Auditable Augmented/Mixed/Virtual Reality: The Practicalities of Mobile System Transparency

RICHARD CLOETE, Compliant & Accountable Systems Group, University of Cambridge, UK
CHRIS NORVAL, Compliant & Accountable Systems Group, University of Cambridge, UK
JATINDER SINGH, Compliant & Accountable Systems Group, University of Cambridge, UK

Virtual, Augmented and Mixed Reality (XR) technologies are becoming increasingly pervasive. However, the contextual nature of XR, and its tight coupling of the digital and physical environments, brings real propensity for loss and harm.

This means that *auditability*—the ability to inspect how a system operates—will be crucial for dealing with incidents as they occur, by providing the information enabling rectification, repair and recourse. However, supporting audit in XR brings considerations, as the process of capturing audit data itself has implications and challenges, both for the application (e.g., overheads) and more broadly.

This paper explores the practicalities of auditing XR systems, characterises the tensions between audit and other considerations, and argues the need for flexible tools enabling the management of such. In doing so, we introduce Droiditor, a configurable open-source Android toolkit that enables the runtime capture of audit-relevant data from mobile applications. We use Droiditor as a means to indicate some potential implications of audit data capture, demonstrate how greater configurability can assist in managing audit-related concerns, and discuss the potential considerations that result. Given the societal demands for more transparent and accountable systems, our broader aim is to draw attention to auditability, highlighting tangible ways forward and areas for future work.

CCS Concepts: • **Human-centered computing** → **Ubiquitous and mobile computing design and evaluation methods**; **Mixed / augmented reality**; **Virtual reality**; **Ubiquitous and mobile computing systems and tools**; • **Social and professional topics** → **Technology audits**.

Additional Key Words and Phrases: audit, transparency, accountability, augmented/mixed/virtual reality, mobile/pervasive systems, audit tooling, Android

ACM Reference Format:
Richard Cloete, Chris Norval, and Jatinder Singh. 2021. Auditable Augmented/Mixed/Virtual Reality: The Practicalities of Mobile System Transparency. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 4, Article 149 (December 2021), 24 pages.
https://doi.org/10.1145/3495001

1 INTRODUCTION

Computing is becoming increasingly mobile and pervasive. Technologies, such as *augmented/mixed/virtual reality* (XR), which interface with our physical environments, are growing in prominence, and fast realising a new range of applications and services, across numerous sectors including entertainment, health, construction, manufacturing and others.

Authors' addresses: Richard Cloete, richard.cloete@cst.cam.ac.uk, Compliant & Accountable Systems Group, University of Cambridge, UK; Chris Norval, chris.norval@cst.cam.ac.uk, Compliant & Accountable Systems Group, University of Cambridge, UK; Jatinder Singh, jatinder.singh@cst.cam.ac.uk, Compliant & Accountable Systems Group, University of Cambridge, UK.

2474-9567/2021/12-ART149
https://doi.org/10.1145/3495001

However, with these technologies comes risk [64], where the nature of such systems mean that incidents can have significant consequences, including damage or harm to people or property [10, 63]. Complicating issues is that incidents in XR need not necessarily arise directly as result of the technical functionality; physical and environmental factors and user actions can also contribute to the occurrence of harms. Moreover, it is challenging to pre-empt and design for the vast range of conditions and states in which systems can operate and in which incidents may arise. This is because systems such as XR are inherently contextual, comprising interactions with various components, and where functionality is tightly coupled with user actions and the physical environment – all of which will directly influence system behaviour [10, 63].

It follows that the **auditability** of XR systems is of key importance. Auditability concerns the ability to inspect how a system operates, how it behaves/behaved, and how it was used in order to better understand *what* happened, and *why*. This often requires capturing information about a system's operation (runtime), including about its wider environment, context and other aspects driving the system, so as to assist an understanding of what led to an incident occurring. Such information aids accountability [11, 70], by supporting review, rectification, repair and recourse; by enabling, for instance, debugging to prevent an adverse incident from reoccurring; apportioning responsibility where harm occurs; monitoring to ensure proper system behaviour (pre-incident); the assessment and understanding of user behaviour; and so on. Moreover, given increasing public concerns and scrutiny of technology, the ability to audit (and thereby better understand, interrogate and critically challenge) emerging technologies will be integral to their acceptance and adoption [58].

Therefore there is a clear need for mechanisms that capture information about deployed systems, so to assist their oversight, monitoring and investigation. Also important to consider, however, are the implications of auditability itself, especially given the realtime (time-sensitive) nature and resource limitations of XR and mobile systems. Indeed, capturing audit data from systems will generally introduce overheads, which can impact the application (to various degrees). Another important consideration is that pre-emptively 'capturing everything', so as to provide the most information for audit, is generally not possible nor appropriate, whether due to technical challenges (e.g., resource or performance constraints) or broader concerns (e.g. privacy, legal limitations and so on). As such, those seeking to implement auditability, whether by building tooling or specifying audit regimes, will need to carefully consider the implications of audit and how to balance various concerns.

This paper focuses on these issues, contributing by highlighting that *there are inherent tensions in auditing XR systems*, and by exploring *how tailorable audit regimes can help in balancing the concerns* intrinsic to audit and risk management. To enable a practical exploration of these concerns, we introduce *Droiditor*, an Android auditability toolkit that enables the customisable capture of runtime information from an app, including its interactions with systems components, the user, and its surrounding environment. We use Droiditor as a means to indicate some potential implications of audit data capture, demonstrate how flexible and configurable audit tooling can assist in balancing audit concerns with application and other requirements, and discuss the potential trade-offs that result. We further contribute by indicating some key audit considerations for the developers of XR systems, and map out areas for future work. Moreover, we seek to support those implementing audit regimes to take a more informed approach going forward. And note that while our focus is on XR, much of what we raise is applicable beyond XR, to other mobile, pervasive and other computing contexts.

Our paper is structured to highlight particular contributions: *(i)* we explore the tensions inherent in auditing XR, arguing the importance for balancing audit requirements with the needs and constraints of applications, the platforms on which they are deployed, and the environments and contexts in which they operate; *(ii)* we introduce Droiditor, an open-source auditability toolkit enabling customisable and highly-configurable data capture regimes, indicating how different auditing regimes introduce different considerations; *(iii)* we demonstrate how configurable audit regimes can assist in managing and balancing audit concerns in practice, investigating their technical implications; *(iv)* we identify a range of broader auditability issues and highlight specific areas for the community to contribute in making technical audit regimes more practical and effective. More generally, we

seek to raise awareness of auditability as an important, yet under-considered area that requires further community attention.

2 THE ROLE OF AUDIT

Some XR applications will operate in 'high-stakes' or consequential scenarios; sectors such as health, transportation, and construction being prime examples. Things can (and will) go wrong, meaning XR has real potential to cause loss or harm, be it economic or data-related, or to persons or property [10]. However, even seemingly benign applications can present risk; for instance, a popular AR game has been contributed to injuries and death, by way of distraction, leading people towards inappropriate locations, and so on [61].

In previous work, we have described how problems in XR can manifest in a number of ways [10]. For instance, there may be issues with the *applications* or components themselves, perhaps due to bugs, poor implementation or testing. Incidents might also arise through improper *use*, such as human error or misuse (deliberate or inadvertent), or *environmental factors*, where the physical surrounds present situations that are not properly accounted for. The *composite nature* of systems can also bring concerns, where interactions with external components, such as a miscalibrated sensor or the down-time of a remote support system can lead to issues.

In all, a variety of issues can manifest in XR, and in practice, will often involve a combination of factors [10]. It follows that there is a clear role for **audit** to help in *understanding what happens/happened* within systems, and *why* [55, 70]. The purpose is to support general oversight, to ensure appropriate system behaviour, and also to identify corrective measures when issues do occur. These aspects will directly impact the technology's uptake and adoption, and relate to broader concerns regarding trust and accountability [71]. In this way, audit will be of increasing importance, not only as XR systems continue to be deployed in 'high-stakes' environments, but as they increasingly pervade daily life.

2.1 Auditability

Auditability concerns the ability to monitor, oversee, investigate, evaluate and interrogate what is happening or has happened within a system, and how the system operates and is used. It entails *capturing information* about the system and its wider operational contexts so as to give greater visibility over its behaviour and its drivers. This works to assist with understanding the system, its effects, and consequences [11, 45, 70].

Auditing can help to ensure systems are built, deployed, tested, and operate appropriately. In the event of an incident or failure, audit mechanisms can enable *ex post* review as to the causes and contributing factors. This is by collecting data such that an incident can, to some degree, be 'reconstructed' so as to support investigation and corrective actions. For example, if a user were to injure themselves (or harm someone/something else) audit data can reveal what the user saw, heard, their actions and responses, the system state, functionality applied, and so on. Such information can help identify and rectify the issue and prevent recurrences, uncover the components or stakeholders involved or at fault, and help in determining what (if any) recourse is appropriate.

Means enabling audit can also provide *ex ante* benefits (i.e. before an incident) by enabling oversight. This can involve monitoring in order to verify correct, expected and appropriate system status and behaviour, and alerting of situations warranting attention or of possible concern. This allows for more proactive measures, e.g., to detect a potential failure before the incident occurs, allowing preventative intervention.

The general aim of auditability is to increase transparency, where relevant information about a system can assist with accountability by supporting oversight and scrutiny, rectification and recourse [11, 45]. Indeed, the ability to monitor happenings, and 'unpack' situations of failure aligns with societal calls for greater transparency and accountability regarding technology [58]. Auditability works towards supporting such.

2.1.1 Benefits for Stakeholders: Importantly, auditability also works to support particular stakeholder aims [55]. For *developers*, auditability can help produce better software through information that can assist the design, testing, and debugging of applications. Audit information can also help ensure that applications are meeting their

specification, interactions with other components are proper, that they are being used appropriately, give insight into particular contexts and environments of use, generate more data and cases to assist testing, among others.

Auditability can also help developers show they are taking their (legal and corporate) responsibilities seriously, e.g., through monitoring for risks and potential emerging problems, and demonstrating that mitigations are employed. Indeed, record-keeping obligations are increasingly part of regulatory regimes (see below). When failure does occur, audit data can provide evidence that may help absolve responsibility and limit reputational damage; e.g., by showing the incident was due to factors beyond their control (human error, third-party service failure, etc).

Regulatory and oversight bodies also have an interest in auditability measures. Audit data helps regulators conduct their oversight and investigatory activities, in order to check for legal compliance, determine who might be at fault and apportion responsibility, issue penalties, etc. Moreover, increasing levels of technical transparency improves visibility, and can therefore help regulators prepare for emerging issues, define best practices, and other efforts to raise overall standards; while also discouraging poor organisational practices. Note that auditability is increasingly becoming legally mandated, with record-keeping requirements featuring in recent and upcoming regulatory frameworks (e.g., [22–24, 74]).

