It’s all in your head(set): Side-channel attacks on AR/VR systems

Yicheng Zhang, Carter Slocum, Jiasi Chen and Nael Abu-Ghazaleh
University of California, Riverside

Abstract

With the increasing adoption of Augmented Reality/Virtual Reality (AR/VR) systems, security and privacy concerns attract attention from both academia and industry. This paper demonstrates that AR/VR systems are vulnerable to side-channel attacks launched from software; a malicious application without any special permissions can infer private information about user interactions, other concurrent applications, or even the surrounding world. We develop a number of side-channel attacks targeting different types of private information. Specifically, we demonstrate three attacks on the victim’s interactions, successfully recovering hand gestures, voice commands made by victims, and keystrokes on a virtual keyboard, with accuracy exceeding 90%. We also demonstrate an application fingerprinting attack where the spy is able to identify an application being launched by the victim. The final attack demonstrates that the adversary can perceive a bystander in the real-world environment and estimate the bystander’s distance with Mean Absolute Error (MAE) of 10.3 cm. We believe the threats presented by our attacks are pressing; they expand our understanding of the threat model faced by these emerging systems and inform the development of new AR/VR systems that are resistant to these threats.

1 Introduction

AR/VR enables a range of new applications where users view virtual objects and scenes through a wearable headset. This information is perceived directly through the user’s senses, enabling the devices to enrich the user’s perception of the real world with overlaid virtual objects (Augmented Reality) or even completely replace the real world with the virtual (Virtual Reality). AR and VR support a range of new and exciting applications, from gaming to virtual exploration, training, and many others. Several commercial headsets are available on the market (*e.g.*, Microsoft Hololens 2 [41], Meta Quest 2 [38]), and an increasing number of applications and content are becoming available.

Security and privacy issues in the context of AR/VR systems have been receiving increasing interest. This application space presents a number of unique challenges because it lies at the intersection of the physical and digital worlds. AR/VR applications augment the physical world and can expose sensitive information about the activities of users [25, 58, 60], their interactions with the system [16], or the environment itself [19, 22, 57]. These applications can be downloaded from an App Store and executed on the headset. As AR/VR applications proliferate, AR/VR platforms are evolving to support the operation of multiple concurrent apps, each providing a different service visible to users [56]. For example, a user could meet with a remote 3D user [51] while simultaneously taking notes on a virtual whiteboard. Thus, it is critical to understand the threat models faced in emerging multi-app scenarios, to guide the development of mitigations and best practices in the supporting systems and software.

In this paper, we explore whether side-channel attacks are feasible threat vectors affecting AR/VR headsets running multiple concurrent apps. Our threat model considers a malicious application without special privileges that seeks to infer sensitive information about other concurrently running applications or user interactions with the device. We show that AR/VR engines expose performance counters and additional information that is accessible to user-level applications without special privileges. Moreover, these performance counters change in a way that correlates with sensitive information, enabling an attacker to infer this sensitive information from the counter values. For example, the frame rate sustained by the device goes down when other applications execute computationally demanding operations, such as rendering a virtual hand or parsing new information about the environment, providing recognizable signatures allowing a spy to track the behavior of other applications.

Although similar side-channel attacks have been demonstrated in the context of Central Processing Unit (CPU) (e.g., [2, 53]) and Graphics Processing Unit (GPU) (e.g., [50, 71]), AR/VR applications and systems are significantly different in terms of the nature of the workloads as well as the characteristics of both the hardware and the run-time on the headsets.

We present a taxonomy of the possible targets of side-channel attacks, showing that the targets can include: (1) inferring user interactions such as hand gestures or voice commands that are used as input to the device; (2) inferring information about other applications: different applications have different signatures based on their baseline behavior, as well as a function of sensitive data they may process; and (3) inferring information about the environment or bystander behavior: the nature of the environment or the user behavior can cause different identifiable signatures in the performance counters we track.

To illustrate these threats, we develop a number of end-to-end attacks selected to represent the attack classes we identify above. Specifically, we construct a side-channel attack that infers the victims' hand gestures and voice commands by tracking the rendering performance counters provided by the shared rendering engine (discussed in § 4.1 and § 4.2). We demonstrate another attack that monitors the keystrokes entered by the victim (discussed in § 4.3). We construct another attack that infers the launch of different applications, identifying the launched application (discussed in § 5). Finally, we demonstrate an attack that infers information about the environment – the attack is able to perceive the presence of an approaching person and estimate the distance (discussed in § 6).

As a user interacts with a Mixed Reality (MR) device, or as the environment changes, these interactions trigger processing tasks that leave a footprint on resource and rendering-related performance counters. Our side-channel attacks look for these signatures to infer sensitive information about other applications, user interactions, or changes in the environment. The attacks illustrate a number of potential leakage sources on AR/VR devices. Although known patterns of defenses such as limiting access to performance counters could be leveraged, we believe that these defenses should be thoughtfully considered due to the fundamentally different nature of AR/VR applications. In particular, while conventional computing applications share only the resources, AR/VR applications share the environment and the user, inviting new solutions that are able to provide access without compromising security and privacy. We discuss the challenges in mitigating these attacks and outline potential solutions in § 7.

