition and image

Table 3 System statistics for input videos used in **text recognition**. All videos had same resolution (1920 x 1080 px) and same running time (20 s)

	Volume of text					Video quality			
	Lvl 1	Lvl 2	Lvl 3	Lvl 4	Lvl 5	Low	Med	High	Lossless
File size	8 MB	11.6 MB	10.7 MB	12.3 MB	12.7 MB	500 KB	1 MB	8 MB	8 MB
CRF	0	0	0	0	0	51	37	23	0
# of Frames	608	614	607	613	610	608	608	608	608

[13] https://firebase.google.com/docs/ml-kit.

Table 4 System statistics for input videos used in **image labeling**. All videos had same resolution (1920 x 1080 px) and same running time (20 s)

	Number of subjects					Video quality			
	1	2	3	4	5	Low	Med	High	Lossless
File size	7.6 MB	13 MB	12.9 MB	16.5 MB	12.3 MB	518 KB	1 MB	7.6 MB	7.6 MB
CRF	0	0	0	0	0	51	37	23	0
# of Frames	606	609	615	610	608	606	606	606	606

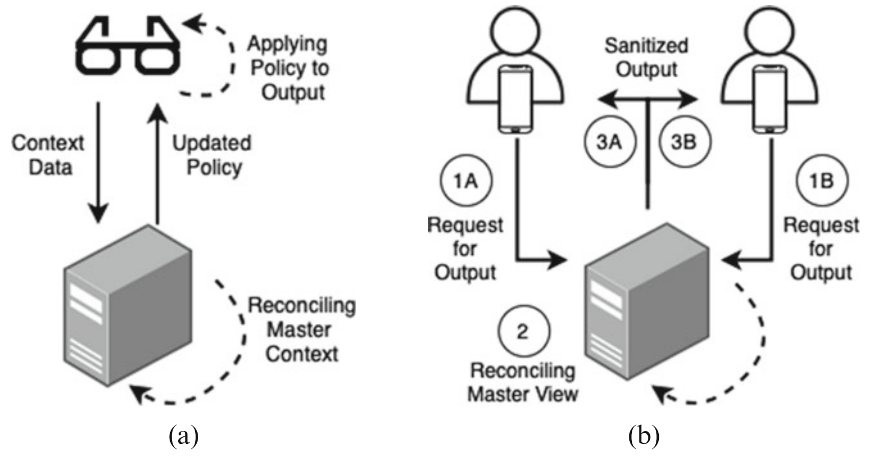


Fig. 8 Service latency when performing local and remote **text recognition** with (**a**) increasing volume of text and (**b**) increasing levels of video quality

labeling) on the appropriate video set, the results of which are shown in Figs. 8 and 9 respectively.

From these results, we can see a clear and significant impact to service latency when offloading a given operation, with minor improvements if the frame content is less complex or of low quality. With offloaded operations taking an average of 600 to 1000 ms to complete, relying on offloaded operations is a non-option for AR applications attempting to maintain a 100ms cap on service latency. However, the potential is there for AR library publishers to integrate support for the edge into their existing products and services. Performing an operation locally will always be faster than offloading, but for certain tasks, the increased compute and storage resources offered by the edge cannot be replicated or replaced on a standalone device. For those situations, integrating edge-support into these popular libraries can only be an improvement for overall system performance.

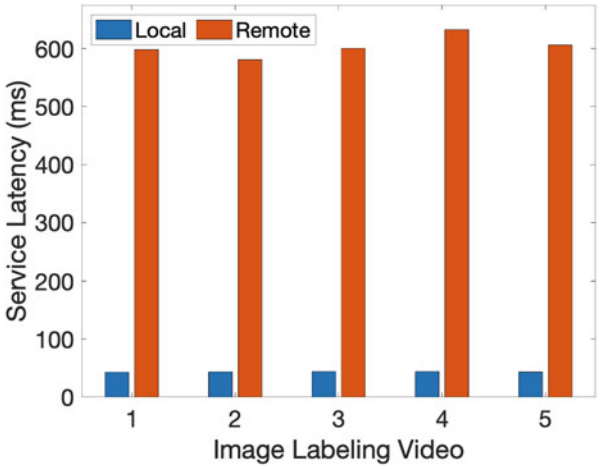


Fig. 9 Service latency when performing local and remote **image labeling** with (**a**) increasing volume of subjects and (**b**) increasing levels of video quality