Users also stand to benefit from increased auditability. Meaningful information about a system can allow users (or those otherwise affected by systems) to better understand what a system does, and therefore how better to use and interact with it. It can also empower users by better enabling them to interrogate, scrutinise and challenge the applications on which they rely. In commercial contexts, organisational users can use audit data for quality assurance, to identify areas where staff need training, etc.

3 Auditable XR

We have outlined how auditability is a concern of increasing importance. We now describe how auditability is particularly relevant for XR.[1]

XR collectively refers to a range of technologies: Virtual/Augmented/Mixed Reality. These technologies generally work to blend the digital and physical environments, providing users with novel and immersive ways to interact. We see XR increasingly being used in a range of contexts, including for entertainment [37], education [31, 67], health [2], construction [84], retail [60, 73], military [34] and manufacturing [54].

In short, *Virtual Reality* (VR) fully immerses the operator's view with virtual content but involves physical interactions in the space [68]; *Augmented Reality* (AR) integrates digital content over a view of the physical world [8]; and *Mixed Reality* (MR) entails a tight integration of the digital and physical world by more widely incorporating a range of devices, often integrating environmental sensors and having direct physical effects through actuators [32]. Note that the lines between these delineations blur [51], and many issues (including audit) apply generally across these.

With any technology comes risks. However, as we have previously argued [10], XR's strong coupling of the digital and physical environment, and the interaction modalities it provides, raises particular challenges. First, as discussed, the physical nature of XR means there is real propensity for harm, to people, property, etc. This is not only in 'high-stakes' scenarios, but also those which appear benign (§2); indeed, such risks are directly acknowledged by XR device manufacturers [35, 49, 56], and as mentioned, even basic XR games have resulted in harms [43]. Moreover, XR functionality is directly informed by the physical environment (and *vice-versa*). This makes it difficult for developers to foresee, *a priori*, all potential issues that can occur, let alone correct for such [10]. This is especially given that many XR applications involve mobile devices, meaning they can potentially be used 'anywhere' a user takes their device. Further, XR systems are composite, in that they involve interactions across a range of components (e.g., device sensors) and external services (e.g., cloud, web services).

[1]Note that though we focus on XR, much of what we raise in this paper is relevant to mobile and pervasive systems in general.

In short, there are a range of aspects requiring attention. In this way, the complex, context-specific nature of XR systems, and their potential for direct physical consequences, motivates the need for mechanisms that facilitate and enable effective auditability.

3.1 Auditing for XR

From an audit perspective, it is often useful to capture information that provides visibility over, or some form of recreation of, what occurs. Though in XR, many risks are contextual and can manifest in different ways, making it sometimes difficult (or impossible) to know in advance what should be captured [10]. Incidents can arise from the application, device, the physical environment, the user and their actions, interactions with remote services, etc. Therefore, in addition to that application-specific information, information about an XR application's broader operational context (such as the device's 'I/O') is needed to provide a more holistic view of the context in which an incident occurs.

We now elaborate three key areas, as elaborated in our previous work [10], of data relevant for XR audit: that regarding user experience; interactions with the physical environment; and details regarding the supporting components.

3.1.1 User Experience: XR is user-centric. Risks can manifest resulting from that presented to the user, such as content on the display, audio generated, haptic feedback, etc. For instance, there is potential for digital material to *obfuscate* vital information from the user's physical environment, whereby obstacles, alarms and so forth might be obscured (see Fig 1), or *misdirect*, whereby the instructions provided to the user could be misleading or incorrect [4, 63]. Similarly, what is presented might be a contributing factor to an incident, should the content operate to *distract*, e.g., the AR game previously mentioned which led to physical harms [61], or reflect problematic experiences in line with *online harms* [63, 79], where bullying, harassment and other inappropriate behaviour and material have been reported in social XR contexts [6].

Therefore, information regarding what a user experiences, and how they interact with the application, is fundamental for audit – by helping to indicate the existence, or potentially the causes of an issue, and to otherwise give some insight into what occurred. In some cases, information regarding what the user directly experienced (the display, audio) may suffice, e.g., for providing evidence of harassment. In other cases, a combination of audit data sources may reveal an issue, e.g., comparing the display (which includes visual overlays) and the raw camera feed (as Fig.1 from [10] illustrates) could help identify situations of obfuscation or distraction. Also relevant are the actions the user performs, such as gestures, speech, screen taps and other inputs, as this helps reveal how the user behaves, reacts, and responds to particular events. User actions can also provide evidence of appropriate or inappropriate system usage, as well as indicate the effects or impacts of an issue, e.g. did an incorrect instruction (misdirection) actually result in the user taking improper action, and so forth.

Fig. 1. Digital overlays can obfuscate critical information, such as manholes or the view ahead [10].

3.1.2 The Physical Environment: As discussed, XR systems are tightly coupled with the physical environment. This is through a range of sensors (e.g. cameras, microphones, and many others sensor types) and actuators which perceive and interact with the physical environment. In this way, the environment drives (in part) the

behaviour of the XR system and the operator. Similarly, XR systems can impact the environment directly through actuation, or by influencing user behaviour.

This can lead to various issues. For example, environmental conditions may be *incorrectly perceived* and thereby lead to erroneous behaviours, e.g., where a faulty light sensor incorrectly dims the display, potentially leaving the user in a precarious situation, or a miscalibrated GPS leads to incorrect overlays being presented. Similarly, components *processing* sensory inputs can result in issues, for instance where an object recognition system fails to properly identify a relevant feature, such as a hazard or the correct part of some machinery, thereby driving inappropriate actions; or where environmental factors hinder the application, e.g., where a passing siren renders voice commands ineffective. *Improper actuations*, whether a direct result of the above or due to other issues, may cause inappropriate physical effects. The consequences for these can be significant, e.g., where machinery is damaged as a result of a bad command.

Again, it will be important to have records of how the system interacts with the environment to help build a picture of what occurs. An auditor might consider whether the various sensory inputs and outputs (actuations) aligned with expectations. Uncovering issues might entail comparing and contrasting a range of data sources, such as the brightness sensor readings with the camera feed when considering the display's dimming, checking that an actuation command had the appropriate effect through capturing audio-visual information (camera/sound), or by providing some detail to allow validation or investigation with an external source.

Moreover, beyond an application's direct interactions with the physical environment, details of the broader physical environment in which the application is being used can provide useful information of its operational context. That is, capturing this *surrounding information* might be relevant for audit, even though that data is not directly used by the application. For example, a navigation app might not utilise the microphone, but its (audio) feed could capture important cues or distractions in the environment, such as alarms, horns, sirens, etc.

3.1.3 Component Interactions: XR typically encompasses a range of components. While some might be tightly integrated and/or under the control of the application, such as those on-device sensors and actuators, there will often be interactions with external components (such as cloud storage and databases, web services, dynamic device interactions as one moves through a smart building, and so on).

The reliance, dependencies and interactions between components can also present problems. For example, issues may be caused by component *misbehaviour*, such as a faulty or miscalibrated sensor. Some components may become *unavailable*; e.g., an app relying on a remote database to present information on the user's display, where a lack of database availability renders the app unusable or dangerous. Some interactions might become *incompatible*, where, for instance, issues might be caused by a service update that changes a protocol or data format. Indeed, even where all components are operating correctly, the orchestration of components in pervasive computing contexts can lead to *emergent properties* [70]. That is, the system might exhibit unforeseen or unexpected system behaviour once deployed as a result of the interactions between various components. And failures can have 'knock-on' effects, where an issue in one system may cause issues in another – which can *impede* investigations by obscuring sources and causes.

It follows that collecting data regarding the (attempted) interactions between components can help hone in on the origin of failures, give insight into the factors contributing to an incident, as well as indicate potential issues or the development of emerging properties [70]. For instance, such information can reveal issues of service availability, such as a failed connection or slow response. Audit data on components helps their validation (e.g., was that received of an expected range and format), and indicates what drove system behaviour – did the sensor reading lead to a misdirection? Did the data come from the right sensor? Moreover, records of interactions with components also helps indicate who might govern or maintain them, which provides more points useful for investigation, and can also help in apportioning responsibility.

3.2 Towards Auditable XR

The above represents a few examples to illustrate the breadth of potential issues in XR systems, and how audit data can assist. The three categories described are simply to indicate that various concerns may be raised; these aspects are interrelated, and in practice, the same audit data may be relevant to these and a range of other issues.

The key takeaway is that *auditing an XR application will often concern information pertaining to (i) system operation, (ii) the application (and device-oriented) I/O, and (iii) the manner and broader contexts of its use.* The ability to review contextual and environmental information about an application's operation (from a broad range of internal and external data sources) can assist in recreating the steps leading up to (and following on from) an incident, helping auditors, developers, users and other interested parties to build up a more complete picture of the events in question. And again, while such concerns are particularly pertinent for XR, given XR's contextual nature and its potential to directly realise tangible harms, we reiterate that many of these are also relevant for pervasive, mobile and other technologies in general.

4 THE NEED FOR AUDIT TOOLS

Given the importance of auditability, there is a clear need for mechanisms that enable audit—that work to capture information about system behaviour so as to facilitate audit—to be built into systems *'by design'*.[2]

Auditability, as considered in the context of transparency and accountability, is something of a nascent, yet growing topic for the technical disciplines, and has had some initial consideration in areas including cloud [72] and AI services [40, 41], the Internet of Things [53, 81], and machine learning [21, 25, 33, 52, 65]. However, generally it is an area warranting further attention [11].

In an XR context, aside from [10] which (as just discussed) argues a general need for auditable XR, the practical dimensions to XR auditability have so far had little consideration. There is some work on capturing information from XR systems to, for example, support system development [28, 36, 62], and to record user actions for training [66], to retain employee knowledge and expertise [26, 42], for maintenance [12, 69], and for health and safety incident detection [83]. However, in short, these either aim at supporting system design (debugging), often entail substantial developer intervention, and/or focus on narrow/specific issues or application contexts.

Indeed, it has been shown that there is little in the way of tooling for auditability in a general sense, to support a broad range of audit concerns. A recent survey we conducted confirms this lack of appropriate audit tooling [10], and also reveals that for those (few) developers who do consider broader audit issues, often they have little choice but to implement their own bespoke audit tools. Naturally this leads to inconsistent, incompatible, and haphazard approaches to collecting audit data – the opposite of what is required for supporting effective and holistic auditing regimes.