In summary, the contributions of this paper are:

- We present a taxonomy of the potential targets and leakage sources of software-accessible side-channel attacks on AR/VR devices and applications.
- We demonstrate five end-to-end side-channel attacks illustrating three types of targets: Inferring (1) user interactions (hand gesture or voice command as inputs, and virtual keyboard inputs); (2) information about concurrent applications (fingerprinting newly launched applications); and (3) distance estimation of a bystander. The proof-of-concept attack can infer the walking bystander distance from the headset, providing evidence that there are side-channel signatures from the surrounding environment.
- We discuss the challenges in mitigating these classes of attacks in the context of AR/VR applications and headsets.

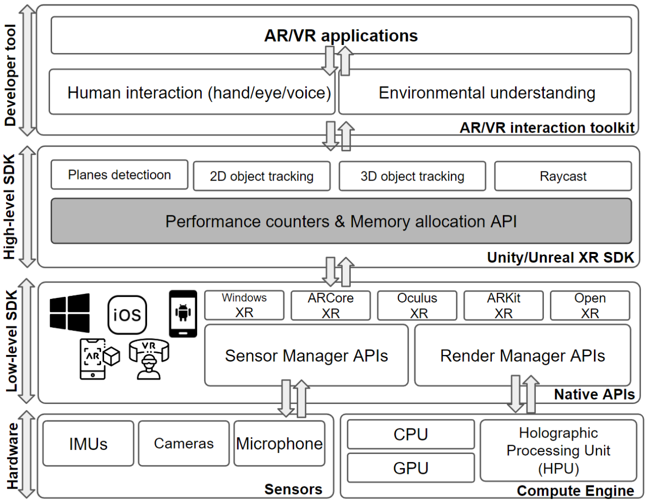


Figure 1: General software and hardware architecture for AR/VR device.

Disclosure: We have reported all of our findings to the headset manufacturers, Microsoft and Meta, as well as the Unity and Unreal Engine security teams.

2 Background

In this section, we first present the general architecture of an AR/VR system and explain the link between development tools, devices, and performance counters. We then discuss how the run time works and how it supports concurrent applications.

Devices and developer tools. We analyzed the Hololens 2 and Meta Quest 2 and investigated their common aspects in terms of device hardware and software developer tools. A similar architecture is also shared with other AR/VR devices, like Magic Leap [27] and HTC Vive [12]. Fig. 1 shows a general software and hardware architecture of an AR/VR headset. The architecture consists of four main subsystems, starting from the bottom: device hardware, low-level Software Development Kit (SDK), high-level SDK, and developer tools.

The hardware components of AR/VR devices include computing engines (*e.g.*, CPU and GPU) and sensing units (*e.g.*, inertial measurement unit (IMU), cameras, and microphones). Each AR/VR device is supported by a manufacturer-specific low-level SDK that reads and processes device-specific sensor data, providing the results to the high-level SDK. For instance, Hololens is supported by Windows XR SDK, while Meta Quest uses the Oculus XR SDK. The low-level SDK also leverages compute engines to accelerate some sensor processing tasks. For example, the Hololens 2 has a holographic processing unit (HPU) 2.0 [64], which provides hardware-accelerated support for computer vision tasks such as hand tracking and eye tracking.

Above the low-level SDKs, high-level SDKs define standard APIs that provide common AR/VR-related functionalities across the supported devices (*e.g.*, plane detection, ray-casting, 2D and 3D object tracking). Two primary high-level SDK options are Unity XR SDK [68] or Unreal XR SDK [15]. Finally, at the top of the architecture, developers writing AR/VR applications can access two main groups of functionality: human interaction and environmental understanding. For human interaction, AR/VR applications can understand user commands through hand tracking, eye tracking, and voice commands. In terms of environmental understanding, it supports spatial mapping and scene understanding.

In summary, AR/VR devices have specific low-level SDKs tailored to their hardware, which provide a common set of AR/VR functionality to the high-level SDKs, which developers can then utilize in their applications.

Performance counters. Specific functionality exposed by the high-level SDK includes performance counters and memory allocation APIs. These help developers track and optimize the performance of their running applications. These counters or APIs rather track application performance at the software level, abstracted away from specific hardware, as shown in Fig. 1. In this work, we focus on performance counters related to rendering and thread time, as they are common across AR/VR devices, no matter their hardware capabilities. Examples include the *Number of draw calls* and *Vertex count*, which reflect the graphical complexity of 2D/3D objects rendered on the display (further details on the performance counters are provided in Section 3.2). As a victim interacts with the device or the environment changes, the low-level SDKs perform processing whose results propagate to the high-level SDK and impact the performance counters there, which we leverage to launch our attacks.