5.2 Impacts of Security Measures on QoA

As our AR systems inevitably grow beyond our standalone devices, the importance of security measures and their impact on system performance cannot be ignored. However, while the latency costs of offloading computer vision operations remain high, developers will be unlikely to add to that burden by incorporating security operations as well. Therefore, developers must be sensitive to the trade-off between latency and security, and manage it carefully according to their systems' needs.

One benefit of the edge that developers should be sure to leverage is the centralized storage offered by edge nodes. Edge node storage allows developers to keep large collections of images, labels, 3D models, privacy and access control policies, system state data, and computer vision modules without having to store them locally on users' devices. State data in particular can be aggregated at a high level to make better-informed decisions than a single device could make on its own. By relying on centralized storage, developers can keep the local versions of their applications small and lightweight, freeing up local resources for additional security operations.

Developers can also take advantage of the edge for redundant, multi-tiered operations with increasing levels of precision. An example of this is exhibited by Firebase's ML Kit, which recognizes 400 image labels when executing locally, but over 10,000 labels when executing in the cloud.[14] By using a more coarse-grained, lightweight model on the local device, and offloading to a more robust and

[14]https://firebase.google.com/docs/ml-kit/label-images.

precise model on the edge when necessary, developers can decrease the number of operations offloaded while still maintaining high levels of precision when necessary. Ultimately, it is the developer's responsibility to balance the security concerns of an AR system with QoA and user experience. As AR systems start to support multiple users and multiple applications in particular, this balance will become even more important.

5.3 Open Problems

In closing, we will discuss a selection of open problems that should be explored for AR systems security at the edge. The first is the development of **more holistic measurements for quality of augmentation (QoA)**. Traditional metrics for QoA such as resource consumption, frame rate, precision, and recall have been borrowed from software engineering and computer vision domains, and can help approximate QoA in standalone systems. However, these metrics fail to capture the impact of service latency and the subsequent trade-offs in computational accuracy that occur when moving AR systems to the edge. Therefore, we must develop new, more holistic metrics to reflect the changing nature of edge-enabled AR systems.

The second is the addition of **edge support to commercial AR-focused libraries and APIs**. There are many AR-focused libraries and APIs which already support the abstraction of common computer vision operations. It makes sense, therefore, for them to also include some support for offloading these operations to the edge. Some APIs, such as Firebase's ML Kit (discussed above), allow a developer to designate whether an operation should be performed locally or remotely, but do not allow the developer to stipulate what server to use. While adding such support to commercial libraries can improve overall system performance, there are risks involved, namely in controlling accidental data leakage. Any research efforts in this area would have to take this into consideration as well.

The third is for **improved user- and policy-driven I/O security for multi-user and multi-application systems**. Current approaches in AR system security focus primarily on single-user, single-application systems, with some works focusing on pair-based interactions between users. However, the capabilities of edge-enabled systems for increased processing, data aggregation, and collaboration between devices pave the way for systems supporting true multi-user, multi-application systems. It is easy to envision, for example, smart vehicles collecting data and collaborating in real-time to build wide-scale views of roadway conditions, and displaying multiple types of output on their augmented windshields, such as weather alerts, traffic updates, route suggestions, and other information. It therefore becomes crucial for these platforms to have methods of reconciling or sandboxing the input and output operations of these applications, to prevent them from compromising or interfering with each other.