As such, there is a clear gap and need for tools that can support a range of auditability aims. Particularly useful are those that are generic and customisable. That is, auditability is contextual, especially for XR systems, and so tooling that can be tailored to meet the specific needs of a given (and potentially dynamic) scenario will help ensure that the 'right things' are captured in the 'right way'.

However, important is that capturing data can itself have implications which impact the application, such as those regarding performance, storage, and energy. As the next section describes, this could be problematic, potentially causing applications to be difficult to use, malfunction, or worse, fail entirely.

5 AUDITABILITY: A BALANCING ACT

As we have discussed, data is central to auditability. Data about systems operation enables situations of interest or concern to be unpacked, thereby supporting monitoring, review, repair, investigation and recourse. However, the processes of audit itself will present challenges and considerations that may have consequences. For instance,

[2] In a similar manner to requirements for privacy-by-design, security-by-design, data protection-by-design, etc.

capturing data for audit may impact an application's functionality and operation, have resource implications, and can raise privacy concerns.

This means that in practice **auditability entails a 'balancing act'** – to ensure that the sufficient and appropriate information for supporting audit is available, while accounting for the range of externalities that an audit regime may bring. This section highlights some key tech-oriented tensions associated with the run-time (operational) capture of audit data from XR systems. We illuminate some ways in which the challenges can manifest, and argue the need for means supporting the balancing of audit/application concerns and requirements.

5.1 Performance

Capturing audit data will generally impact system performance. This is because retrieving and recording this data entails additional processes and resources beyond those of the XR application's core functionality.

The additional processing that an audit regime entails can potentially impact the app's operation. Given the real-time nature of XR apps, there is a real risk for heavyweight audit regimes to significantly impact the app such as to render it wholly unusable. Though even smaller overheads that, for instance, introduce a slight lag or subtle delays can distract, physically disturb (e.g., inducing nausea, a well-known XR concern [82]), and generally cause frustration. Moreover, performance impacts can result in synchronisation issues, that might affect the positions of overlays, for example. These considerations may impede an operator's ability to perform tasks, cause mistakes and errors, potentially with serious consequences.

In this way, the need (or desire) to capture certain audit data and its implications on system performance (and thereby usability, safety, etc.) requires consideration.

5.2 Data

XR systems can produce vast amounts of data relevant for audit. This can include data from inputs (sensors, cameras, microphones, etc.), outputs (displays, speakers, etc.), and the applications themselves (representations of state, environmental maps, etc.). Audit regimes may seek data from a variety of sources, encompassing a range of inputs, outputs and system state, which can be at high-levels of detail. For example, in XR, often what the operator 'sees' is important, which might entail capturing every frame the system produces for display (for AR this is often 30–60 frames per second); or details of some sensor stream might be relevant, where the sensor produces readings at 1000Hz; and so on.

This means that in practice audit can entail the capture of large volumes of data. This has storage implications, particularly for XR systems as they are often mobile, where devices generally have limited storage capacity (e.g., Microsoft's HoloLens 2 has 64GB on-board [50]). Where storage is depleted, applications may be affected, potentially resulting in the inability to undertake important functionality such as persisting data, decompressing files or textures, and in the worst case, could cause the application to crash. Naturally, this has consequences for audit, in that the lack of storage may lead to audit data being lost, incomplete, or corrupted, depending on the implementation and circumstances.

An alternative to storing data locally is to offload it to remote storage. This, however, raises other considerations, including those regarding the network, in terms of bandwidth consumption, network coverage and stability (given XR devices can be mobile), performance, possible data fragmentation (where audit data might be split across several locations), among others. Other considerations around data also exist, including those regarding the nature of the data itself; e.g., privacy, security, access, etc. (see §5.4,8).

In all, there will need to be a consideration of the nature of the data required for audit, and its level of detail, with respect to the practicalities of its storage, management and other concerns.

5.3 Energy

The capturing, processing, transfer, writing and management of audit data will also impact energy consumption. This is important particularly where XR systems are mobile and battery-powered, which will mean that operators have limited time in which to complete tasks (Microsoft estimates that its HoloLens can sustain ~2-3 hours of active use [50]). A consequence, therefore, is that making an application auditable may result in a shortened usable device time.

This is an important concern given this affects the time in which the application is operational. This could lead to situations where there might be insufficient time for an operator to complete a task [77], which in turn could not only have business and financial implications, but lead to risks of harm to the operator, to bystanders, to property, and so on. Untimely shutdowns might also affect application and audit data, potentially through corruption or data loss.

Again, the requirements for audit data need to be considered with regards to the additional energy requirements incurred and any associated risks thereof.

5.4 Privacy and Confidentiality

Importantly, XR audit data will often relate to people (operators, bystanders, etc), their behaviours, and their surrounds. This has privacy and confidentiality implications. That is, through cameras, microphones and a range of other sensors, audit data may describe precise movements, locations, where attention is focused, interactions with other systems and people, etc. Here, much data will be personal, and may reveal underlying health conditions [9, 47], likes/dislikes [78], locations of homes or workplaces [5], etc. Data may also present a security risk, perhaps by containing usernames and passwords, bank details, revealing sensitive business processes (security protocols, IP, trade-secrets), and so on [14, 44]. And importantly, data protection law [24, 74] might impose certain obligations and requirements regarding data that is personal.

Naturally, this raises tensions, where the collection of certain information may be deemed important or necessary for supporting audit, but also raises privacy, data protection, and confidentiality concerns. Again, the concerns are contextual, and various technical and organisational measures [24] might be employed to help balance these. However, note that the technical mechanisms employed to help balance these privacy and confidentiality risks themselves will likely have consequences. For example, encrypting or perturbing data can have performance implications. Moreover, selectively capturing or perturbing the data captured can result in important information being lost, thereby hindering specific audit aims – for instance, a recent court case discusses how deleting the images of people 'irrelevant' for the system also hinders the ability to test for discrimination [57].

The sensitivity of data will be important when deciding which audit strategies to adopt, which will need to be considered in relation to the requirements of the application, audit, and other concerns.

5.5 Balancing Concerns

Capturing contextually-relevant information (audit data) [11] about the nature and operation of a system will be important for unpacking situations of interest. However, as outlined, this itself will have overheads which could have consequences for the XR system, the running application, the collection of audit data, the safety of operators or bystanders, and so on. There is therefore a real risk *that audit regimes can themselves contribute to issues.*

In practice, this means there will often be a need to balance audit data requirements with application constraints, performance requirements, resource limitations, privacy, confidentiality and other concerns. Tooling that provides flexibility over what, when, and how data is captured can help manage these concerns. That is, tools that enable the tailoring of audit regimes to accord with the constraints of the circumstances—whether application-specific or otherwise—can provide a means for helping to balance audit requirements. This may be by allowing audit regimes to be configured in ways so as to only collect certain data (from certain sources), or to reduce the frequency of what is recorded, and thus reduce overheads. Similarly, the level of detail (granularity) of captured data can be

reduced; e.g., images could be captured at a reduced resolution or colour space, audio could be captured with a low bit-rate, or summary statistics over data streams could be captured rather than raw sensor values. The ability to dynamically control audit regimes also appears useful, by perhaps enabling more comprehensive audit data to be collected as situations become more 'risky', by reducing audit coverage as and when a risky situation resolves, or to adjust levels of recording in line with resource availability (such as available battery).

Naturally, the precise audit regimes and configurations employed will depend on the specifics of the XR system, the audit requirements, and the contexts in which the system is deployed and used. As such, having methods that enable the flexible configuration of data capture regimes will be important. Further, an indication as to what impacts (e.g., energy, performance, storage, etc) can be expected from various capture mechanisms can go some way in helping practitioners take better informed approaches to auditability, and balancing practical concerns.

Towards this, we now introduce Droiditor—our Android-based auditability toolkit—which we later use to indicate the potential implications of auditability and how concerns can be balanced in practice.

6 THE DROIDITOR AUDITABILITY TOOLKIT

We have argued that capturing data from an XR application will often be fundamental to unpacking how a system operated and how it was influenced and used. However, as the previous section outlines, auditability has a range of practical considerations that require balancing.

Towards this, we now introduce *Droiditor*, a flexible auditability library for Android. Droiditor allows for tailored audit regimes that passively capture a broad range of information to provide insights about an app's operational context. The library was designed to be configurable so as to provide control over what, when, and how relevant data is captured. Drioditor is open-source and is available at https://github.com/compacctsys/Droiditor.

We use Droiditor both to illustrate how flexible audit tooling—enabling the control and tailoring over what, where and how data is captured—can assist in supporting audit and balancing various concerns (see §5); and as a way forward in addressing the gap in tools (see §4) for capturing information relevant for auditing XR (and indeed, other mobile) applications. First, however, we describe Droiditor, its setup and configuration, and its operation.

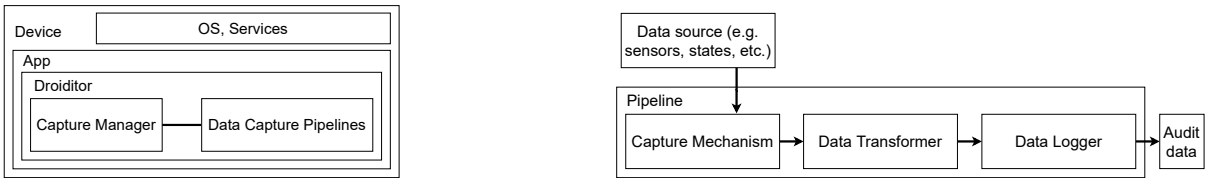
6.1 Drioditor Background

In developing Droiditor, our aim was to provide tooling to enable applications to become auditable, by enabling the means for *(i)* capturing information about the broader contexts in which an application operates, and *(ii)* balancing audit information requirements with other aspects, such as application performance. Note that Droiditor is an example of a tool that works to *facilitate* audit. It does not determine what data should be captured for a particular scenario, as that is very much dependent on the application and other factors. Instead, Droiditor provides an accessible means for making mobile applications auditable, by enabling the flexible and passive capture of a wide range of potentially relevant audit data.

We targeted Android as it is a key platform for XR applications, and as a mobile platform, it is one where the user's physical environment and actions (e.g., using XR while walking through a city) will play a key role in determining app behaviour. Moreover, Android is the major mobile operating system by market share [27], currently running on billions of devices [7], across a wide-range of application types. Android is therefore a natural choice for demonstrating the role and potential for a general and holistic audit regime, that has wide applicability across XR applications, and other applications as well.

6.2 Architecture

In essence, Droiditor operates to create *audit data pipelines,* where information is captured from the application's (and device's) operational context, and recorded as appropriate. Fig. 2a depicts Droiditor's high-level architecture, while Fig. 2b illustrates an overview of a data pipeline. Data capture pipelines consist of a *Capture Mechanism*,



(a) High-level architecture (b) Audit pipeline illustration

Fig. 2. Overview of Droiditor's modular architecture.

which have attached *Data Transformers* and *Data Loggers*. The Capture Mechanism operates to capture the audit data from a data source (e.g., a sensor), which moves through a Data Transformer (if specified), before being recorded (outputted) by way of the Data Logger. The architecture is designed to be modular, configurable and extensible, such that each of these components can be custom-defined, and dynamically loaded/unloaded and configured to provide flexible audit pipelines.

To elaborate, the *Capture Mechanisms* work to capture runtime data for audit from the particular data source. Table 1 describes those we currently support by default, which broadly represent the main forms of standard Android I/O (most implemented using Android services). However, the architecture is flexible and extensible in that new capture mechanisms can be defined, and we do have extensions planned. Importantly, when enabled, capture mechanisms will collect data regardless of whether or not the running application directly uses that data (e.g., an app might not use the microphone, but AudCap (see Table 1) can still be used to record from the mic). In this way, Droiditor can help to capture contextual information beyond that of specific app functionality, which as discussed can be relevant for audit purposes. Each capture mechanism (and thus its associated pipelines) can be independently enabled, disabled, and configured, giving flexibility over what is captured.

Table 1. Droiditor's currently available capture mechanisms and their descriptions. Note that Droiditor is highly extensible, allowing for new capture mechanisms to be defined and customised.

Capture Mechanism	Description
ScreenCapture [ScrCap]	Records the display as a video.
SnapShot [ScrShot]	Used to take screenshots of the display.
CameraCapture [CamCap]	Captures frames as perceived by the camera, at the rate they are available. These are 'raw' frames, i.e. without digital overlays (cf. the screen for AR).
AudioCapture [AudCap]	Captures sounds produced by the device and surrounding environment [3].
SensorCapture [SenCap]	Captures data from specified on-device sensors. (excluding those audio/camera related which are dealt with separately)
NetworkCapture [NetCap]	Records details of network connections (e.g., addresses) and of data transferred
ControllerCapture [ConCap]	Records actions of the 'controllers' with which users interact with the device.

The next stage of the data pipeline concerns *Data Transformers*. Transformers are modules, that when defined, can be assigned to a capture mechanism to perform some processing of the data stream that the capture mechanism produces. This can include, for example, changing the image or video quality captured, selecting or filtering particular data (e.g., sensor readings above a threshold) or otherwise aggregating and summarising streaming data, perturbing data for privacy reasons (e.g., masking faces), and so on.

Data then moves to a *Data Logger*. The loggers work to record (i.e. 'output' or 'write') the data from the system. Droiditor currently includes two default loggers; one that writes data to disk, the other streams the data over the network to a remote service. These default loggers record video as MP4 and audio as 3GP, while images

are recorded as JPEG – all of which are written in an asynchronous manner. More stream-oriented data (such as for SenCap, ConCap) default to writing to JSON files for wide-compatibility. Again, custom logger modules are possible, for example, to output data in different formats or handle more complex and customised output arrangements.

Drioditor is driven by the *Capture Manager*, which is responsible for setting-up and enabling the audit data pipelines, including the loading, registering and unloading of various modules (the capture mechanisms, loggers and transformers), and starting and stopping their operation (i.e. the recording of the data).

The Capture Manager is driven by audit configurations, which describe the audit data pipelines and their operation. The Capture Manager controls these configurations which may be defined and loaded from a file, set programatically, or through a network (remotely accessible) API. This provides flexibility, in that it allows audit configurations to be set at design-time or otherwise pre-defined and loaded as appropriate, as well as dynamically at runtime (through application code, or as instructed from a remote entity) so as to enable 'on demand' adjustments over what is captured – which might be, for instance, to account for a change in circumstances, where a certain event or occurrence might warrant recording more information.

In this way, Drioditor's architecture provides flexibility and enables customisation over what, when, and how audit data is captured.

6.3 Set-up and Operation

Developers integrate Droiditor by importing the library and then making two API calls: one to initialise Droiditor with a reference to the app, and the other to begin capturing audit data. Audit pipelines are configured according to a specified configuration (reverting to a default configuration if one is not specified).

Note Droiditor does not require a rooted device, and is consistent with Android best-practice. This, however, means that because Android's permission's model considers monitoring screen contents and the network to be higher risk ("dangerous") [16], the NetCap and ScrCap mechanisms, requires invoking the (standard) Android procedure for obtaining user permission (at runtime). In this way, Drioditor accords with Android's permission models, which also means that users also have a degree of awareness and control over what is captured. Note, however, that this does not itself ameliorate privacy risks, as we discuss elsewhere.

During operation, Droiditor runs in the background, passively capturing and recording data in accordance with the data pipelines defined by the *audit configurations*.

7 EVALUATION: CONFIGURABLE AUDIT FOR BALANCING CONCERNS

As described in §5, auditing often entails a balancing of various considerations. Here we provide insight into how audit tooling can assist, by allowing audit regimes to be tailored and customised to particular concerns, while at the same time giving a sense as to what might be expected by capturing different audit data, as a means for generally bringing about a more informed approach to audit.

Specifically, we use Droiditor to consider the impacts of different capture configurations, as a means for indicating the considerations of audit. We first illustrate the implications of a 'comprehensive' (or detailed) audit data capture regime that attempts to capture as much information as possible. We then consider more tailored audit configurations—including with regards to the impact of individual capture mechanisms, the processing of audit data streams, and the offloading of audit data—to show how such functionality can operate and assist in balancing concerns with the need for information supporting audit, and the associated implications.

7.1 Experimental Setup

We use Google's HelloAR [19] as an exemplar XR application for capturing audit data and exploring the impact of auditing on performance under various conditions and configurations. HelloAR encapsulates a range of typical XR functionality, such as leveraging on-device cameras (to provide a view over the physical-world), gyroscopes

(to orientate the XR device), performing plane-scanning (to detect surfaces and boundaries), image processing (to overlay digital annotations), etc. Moreover, HelloAR uses ARCore (a common XR library [29]), and reflects recommended Android XR design practices.

We instrumented HelloAR with Droiditor, and extended the app to include additional functionality, so as to better enable us to explore the implications of audit. This included having the app continually receive data (a stream) from over a network, given many XR apps will receive data from external systems, and to continually play an audio file, given that XR apps often generate sounds, e.g., voice instructions, to convey information, etc.

Experiments were conducted on a stock (unrooted) Samsung Galaxy Tab S5E (Octa Core [Dual 2.0GHz + Hexa 1.7GHz] CPU, 4GB RAM, 64GB storage, 10.5" 2560x1600 display) running Android 10. All execution runs were initiated by Android Debug Bridge (ADB), ran for 60 seconds, conducted under consistent usage conditions: the device was set in an upright position, in portrait mode, and remained stationary with the rear camera pointing at a common target, in consistent lighting, with sufficient battery and the device reset between runs.

Each experiment was performed 20 times. System aggregate CPU and RAM usage, mean system current draw (current/energy), and the frames per second (FPS), were all measured to provide a broad view over the impacts on performance and app behaviour. For the latter, FPS is a key metric [30], given lower frame-rates can cause various issues, such as degrading user task-completion performance, inducing nausea, or rendering apps inoperable to varying degrees [13, 15]. Perfetto [3] (a performance instrumentation toolkit) was used to measure CPU, RAM usage, and current power draw (energy usage). Storage requirements were calculated by measuring the directory size of the specified audit data folder.

7.2 Performance Impacts

We first consider the performance overheads of employing a heavyweight capture regime (i.e. enabling the range of capture mechanisms). We then explore the impacts of the capture mechanisms individually.

7.2.1 'Full' Audit: Often audit requirements will mean that several capture mechanisms need to be enabled simultaneously. It is therefore important to understand the performance profile and resource requirements of a 'fuller' (or detailed) capture configuration; that is, one that attempts to capture as much information as possible. Not only does this provide us a baseline, but importantly, represents a scenario that attempts to allow as much of the application experience to be reconstructed as possible, by providing much detail about what has occurred. As such, we first explored the combined effects of all capture mechanisms (per §6.1) enabled, except for that of ScrShot (as often it will be unnecessary to both record the screen (via ScrCap) *and* take continuous screenshots).

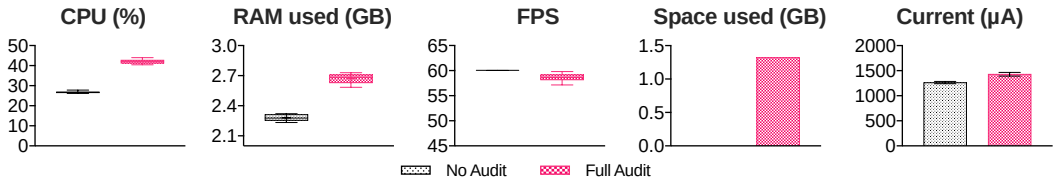


Fig. 3. The impact of the combined capture mechanisms. Error bars (in all figures) represent 95% confidence intervals.

Fig. 3 presents the performance and storage requirements of the unaudited app and that of combined audit. We see that with audit enabled, CPU usage is ~45%, RAM ~15%, with a negligible impact on FPS (60 FPS is the device's maximum). Energy usage (current) increased by about ~15%, while ~1.3GB of data was produced per 60s run.

7.2.2 Per-mechanism Impacts: Naturally, some capture mechanisms will be more resource-intensive than others. As such, we now explore the overheads associated with different capture mechanisms. In addition to indicating

the implications of particular audit data sources, which can help those specifying their audit configurations to balance any associated implications, these results also help demonstrate the practicalities of taking a more informed approach to audit.

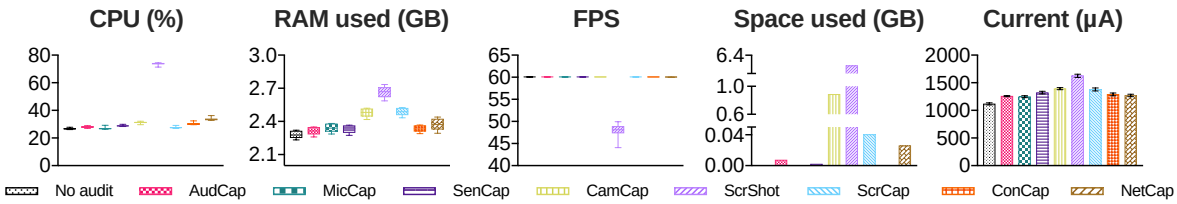


Fig. 4. Impacts of individually enabled capture mechanisms on performance and storage.