6 Conclusions

In this chapter, we have introduced the basic concepts of AR systems and their related security concerns, as well as the implications of moving such system operations to the edge. We presented a number of potential impacts that edge computing can make on AR system security, including user authentication, data collection, transformation, and output verification. We also presented three open problems for future work: holistic metrics for quality of augmentation, edge support in commercial AR-focused libraries, and improved I/O security for multi-user-multi-app systems.

References

1. Aditya, P., Sen, R., Druschel, P., Joon Oh, S., Benenson, R., Fritz, M., Schiele, B., Bhattacharjee, B., Wu, T.T.: I-pic: A platform for privacy-compliant image capture. In: Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, pp. 235–248 (2016)
2. Ahn, S., Gorlatova, M., Naghizadeh, P., Chiang, M., Mittal, P.: Adaptive fog-based output security for augmented reality. In: Proceedings of the 2018 Morning Workshop on Virtual Reality and Augmented Reality Network, pp. 1–6 (2018)
3. Akçayır, M., Akçayır, G.: Advantages and challenges associated with augmented reality for education: A systematic review of the literature. Educ. Res. Rev. **20**, 1–11 (2017)
4. Akhtar, N., Mian, A.: Threat of adversarial attacks on deep learning in computer vision: A survey. IEEE Access **6**, 14410–14430 (2018)
5. Card, S.K.: The Psychology of Human-Computer Interaction. Crc Press (2018)
6. Card, S.K., Robertson, G.G., Mackinlay, J.D.: The information visualizer, an information workspace. In: Proceedings of the SIGCHI Conference on Human factors in computing systems, pp. 181–186 (1991)
7. Carlson, K.J., Gagnon, D.J.: Augmented reality integrated simulation education in health care. Clin. Simul. Nurs. **12**(4), 123–127 (2016)
8. Chakravarthula, P., Dunn, D., Akşit, K., Fuchs, H.: Focusar: Auto-focus augmented reality eyeglasses for both real world and virtual imagery. IEEE Trans. Vis. Comput. Graph. **24**(11), 2906–2916 (2018)
9. Chen, K., Li, T., Kim, H.S., Culler, D.E., Katz, R.H.: Marvel: Enabling mobile augmented reality with low energy and low latency. In: Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems, pp. 292–304. ACM (2018)
10. Chen, P., Liu, X., Cheng, W., Huang, R.: A review of using augmented reality in education from 2011 to 2016. In: Innovations in Smart Learning, pp. 13–18. Springer (2017)
11. Chung, N., Han, H., Joun, Y.: Tourists' intention to visit a destination: The role of augmented reality (ar) application for a heritage site. Comput. Hum. Behav. **50**, 588–599 (2015)
12. Denning, T., Dehlawi, Z., Kohno, T.: In situ with bystanders of augmented reality glasses: Perspectives on recording and privacy-mediating technologies. In: Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems, pp. 2377–2386. ACM (2014)
13. tom Dieck, M.C., Jung, T.: A theoretical model of mobile augmented reality acceptance in urban heritage tourism. Curr. Issues Tour. **21**(2), 154–174 (2018)