As Fig. 4 shows, all capture mechanisms introduce some overheads. However, we observe those visually-related (screen/camera) capture mechanisms realise the largest overheads in terms of RAM, storage and energy. ScrShot[4] brings particularly significant overheads, increasing CPU usage by ~3-fold, RAM by ~400MB, current by ~50%, and used ~6.3GB of storage. ScrShot also had a notable impact on frame rate, which dropped to ~48 FPS. Unsurprisingly, we also observe that the more significant impacts are on energy usage and with storage, again particularly for the visual information.

Though the above (both §7.2.1 and §7.2.2) represents but one experimental context, these results seek to indicate the potential overheads of auditability and the implications of leveraging particular audit data sources. While capturing audit data does entail overheads, from a usability perspective we observed that the app remained fully functional with no perceptible effects of audit during manual testing (noting that many AR apps run on devices at 30FPS, and so even a drop to ~48 FPS when using ScrShot was imperceptible to us for this application).

Of the various capture mechanisms, those recording visual data (ScrCap, ScrShot, CamCap) appear the most expensive. Expectedly, but particularly notable for a mobile context is that all of Droiditor's capture mechanisms *increase energy consumption*, as well as generate a substantial *volumes of data*; in generating ~1.4GB/min on our device, the app with combined 'fuller' audit (per Fig. 3) can be sustained for only ~45min until storage is exhausted, or 140 minutes until the battery is depleted (~7% per 10min).

Ultimately, whether performance and resource usage is an issue depends on context. For instance, if a device is running low on resources (perhaps due to a particularly compute-intensive app), the overheads of recording audit data may well push the system past a 'tipping point' in which functionality is hindered. Indicating the potential impacts of auditing overheads—as the above results do—assists developers in determining the mechanisms and ways by which particular audit information is captured, informing by providing a practical illustration as to the implications of different approaches.

7.3 Audit Configurations

The above results indicate that capturing data for audit entails (some) overheads. Though one might be tempted 'capture everything' so as to provide the most detail should it be needed, in reality, the above shows that often this is impractical (in addition to often being undesirable [75] – again, audit can have many consequences (§5)).

Therefore, to facilitate the balancing of audit with other concerns, tools supporting auditability must be highly configurable, enabling an audit regime to be aligned with the various considerations as appropriate for the context: be they application or performance requirements, device limitations, legal obligations, etc.

As §6 describes, Droiditor was designed with this in mind, enabling the flexible customisation of the audit data pipelines in order to *(i)* select the audit data sources (capture mechanisms) – as just discussed, *(ii)* determine the

[4]Note the ScrShot mechanism uses Android's *PixelCopy* functionality [17], the recommended method for capturing screenshots [20].

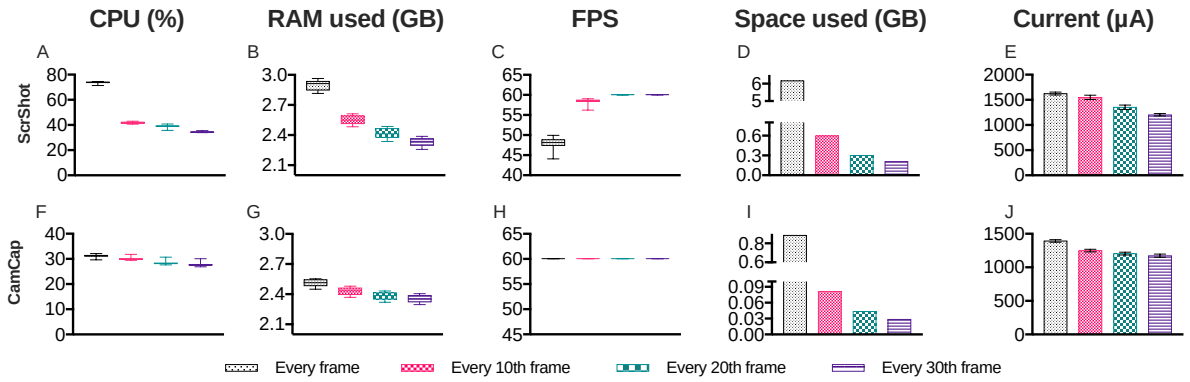


Fig. 5. Capture frequency trade-offs. Data captured every frame, every 10^{th}, 20^{th}, or 30^{th} frame.

nature of the data recorded from these capture streams (transformers), and *(iii)* control how data is outputted (loggers). Such functionality enables for some management of the overheads, but importantly, also allows for a finer, more nuanced definition as to the audit data captured, which assists the balancing of concerns.

In this way, this section explores how audit configurations provide flexibility in terms of *what* data can be captured, and also in the specifics of *how* it is captured. We now further consider such functionality both to indicate its implications and potential for balancing audit concerns.

7.3.1 Capture Frequency: One configuration option concerns the frequency by which runtime data is captured. In some instances it will be necessary to capture data as frequently as possible (e.g., fast-paced or high-risk contexts such as XR-assisted driving or surgery), so as to ensure a detailed coverage of the events leading up to a moment of interest. In other cases, periodic samples giving an indication of what is occurring may suffice, e.g., perhaps in an XR educational app.

Here we explore the effects of sampling frequency. We focus on the ScrShot and CamCap capture mechanisms, given that visual data was found to have the largest overheads (§7.2.2). We performed experiments where Droiditor was configured to capture (at maximum quality) each frame, or every 10^{th}, 20^{th} or 30^{th} frame.

From Fig. 5 we see that reducing the capture frame rate decreased CPU and RAM impacts for both mechanisms, though this was most pronounced for ScrShot. FPS appears largely unaffected by CamCap which operated at the maximum FPS even while capturing every frame, while for ScrShot, FPS is improved by reducing the capture frequency. Regarding energy usage, we see a general trend towards some reduction in energy usage in both cases (Fig. 5 (E & J)). This is less pronounced for CamCap as the camera is already being accessed (and its feed processed) by the app (see §6.2). Predictably, storage requirements are significantly reduced by lowering how often, and thus how much data is captured, which is important given storage implications can be substantial.

Overall, these results show the relevance and importance of configurable audit – whereby adjusting the capture frequency is one way to balance the detail of audit information with application performance.

7.3.2 Image Quality: To further explore the overheads of capturing visual data, we now consider a situation where the image quality is reduced as a means for lessening the resource implications.

We created a Droiditor data transformer that uses image compression to adjust the quality of the images captured. This functionality allows image quality to be tailored to particular situations; e.g. where image quality is less important, such as an XR-training app, images can be significantly compressed to reduce overheads, c.f. situations where image detail or colour accuracy is important, such as in a medical scenario.

Fig. 6. ScrShot with compression employed (quality=100 vs quality=10).

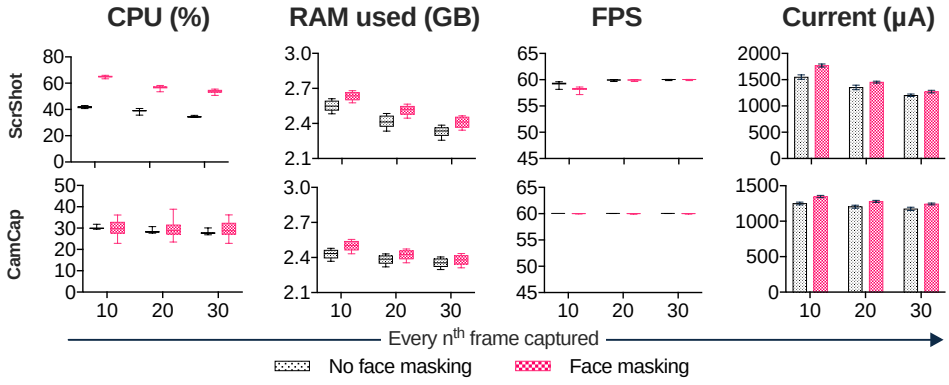
As Fig. 6 shows, higher compression (and therefore a reduced image quality) lowers the energy and storage impacts. Images at *quality=100* were ~1.8MB in size, while at *quality=10* they were 74KB. This represents a significant reduction in storage requirements, but also with a degradation in image quality. Energy shows a similar trend dropping by ~23% for the lower-quality images.

In this way, this example shows how audit tools can help in tailoring the captured data to the situation: in some cases, a lower image quality may be acceptable, but unacceptable in others; in some cases energy or storage might be scarce, in other cases resource availability may not be of concern.

7.3.3 Face Masking: We have discussed how capturing data in XR can have privacy and confidentiality considerations. Capturing data from the camera, for example, raises privacy and data protection concerns, given such data will relate to people, including users or bystanders, who may not even be aware their data is being captured.

It is therefore likely that privacy protection measures will form part of an auditability regime. To explore the potential and overheads of such, we created a data transformer that masks faces in images. Connected to the CamCap and ScrShot mechanisms, this works to detect and mask (block-out) a face before an image is persisted (see Fig. 7a). We ran a series of experiments, comparing the baseline performance (without face masking) with that of face masking while capturing every 10^{th}, 20^{th} and 30^{th} frame, again measuring CPU, RAM, FPS and current.

(a) Sample image from ScrShot with face masking.



(b) The impact of face detection and masking for the ScrShot and CamCap capture mechanisms.

Fig. 7. Face masking (left) and impacts (right).

Fig. 7b shows that performing face masking on images captured through ScrShot has notable effects on CPU, and on energy usage (~42% and ~12% on average across trials, respectively), while FPS and RAM is impacted slightly. CamCap generally has a marginal impact on performance (save some outliers) across all metrics, except for energy usage which increased by ~10% across all trials.

In performing this experiment, we aimed to demonstrate the potential for customised data perturbation to account for the audit concerns (here, privacy) of a particular scenario. And while we do observe some overheads, we found that even a fairly heavyweight transformation (involving real-time image processing, face detection and image manipulation) still enables reasonable performance against key XR performance measures such as FPS.

7.3.4 Offloading Data: We have shown that capturing audit data can significantly impact storage. Through a Drioditor logger, audit data can be offloaded, streamed to a remote server, rather than stored on device. This can be useful given the storage limitations of many XR devices, or indeed, to enable audit information to be made immediately and more widely available (e.g., via the cloud, rather than requiring access to the device).

To explore the potential and the implications of this, we again focus on the ScrShot and CamCap capture mechanisms due to their high storage requirements. Here, data is captured at maximum rate and quality, offloaded as it is captured, and streamed to a remote machine over a wireless LAN (802.11ac) connection. We measured a network load of ~70MB/s for ScrShot and ~15MB/s for CamCap, with a total data offload volume of ~4.2GB and ~0.9GB respectively over the one minute period. We compared the baseline (writing to local device storage) performance impacts with those measured while capturing and streaming audit data off-device.