14. Egelman, S., Kannavara, R., Chow, R.: Is this thing on?: Crowdsourcing privacy indicators for ubiquitous sensing platforms. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, pp. 1669–1678. ACM (2015)
15. Elsayed, G., Shankar, S., Cheung, B., Papernot, N., Kurakin, A., Goodfellow, I., Sohl-Dickstein, J.: Adversarial examples that fool both computer vision and time-limited humans. Adv. Neural Inf. Process. Syst., 3910–3920 (2018)
16. Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., Song, D.: Robust physical-world attacks on deep learning visual classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1625–1634 (2018)
17. Freina, L., Ott, M.: A literature review on immersive virtual reality in education: state of the art and perspectives. In: The International Scientific Conference eLearning and Software for Education, vol. 1, p. 133. "Carol I" National Defence University (2015)
18. Gaebel, E., Zhang, N., Lou, W., Hou, Y.T.: Looks good to me: Authentication for augmented reality. In: Proceedings of the 6th International Workshop on Trustworthy Embedded Devices, pp. 57–67 (2016)
19. Gonzalez-Franco, M., Pizarro, R., Cermeron, J., Li, K., Thorn, J., Hutabarat, W., Tiwari, A., Bermell-Garcia, P.: Immersive mixed reality for manufacturing training. Frontiers Robotics AI **4**, 3 (2017)
20. Herron, J.: Augmented reality in medical education and training. J. Electron. Resour. Med. Libr. **13**(2), 51–55 (2016)
21. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. Preprint (2015). arXiv:1503.02531
22. Jana, S., Molnar, D., Moshchuk, A., Dunn, A., Livshits, B., Wang, H.J., Ofek, E.: Enabling fine-grained permissions for augmented reality applications with recognizers. In: Presented as Part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13), pp. 415–430 (2013)
23. Jana, S., Narayanan, A., Shmatikov, V.: A scanner darkly: Protecting user privacy from perceptual applications. In: 2013 IEEE Symposium on Security and Privacy, pp. 349–363. IEEE (2013)
24. Joda, T., Gallucci, G., Wismeijer, D., Zitzmann, N.: Augmented and virtual reality in dental medicine: A systematic review. Comput. Biol. Med. **108**, 93–100 (2019)
25. Jung, T., tom Dieck, M.C., Lee, H., Chung, N.: Effects of virtual reality and augmented reality on visitor experiences in museum. In: Information and Communication Technologies in Tourism 2016, pp. 621–635. Springer (2016)
26. Kennedy, R.S., Lane, N.E., Berbaum, K.S., Lilienthal, M.G.: Simulator sickness questionnaire: An enhanced method for quantifying simulator sickness. Int. J. Aviat. Psychol. **3**(3), 203–220 (1993)
27. Kim, Y., Kim, H., Kim, Y.O.: Virtual reality and augmented reality in plastic surgery: a review. Arch. Plast. Surg. **44**(3), 179 (2017)
28. Lebeck, K., Kohno, T., Roesner, F.: How to safely augment reality: Challenges and directions. In: Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications, pp. 45–50. ACM (2016)
29. Lebeck, K., Ruth, K., Kohno, T., Roesner, F.: Securing augmented reality output. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 320–337. IEEE (2017)
30. Lebeck, K., Ruth, K., Kohno, T., Roesner, F.: Arya: Operating system support for securely augmenting reality. IEEE Secur. Priv. **16**(1), 44–53 (2018)
31. Lebeck, K., Ruth, K., Kohno, T., Roesner, F.: Towards security and privacy for multi-user augmented reality: Foundations with end users. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 392–408. IEEE (2018)
32. Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al.: Photo-realistic single image super-resolution using a generative adversarial network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4681–4690 (2017)