Fig. 8 presents the results, where it can be seen that for both ScrShot and CamCap, offloading data appears to result in a reduction in energy usage, and a slight trend towards less RAM usage. For ScrShot, offloading slightly improves FPS, though does not affect CamCap which already operates at the maximum FPS even without offloading (again meaning performance improvements may not be observable through this measure). These results indicate the potential and propensity for offloading audit data at runtime, be it due to resource requirements or other aims, such as making audit data more accessible. Conversely, there may be situations where offloading audit data is impractical, e.g., due to a lack of network coverage in remote locations, or network usage fees. This again highlights the contextual nature of audit, and the need for tools to assist. Note that while Droiditor's current remote logger only supports direct streaming, other loggers could offload data in different ways (e.g., using batches, when the app is closed, etc). Here, the purpose was simply to indicate the potential for, and impacts of, offloading in general.

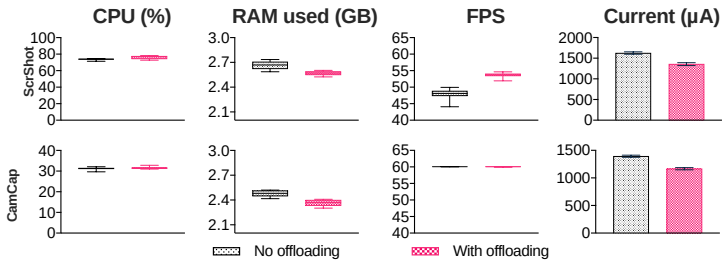


Fig. 8. The impact of offloading data for the ScrShot and CamCap capture mechanisms.

7.4 Summary

Our results indicate the viability of tools that enable the runtime capture of audit data, and importantly, how audit regimes can be tailored to accommodate different application and audit needs. Specifically, we showed how interventions at different stages of the audit data pipeline can be adapted and configured to align the capture of

audit data with the requirements of the particular scenario – here through mechanisms that give the flexibility to *(i)* define the relevant audit data sources; *(ii)* select and determine the nature of the data arising from those sources; and *(iii)* control the form and location of the records produced.

Naturally, our results are merely intended to be illustrative, as the precise overheads incurred will vary (as will the audit considerations) according to the specific circumstances, including, for example, the nature of the application, scenario, usage context, and audit configurations employed. And indeed, while our focus here was on AR, VR, with its more immersive nature, and MR, with its integration of various devices, may raise additional concerns and complexities – aspects that can be explored in future work. That said, presenting details of different capture functionality, configurations, and the resulting overheads, helps illustrate relevant concerns, and in enabling more informed approaches to audit. Moreover, our findings do demonstrate that there are a range of feasible approaches for managing and adapting data capture to a given scenario, as a means for balancing audit-related considerations. In this way, our results highlight in a general sense the need for auditability tooling to be sufficiently flexible and configurable, to enable such concerns to be appropriately managed and balanced.

8 DISCUSSION

Though audit is important, we have argued (§5) and shown (§7) that auditability itself will raise challenges. In identifying and quantifying some ways in which tensions between audit and application needs can be balanced, and by demonstrating the feasibility of methods for the runtime capture of audit data from a range of sources, we provide a step forward towards more practical audit. We now discuss a number of general areas concerning audit for further research and attention.

8.1 The Practicalities of Audit

Audit looks set to be of increasing importance (§2). However, auditability brings practical considerations. That is, while audit will have its own set of requirements that must be met, there will be also be a range of performance, resource, privacy, security and other related constraints that have direct technical implications for those building systems (§5).

Importantly, however, this should not be framed as zero-sum: audit *or* some other application aim. Rather, those involved must consider how capturing data can best support any audit obligations and other responsibilities in line with the application's operational needs and wider concerns. For instance, and as we have discussed, a reduction in frame rate (or another performance measure), greater power consumption, and/or large audit data management requirements may cause applications to become less responsive, exhaust resources, result in spurious or inappropriate behaviour, or fail entirely. This itself can increase the risk of loss, damage, or harm occurring. As such, the overheads of auditability are an important consideration; i.e. *it will be crucial to ensure that audit mechanisms do not further contribute to the very issues they are supposed to help manage and prevent.*

Ultimately, audit and application requirements, and what sort of balancing is 'acceptable', is highly contextual and will therefore require consideration on a case-by-case basis. Indeed, not all aspects will be important in all circumstances. For some applications, the user experience might be paramount; in others, it may be the environmental or system-related factors. In some scenarios, a screenshot every second or on particular events may suffice; for others, it may be of utmost importance that all frames are available for scrutiny. In some scenarios, more granular data may be required; in others, summarial or sanitised data may be appropriate. In some scenarios, the characteristics of the user (or intended user), including their age and abilities, may inform as to what information is relevant for audit, which might vary between users. There are therefore real opportunities for future work to explore what factors might be relevant in different contexts. We have, for instance, explored the use of taxonomies to describe the information relevant for auditing for the misuse of AI services [41], where similar approaches might be useful for other domains.

In short, audit data capture regimes will need to be tailored (and tailorable) to the situation. This paper brings attention to the need for such, illustrating how flexible means for customising, controlling and adjusting what is captured for audit (and how it is recorded) can assist in managing and balancing audit concerns.

8.2 Audit Tooling

To explore the considerations of capturing audit data, and how tooling can assist with balancing these, we introduced Droiditor. Droiditor was explicitly designed for such, by providing a highly configurable and adaptable data capture capability. It enables the capture of audit data from a range of types and sources (system and environmental), flexible audit configurations (adjustable both remotely and at runtime), and fully-customisable audit regimes. By focusing on Android as a platform, Droiditor is widely-applicable; able serve a broad range of apps; provides a common approach that reduces the need for (and thus inconsistencies associated with) custom tooling (§4); while showing the potential for audit tools that can be relevant at scale. And indeed, as it works on a mobile platform, *it is not only relevant for XR apps, but across mobile apps in general.*

Droiditor represents but one approach, and is a work in progress (our development roadmap includes additional capture mechanisms, data transformers and loggers, among others). Given the lack of available audit tooling (§4), we used Drioditor to enable an exploration of how audit tensions and trade-offs might be balanced, and highlight key types of functionality that audit tooling should generally support. Though our focus was on AR, many similar considerations will also be relevant for VR and MR, and so our work provides a starting point for research into the specific requirements for audit tools supporting those other forms of XR – which might, for example, account for the immersive nature of VR, and the complex device interactions as envisaged for MR.

In terms of facilitating auditability, there is an argument that the XR and OS platforms should themselves integrate audit functionality into their offerings. Indeed, though we sought to minimise the work required by developers for making their apps auditable, some workarounds were required. Platforms that directly integrate auditability mechanisms could better streamline developer interaction and perhaps enable more performant audit regimes. Given the increasing concerns regarding technology and the emerging regulatory environment, things may well go this way.

There is also potential for integrating generic audit functionality elsewhere in the 'stack', which is an area warranting community attention. For instance, various platforms and frameworks exist that support the development of XR, such as Unity [76], UnrealEngine [38], ARCore [29], Vuforia [80], and A-Frame [1], and there are also those for other domains (e.g., the IoT). The integration of audit tooling into such can allow for capturing more application- and domain-specific concerns. Note that the XR frameworks (mentioned above) can run on Android, meaning that Droiditor can operate across these, and thereby work to complement any audit-relevant information captured by such. There also appears scope for audit tools that focus on low-level and more general system behaviour (e.g., system-calls [59, 70]), as well as those at the opposite extreme, which involve guiding and helping developers integrate narrow and highly application-specific audit considerations into their code. In this way, there can be interplays between the data recording mechanisms operating at different levels of specificity, detail and abstraction.

As we have detailed, auditability is an area not yet sufficiently considered by the tech community, and there is a clear need for tooling that supports the audit process. However, by presenting Droiditor, we not only provide a tangible (and open-source) step forward, but also draw attention to this area of increasing importance. There are clear opportunities, and also a clear need for work on audit tooling for different forms of XR, frameworks, platforms, and at different levels of abstraction, and for considering how these might interoperate.

8.3 General Auditability Considerations

In terms of moving forward, there are a number of additional areas warranting attention by the research community, some of which we now discuss.

8.3.1 Data Integrity and Veracity: The veracity ('truthfulness') of audit data is important, particularly where audit relates to responsibilities and legal obligations. For instance, to serve as evidence over what occurred, or to inform an investigation, audit data must properly reflect what happens; i.e. records must be complete, representative, consistent and not tampered with. This requires means and processes to ensure the integrity of *(i)* the audit data recorded, as well as *(ii)* any processing of that data, and *(iii)* the audit data capture/tooling mechanisms themselves. Such considerations are crucial for audit data to enable oversight, transparency, and therefore accountability. Relevant in this context is that balancing the audit implications may involve some selection/transformation of the data, so an important consideration is that the auditability tools themselves do not undermine (or raise questions about) the audit data's validity, or the audit process in general.

8.3.2 Managing Access: Much audit data has the propensity to be sensitive, possibly including personal or confidential information regarding the user, bystanders, a physical location (factories, homes), equipment, business processes, and so on. A range of parties may be interested in audit data—legitimately or otherwise—and therefore there is a need to ensure that the access to and use of audit data is carefully managed, where only those who should have access to particular information can do so when and where necessary.

Managing access to audit data can be challenging. Clearly there is a role for well-established data management practices, including strong access controls and encryption. However, the broader questions over when, how, and by whom data can be accessed requires consideration. Indeed, certain data will be more sensitive than others [46]. And different stakeholders may have different and competing interests: an operator might not want certain data flowing to their employer, a developer might need access to raw data whereas a manager may (and should) not. The appropriateness of such access may be contextual, i.e. it may only be under certain, specific circumstances that access should be allowed. Further, if data is distributed across technical or organisational boundaries, maintaining a consistent management regime can easily become complex. An important area for future work concerns robust, yet flexible controls that can sufficiently account for the complexities surrounding real-world audit data requirements. Also important is the audit of when such records are accessed and how they are used; i.e. there is a need for appropriate mechanisms for 'watching the watcher'.

8.3.3 Usability of Audit Data: Ultimately, the purpose of audit data is to enable transparency, monitoring and review. Given a range of stakeholders can have an interest in audit data, there is scope for work that explores how best to cater for the needs of each. That is, audit data should support, for instance, developers to explore, debug and diagnose, regulators to interrogate, investigate and verify, and users to understand and scrutinise [55]. In this way, it is important that any audit regime facilitates *useful* transparency, by providing information that is *contextually appropriate:* relevant, accurate and complete, proportionate, and comprehensible [11].

Of course, how this information can be meaningfully conveyed to the relevant stakeholders will again depend on the specific circumstances. However, actively considering the dimensions to 'appropriateness' (e.g., sensitivity, whether appropriate for the target audience, etc.) when defining a data capture regime may provide a way for balancing some of these concerns. And note that regulatory requirements, guidance, as well as standards and best practices are emerging to shape directions in this space [23, 24, 39].

In all, there are clear opportunities for work on how best to support different stakeholders in understanding, interrogating, analysing, and visualising audit data, both generally and in specific contexts.

8.3.4 Legal Considerations: Also important is that audit data may have legal implications. As discussed, data protection law, which regulates the use of personal data (that relating to individuals), is clearly relevant, given that much XR data will relate to users and bystanders. Indeed, in many cases, audit infrastructures ultimately operate to surveil. This is important given the global relevance of data protection principles [24, 74], where such issues will need to be considered in the design of auditing regimes (e.g. 'data protection by design' [48]). This may include organisational measures such as impact assessments [24] or particular business processes to manage

the risks of audit data, or technical measures that attempt to mitigate particular concerns, with one such example being our face-masking mechanism (§7.3.3). Though again, note that any intervention, including those privacy related, can raise other concerns, e.g., where removing people from audit data can hinder bias and discrimination assessments [57].

Beyond data protection, other considerations are relevant as well. For instance, capturing detailed information about a system and how it works might raise commercial sensitivity issues around intellectual property and trade secrets [63]. There can also be complications should data move across geographic and jurisdictional boundaries, which might impact how data is controlled and accessed, and where certain aspects might require adaptation to local requirements. Audit data can serve as evidence, which might result in particular management requirements for particular sectors. And indeed, there may be positive obligations for organisations to keep particular records, which might stem from contracts, standards, or regulations, including those sector-specific (such as those in finance) or those more general (as we see in the EU's proposed AI Act [22]).

In this way, there are legal dimensions that will require consideration as part of an auditability regime, which in turn will influence the nature of the information captured.

8.4 Summary

In all, there is a clear need for auditability, and for tools that help facilitate the capture of relevant data for audit. Again, auditability itself will raise a range of concerns, both those specific to audit and those broader. This means that audit requirements will need to be balanced with the needs and constraints of applications, the platforms on which they run, as well as the contexts in which they are used and operate. Through Droiditor, we have explored some of the practicalities of audit, and have shown that flexible tooling represents a promising way forward.

9 CONCLUSION

As systems become more complex and context-dependent, and given public calls for greater scrutiny over technology, audit is an area of increasing importance. Crucial to this is the ability to audit systems. Auditability, however, is currently an area that is under-considered, yet it is particularly important for XR, and pervasive and mobile systems more generally.

We have indicated that audit poses challenges, where a range of considerations must be balanced. Tackling these will be crucial for bringing about greater transparency, oversight, and accountability regarding technology that is rapidly growing in prominence. Here we have demonstrated the need and potential for flexible tooling that enables XR systems to become more auditable, and introduce Droiditor as a generic, configurable, and open source auditability tool (available to the community to use and to extend) to show how *configurable and flexible* auditability mechanisms can assist in balancing the inherent tensions regarding the practicalities of audit. Our broader aim is to raise community awareness of audit-related concerns, and to encourage work on making systems more *auditable by default*, particularly as audit becomes increasingly on the agenda.

ACKNOWLEDGMENTS

We acknowledge the financial support of the UK Engineering & Physical Sciences Research Council (EP/P024394/1, EP/R033501/1) and Microsoft via the Microsoft Cloud Computing Research Centre.

REFERENCES

[1] A-Frame. 2021. A-Frame - Make WebVR. https://aframe.io/

[2] Hasaneen Fathy Al Janabi, Abdullatif Aydin, Sharanya Palaneer, Nicola Macchione, Ahmed Al-Jabir, Muhammad Shamim Khan, Prokar Dasgupta, and Kamran Ahmed. 2019. Effectiveness of the HoloLens mixed-reality headset in minimally invasive surgery: a simulation-based feasibility study. Surgical Endoscopy 34, 3 (2019), 1143–1149.

[3] Android Developers. 2020. Perfetto: Android Developers. https://developer.android.com/studio/command-line/perfetto. Accessed: 2020-09-15.

[4] Ronald Azuma and Chris Furmanski. 2003. Evaluating Label Placement for Augmented Reality View Management. In Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR '03). IEEE Computer Society, USA, 66.

[5] Claudio Bettini and Daniele Riboni. 2015. Privacy protection in pervasive systems: State of the art and technical challenges. Pervasive and Mobile Computing 17 (2015), 159–174.

[6] Lindsay Blackwell, Nicole Ellison, Natasha Elliott-Deflo, and Raz Schwartz. 2019. Harassment in social virtual reality: Challenges for platform governance. Proceedings of the ACM on Human-Computer Interaction 3, CSCW (2019), 1–25.

[7] Russell Brandom. 2019. There are now 2.5 billion active Android devices. https://www.theverge.com/2019/5/7/18528297/google-io-2019-android-devices-play-store-total-number-statistic-keynote.

[8] Julie Carmigniani and Borko Furht. 2011. Augmented reality: an overview. (2011), 3–46.

[9] Woohyeok Choi, Sangkeun Park, Duyeon Kim, Youn-kyung Lim, and Uichin Lee. 2019. Multi-Stage Receptivity Model for Mobile Just-In-Time Health Intervention. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 3, 2, Article 39 (June 2019), 26 pages. https://doi.org/10.1145/3328910

[10] Richard Cloete, Chris Norval, and Jatinder Singh. 2020. A Call for Auditable Virtual, Augmented and Mixed Reality. In 26th ACM Symposium on Virtual Reality Software and Technology (Virtual Event, Canada) (VRST '20). Association for Computing Machinery, New York, NY, USA, Article 16, 6 pages. https://doi.org/10.1145/3385956.3418960

[11] Jennifer Cobbe, Michelle Seng Ah Lee, and Jatinder Singh. 2021. Reviewable Automated Decision-Making: A Framework for Accountable Algorithmic Systems. In Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (Virtual Event, Canada) (FAccT '21). Association for Computing Machinery, New York, NY, USA, 598–609. https://doi.org/10.1145/3442188.3445921

[12] Corentin Coupry, Sylvain Noblecourt, Paul Richard, David Baudry, and David Bigaud. 2021. BIM-Based Digital Twin and XR Devices to Improve Maintenance Procedures in Smart Buildings: A Literature Review. Applied Sciences 11, 15 (2021), 6810.

[13] Eduardo Cuervo, Krishna Chintalapudi, and Manikanta Kotaru. 2018. Creating the perfect illusion: What will it take to create life-like virtual reality headsets?. In Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications. 7–12.

[14] Jaybie A De Guzman, Kanchana Thilakarathna, and Aruna Seneviratne. 2019. Security and privacy approaches in mixed reality: A literature survey. ACM Computing Surveys (CSUR) 52, 6 (2019), 1–37.

[15] K. Debattista, K. Bugeja, S. Spina, T. Bashford-Rogers, and V. Hulusic. 2018. Frame Rate vs Resolution: A Subjective Evaluation of Spatiotemporal Perceived Quality Under Varying Computational Budgets. Computer Graphics Forum 37, 1 (2018), 363–374. https://doi.org/10.1111/cgf.13302 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13302

[16] Android Developers. 2020. Permissions overview: Android Developers. https://developer.android.com/guide/topics/permissions/overview.

[17] Android Developers. 2021. PixelCopy : Android developers. https://developer.android.com/reference/android/view/PixelCopy

[18] Android Developers. 2021. Playback Capture | Android Developers. https://developer.android.com/guide/topics/media/playback-capture.

[19] Android Developers. 2021. Quickstart for Android - ARCore - Google Developers. https://developers.google.com/ar/develop/java/quickstart.

[20] Android Developers. 2021. View : Android developers. https://developer.android.com/reference/android/view/View#getDrawingCache()

[21] Nicholas Diakopoulos. 2016. Accountability in algorithmic decision making. Commun. ACM 59, 2 (2016), 56–62.

[22] European Commision. 2021. Proposal for a regulation of the European Parliament and of the Council laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain Union legislative acts. 2021/0106 (COD) (21 April 2021), 1–108.

[23] European Commission. 2020. White Paper On Artificial Intelligence - A European Approach to Excellence and Trust. COM(2020) 65 (2020).

[24] European Union. 2016. Regulation (EU) 2016/679 General Data Protection Regulation. Official Journal of the European Union L119 (4 May 2016), 1–88.

[25] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. 2018. Datasheets for datasets. arXiv preprint arXiv:1803.09010 (2018).

[26] Vesna Geršak, Helena Smrtnik Vitulić, Simona Prosen, Gregor Starc, Iztok Humar, and Gregor Geršak. 2020. Use of wearable devices to study activity of children in classroom; Case study—Learning geometry using movement. Computer communications 150 (2020), 581–588.

[27] GlobalStats. 2021. Mobile operating system market share worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide

[28] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim, and Todd Millstein. 2013. Reran: Timing-and touch-sensitive record and replay for android. In 2013 35th International Conference on Software Engineering (ICSE). IEEE, 72–81.

[29] Developers Google. 2020. ARCore overview | Google Developers. https://developers.google.com/ar/discover

[30] Ankur Handa, Richard A Newcombe, Adrien Angeli, and Andrew J Davison. 2012. Real-time camera tracking: When is high frame-rate best?. In European Conference on Computer Vision. Springer, 222–235.

[31] Jennifer Herron. 2016. Augmented reality in medical education and training. Journal of Electronic Resources in Medical Libraries 13, 2 (2016), 51–55.