33. Lehman, S.M., Ling, H., Tan, C.C.: Archie: A user-focused framework for testingaugmented reality applications in the wild. In: 2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR). IEEE (2020)
34. Lehman, S.M., Tan, C.C.: Privacymanager: An access control framework for mobile augmented reality applications. In: 2017 IEEE Conference on Communications and Network Security (CNS), pp. 1–9. IEEE (2017)
35. Liu, L., Li, H., Gruteser, M.: Edge assisted real-time object detection for mobile augmented reality. In: The 25th Annual International Conference on Mobile Computing and Networking, pp. 1–16 (2019)
36. Liu, Q., Han, T.: Dare: Dynamic adaptive mobile augmented reality with edge computing. In: 2018 IEEE 26th International Conference on Network Protocols (ICNP), pp. 1–11. IEEE (2018)
37. Liu, Q., Huang, S., Opadere, J., Han, T.: An edge network orchestrator for mobile augmented reality. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pp. 756–764. IEEE (2018)
38. Makris, S., Karagiannis, P., Koukas, S., Matthaiakis, A.S.: Augmented reality system for operator support in human–robot collaborative assembly. CIRP Annals **65**(1), 61–64 (2016)
39. Meyer-Lee, G., Shang, J., Wu, J.: Location-leaking through network traffic in mobile augmented reality applications. In: 2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC), pp. 1–8. IEEE (2018)
40. Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1765–1773 (2017)
41. Moro, C., Štromberga, Z., Raikos, A., Stirling, A.: The effectiveness of virtual and augmented reality in health sciences and medical anatomy. Anat. Sci. Educ. **10**(6), 549–559 (2017)
42. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 582–597. IEEE (2016)
43. Poretski, L., Lanir, J., Arazy, O.: Normative tensions in shared augmented reality. Proc. ACM Hum. Comput. Interact. **2**(CSCW), 1–22 (2018)
44. Qiu, H., Ahmad, F., Bai, F., Gruteser, M., Govindan, R.: Avr: Augmented vehicular reality. In: Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services, pp. 81–95 (2018)
45. Ran, X., Chen, H., Zhu, X., Liu, Z., Chen, J.: Deepdecision: A mobile deep learning framework for edge video analytics. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pp. 1421–1429. IEEE (2018)
46. Rauschnabel, P.A., Rossmann, A., tom Dieck, M.C.: An adoption framework for mobile augmented reality games: The case of pokémon go. Comput. Hum. Behav. **76**, 276–286 (2017)
47. Raval, N., Srivastava, A., Lebeck, K., Cox, L., Machanavajjhala, A.: Markit: Privacy markers for protecting visual secrets. In: Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication, pp. 1289–1295 (2014)
48. Raval, N., Srivastava, A., Razeen, A., Lebeck, K., Machanavajjhala, A., Cox, L.P.: What you mark is what apps see. In: Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, pp. 249–261. ACM (2016)
49. Ren, J., He, Y., Huang, G., Yu, G., Cai, Y., Zhang, Z.: An edge-computing based architecture for mobile augmented reality. IEEE Network **33**(4), 162–169 (2019)
50. Roesner, F., Molnar, D., Moshchuk, A., Kohno, T., Wang, H.J.: World-driven access control for continuous sensing. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1169–1181. ACM (2014)
51. Ruth, K., Kohno, T., Roesner, F.: Secure multi-user content sharing for augmented reality applications. In: 28th {USENIX} Security Symposium ({USENIX} Security 19), pp. 141–158 (2019)

52. Shang, J., Wu, J.: Enabling secure voice input on augmented reality headsets using internal body voice. In: 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), pp. 1–9. IEEE (2019)
53. Shen, Y., Wen, H., Luo, C., Xu, W., Zhang, T., Hu, W., Rus, D.: Gaitlock: Protect virtual and augmented reality headsets using gait. IEEE Trans. Dependable Secure Comput. **16**(3), 484–497 (2018)
54. Sluganovic, I., Serbec, M., Derek, A., Martinovic, I.: Holopair: Securing shared augmented reality using microsoft hololens. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 250–261. ACM (2017)
55. Thys, S., Van Ranst, W., Goedemé, T.: Fooling automated surveillance cameras: adversarial patches to attack person detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 0–0 (2019)
56. Tussyadiah, I.P., Jung, T.H., tom Dieck, M.C.: Embodiment of wearable augmented reality technology in tourism experiences. J. Travel Res. **57**(5), 597–611 (2018)
57. Yew, A., Ong, S., Nee, A.: Towards a griddable distributed manufacturing system with augmented reality interfaces. Robot. Comput. Integr. Manuf. **39**, 43–55 (2016)
58. Zsila, Á., Orosz, G., Bőthe, B., Tóth-Király, I., Király, O., Griffiths, M., Demetrovics, Z.: An empirical study on the motivations underlying augmented reality games: The case of pokémon go during and after pokémon fever. Personal. Individ. Differ. **133**, 56–66 (2018)