[32] Wolfgang Hoenig, Christina Milanes, Lisa Scaria, Thai Phan, Mark Bolas, and Nora Ayanian. 2015. Mixed reality for robotics. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 5382–5387.
[33] Fred Hohman, Kanit Wongsuphasawat, Mary Beth Kery, and Kayur Patel. 2020. Understanding and visualizing data iteration in machine learning. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. 1–13.
[34] Sean Hollister. 2019. Here's the US Army version of HoloLens that Microsoft employees were protesting. https://www.theverge.com/2019/4/6/18298335/microsoft-hololens-us-military-version. Accessed: 2020-01-31.
[35] HTC. 2020. HTC Vive Health and Safety. https://dl4.htc.com/vive/safty_guide/91H02887-08MRev.A.PDF. Accessed: 2020-03-07.
[36] Gang Hu, Xinhao Yuan, Yang Tang, and Junfeng Yang. 2014. Efficiently, effectively detecting mobile app bugs with appdoctor. In Proceedings of the Ninth European Conference on Computer Systems. 1–15.
[37] C. E. Hughes, C. B. Stapleton, D. E. Hughes, and E. M. Smith. 2005. Mixed reality in education, entertainment, and training. IEEE Computer Graphics and Applications 25, 6 (2005), 24–30.
[38] Epic Games Inc. 2021. The most powerful real-time 3D creation tool - Unreal Engine. https://www.unrealengine.com/en-US/
[39] Information Commissioner's Office. [n. d.]. https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/documentation/. Accessed: 2020-10-01.
[40] Seyyed Ahmad Javadi, Richard Cloete, Jennifer Cobbe, Michelle Seng Ah Lee, and Jatinder Singh. 2020. Monitoring Misuse for Accountable'Artificial Intelligence as a Service'. In Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society. 300–306.
[41] Seyyed Ahmad Javadi, Chris Norval, Richard Cloete, and Jatinder Singh. 2021. Monitoring AI Services for Misuse. In Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society. 597–607.
[42] Y. Jin and Y. Ishino. 2006. DAKA: Design Activity Knowledge Acquisition through Data-Mining. International Journal of Production Research 44, 14 (July 2006), 2813–2837. https://doi.org/10.1080/00207540600654533
[43] Bellal Joseph and David G Armstrong. 2016. Potential perils of peri-Pokémon perambulation: the dark reality of augmented reality? Oxford medical case reports 2016, 10 (2016), omw080.
[44] Andrew King, Faisal Kaleem, and Khaled Rabieh. 2020. A Survey on Privacy Issues of Augmented Reality Applications. In 2020 IEEE Conference on Application, Information and Network Security (AINS). IEEE, 32–40.
[45] Joshua A. Kroll. 2021. Outlining Traceability: A Principle for Operationalizing Accountability in Computing Systems. In Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (Virtual Event, Canada) (FAccT '21). Association for Computing Machinery, New York, NY, USA, 758?771. https://doi.org/10.1145/3442188.3445937
[46] Kiron Lebeck, Kimberly Ruth, Tadayoshi Kohno, and Franziska Roesner. 2017. Securing Augmented Reality Output. In 2017 IEEE Symposium on Security and Privacy (SP). IEEE, San Jose, CA, USA, 320–337. https://doi.org/10.1109/SP.2017.13
[47] Zongyu Lin, Shiqing Lyu, Hancheng Cao, Fengli Xu, Yuqiong Wei, Hanan Samet, and Yong Li. 2020. HealthWalks: Sensing Fine-Grained Individual Health Condition via Mobility Data. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 4, 4, Article 138 (Dec. 2020), 26 pages. https://doi.org/10.1145/3432229
[48] Tom Lodge, Andy Crabtree, and Anthony Brown. 2018. IoT App Development: Supporting Data Protection by Design and Default. In Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers (Singapore, Singapore) (UbiComp '18). Association for Computing Machinery, New York, NY, USA, 901–910. https://doi.org/10.1145/3267305.3274151
[49] Microsoft. 2021. Safety Information. https://support.microsoft.com/en-gb/help/4023454/safety-information.
[50] Microsoft HoloLens. 2021. HoloLens 2-overview, features, and specs. https://www.microsoft.com/en-us/hololens/hardware
[51] Paul Milgram and Fumio Kishino. 1994. A taxonomy of mixed reality visual displays. IEICE TRANSACTIONS on Information and Systems 77, 12 (1994), 1321–1329.
[52] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In Proceedings of the conference on fairness, accountability, and transparency. 220–229.
[53] Ibrahim Nadir, Zafeer Ahmad, Haroon Mahmood, Ghalib Asadullah Shah, Farrukh Shahzad, Muhammad Umair, Hassam Khan, and Usman Gulzar. 2019. An Auditing Framework for Vulnerability Analysis of IoT System. In 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). IEEE, 39–47.
[54] A.Y.C. Nee and S.K. Ong. 2013. Virtual and Augmented Reality Applications in Manufacturing. IFAC Proceedings Volumes 46, 9 (2013), 15–26. https://doi.org/10.3182/20130619-3-RU-3018.00637 7th IFAC Conference on Manufacturing Modelling, Management, and Control.
[55] Chris Norval, Jennifer Cobbe, and Jatinder Singh. 2021. Towards an accountable Internet of Things: A call for 'reviewability' . In Privacy by Design for the Internet of Things: Building Accountability and Security. IET.
[56] Oculus. 2020. Oculus Health and Safety Warnings. https://securecdn.oculus.com/sr/oculusquest-warning-english. Accessed: 2020-03-07.
[57] Royal Courts of Justice. 2020. R (Bridges) v Chief Constable of South Wales Police & Information Commissioner [2020] EWCA Civ 1058.
[58] Frank Pasquale. 2015. The Black Box Society: The Secret Algorithms That Control Money and Information. Harvard University Press.

[59] Thomas FJ-M Pasquier, Jatinder Singh, David Eyers, and Jean Bacon. 2015. CamFlow: Managed data-sharing for cloud services. IEEE Transactions on Cloud Computing 5, 3 (2015), 472–484.
[60] Gabriele Pizzi, Daniele Scarpi, Marco Pichierri, and Virginia Vannucci. 2019. Virtual Reality, Real Reactions?: Comparing Consumers' Perceptions and Shopping Orientation Across Physical and Virtual-Reality Retail Stores. Computers in Human Behavior 96 (2019), 1–12. https://doi.org/10.1016/j.chb.2019.02.008
[61] Pokémon GO Death Tracker. 2020. Pokémon GO Death Tracker. http://pokemongodeathtracker.com/. Accessed: 2020-06-02.
[62] Zhengrui Qin, Yutao Tang, Ed Novak, and Qun Li. 2016. Mobiplay: A remote execution based record-and-replay tool for mobile applications. In Proceedings of the 38th International Conference on Software Engineering. 571–582.
[63] Franziska Roesner, Tamara Denning, Bryce Clayton Newell, Tadayoshi Kohno, and Ryan Calo. 2014. Augmented reality: Hard problems of law and policy. In Proceedings of the 2014 ACM international joint conference on pervasive and ubiquitous computing: adjunct publication. 1283–1288. https://doi.org/10.1145/2638728.2641709
[64] E. E. Sabelman and R. Lam. 2015. The real-life dangers of augmented reality. IEEE Spectrum 52, 7 (2015), 48–53.
[65] Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Müller. 2019. Explainable AI: interpreting, explaining and visualizing deep learning. Vol. 11700. Springer Nature.
[66] Maxim Schwartz, SK Gupta, DK Anand, and Robert Kavetsky. 2007. Virtual mentor: A step towards proactive user monitoring and assistance during virtual environment-based training. In Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems. 280–287.
[67] Qijia Shao, Amy Sniffen, Julien Blanchet, Megan E. Hillis, Xinyu Shi, Themistoklis K. Haris, Jason Liu, Jason Lamberton, Melissa Malzkuhn, Lorna C. Quandt, James Mahoney, David J. M. Kraemer, Xia Zhou, and Devin Balkcom. 2020. Teaching American Sign Language in Mixed Reality. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 4, 4, Article 152 (Dec. 2020), 27 pages. https://doi.org/10.1145/3432211
[68] William R Sherman and Alan B Craig. 2018. Understanding virtual reality: Interface, application, and design. Morgan Kaufmann.
[69] Sanni Siltanen and Hanna Heinonen. 2020. Scalable and responsive information for industrial maintenance work: Developing XR support on smart glasses for maintenance technicians. In Proceedings of the 23rd International Conference on Academic Mindtrek. 100–109.
[70] J. Singh, J. Cobbe, and C. Norval. 2019. Decision Provenance: Harnessing Data Flow for Accountable Systems. IEEE Access 7 (2019), 6562–6574.
[71] J. Singh, C. Millard, J. Reed, J. Cobbe, and J. Crowcroft. 2018. Accountability in the IoT: Systems, Law, and Ways Forward. IEEE Computer 51, 7 (2018), 54–65.
[72] Jatinder Singh, Julia Powles, Thomas Pasquier, and Jean Bacon. 2015. Data Flow Management and Compliance in Cloud Computing. IEEE Cloud Computing 2, 4 (July 2015), 24–32. https://doi.org/10.1109/MCC.2015.69
[73] Marco Speicher, Sebastian Cucerca, and Antonio Krüger. 2017. VRShop: A Mobile Interactive Virtual Reality Shopping Environment Combining the Benefits of On- and Offline Shopping. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 1, 3, Article 102 (Sept. 2017), 31 pages. https://doi.org/10.1145/3130967
[74] State of Califorina. 2018. The California Consumer Privacy Act (CCPA).
[75] Cynthia Stohl, Michael Stohl, and Paul M Leonardi. 2016. Digital age| managing opacity: Information visibility and the paradox of transparency in the digital age. International Journal of Communication 10 (2016), 15.
[76] Unity Techologies. 2021. Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine. https://unity.com/
[77] Oren M Tepper, Hayeem L Rudy, Aaron Lefkowitz, Katie A Weimer, Shelby M Marks, Carrie S Stern, and Evan S Garfein. 2017. Mixed reality with HoloLens: where virtual reality meets augmented reality in the operating room. Plastic and reconstructive surgery 140, 5 (2017), 1066–1070.
[78] Helma Torkamaan and Jürgen Ziegler. 2020. Mobile Mood Tracking: An Investigation of Concise and Adaptive Measurement Instruments. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 4, 4, Article 155 (Dec. 2020), 30 pages. https://doi.org/10.1145/3432207
[79] UK Government. 2020. Online Harms White Paper. https://www.gov.uk/government/consultations/online-harms-white-paper. Accessed: 2020-03-23.
[80] Vuforia. 2021. Vuforia engine developer portal. https://developer.vuforia.com/
[81] Qi Wang, Wajih Ul Hassan, Adam Bates, and Carl Gunter. 2018. Fear and logging in the internet of things. In Network and Distributed Systems Symposium.
[82] Yukang Yan, Chun Yu, Xin Yi, and Yuanchun Shi. 2018. HeadGesture: Hands-Free Input Approach Leveraging Head Movements for HMD Devices. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 2, 4, Article 198 (Dec. 2018), 23 pages. https://doi.org/10.1145/3287076
[83] Kanghyeok Yang, Sepi Aria, Changbum R. Ahn, and Terry L. Stentz. 2014. Automated Detection of Near-Miss Fall Incidents in Iron Workers Using Inertial Measurement Units. In Construction Research Congress 2014. American Society of Civil Engineers, Atlanta, Georgia, 935–944. https://doi.org/10.1061/9780784413517.096
[84] Z. Zhu, S. German, and I. Brilakis. 2010. Detection of Large-scale Concrete Columns for Automated Bridge Inspection. Automation in Construction 19, 8 (2010), 1047–1055. The role of VR and BIM to manage the construction and design processes.