Permissions in the multi-app AR/VR platform. Many popular augmented reality headsets, like the Microsoft Hololens 2, Meta Quest 2, and Magic Leap, support multiple applications simultaneously, enabling these applications to enrich the user's mixed reality experience collectively. However, the isolation of these multiple applications in AR/VR platforms face presents a number of challenges: each application likely needs information about the environment for rendering and potentially also to provide the opportunity for the user to interact with the environment. Thus, even though the applications may be isolated in the user's field of view/display, it is unclear how to set up permissions without interfering with usability. As a result, we observed on the headsets we tested that applications running in the background have access to various APIs in the high-level SDK, such as plane detection, which is shared with other applications. This access is necessary since the different applications may need to update what they display according to changes in where the user is looking or in response to user interactions. While there is some coarse-grained permission control (for example, only the foreground application has access to the hand tracking API), most other sensor data and performance counters were not subject to access control on the headsets we tested.

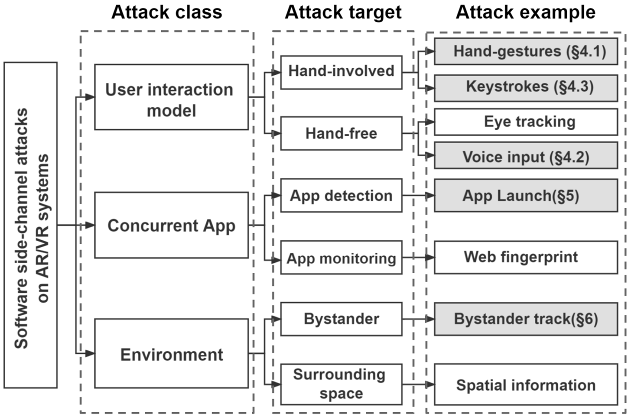


Figure 2: Taxonomy of software side-channel attacks on AR devices. The shaded attack examples specifically are demonstrated in the paper.

3 Attack Model and Experiment Setup

In this section, we discuss the threat models considered by our attacks. Specifically, we identify potential attack goals and leakage sources exploitable by attackers, and also describe our experimental setup and workflow.

3.1 Threat model and attack taxonomy

We assume a malicious program is running in the background with standard application-level permissions. Such threat models have been widely deployed in prior works (*e.g.*, [3,30,48,50,55,72,76,77]). A recent paper [35] presenting a side-channel attack on an AR device shares the same threat model, although without considering performance counters. The malicious program periodically probes certain performance counters available either from the hardware or the run-time system and analyzes the results to infer the activities of other concurrent apps, the user, or the environment.

AR/VR headsets and applications are unique in that they bridge the physical world, perceiving it through sensors, augmenting it, or potentially completely replacing it (in VR) with digitally projected models. As a result, these applications provide a target-rich environment for attackers. We first present a

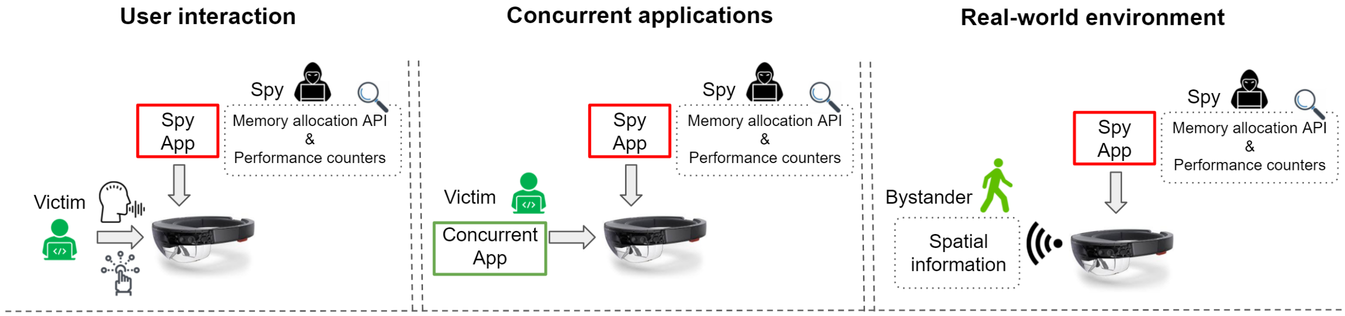


Figure 3: Three attack scenarios are considered in this work. For user interaction, we assume a victim interacts with the device through hand gestures or voice commands. For concurrent applications, the victim launches a new application. In the real-world environment, a bystander passes through the field of view of the AR/VR device. In all scenarios, a spy program runs concurrently and collects memory allocation API and performance counters.

taxonomy of the types of attack goals we envision our attack targeting.

Fig. 2 presents a taxonomy of the attack space we expect to be of interest to attackers in our threat model. Broadly, the attacks can be placed in three categories:

- **Spying on user interactions**: In AR/VR applications, the user is at the confluence of the physical and digital worlds. User interactions can provide valuable information to a spy, to enable them to track the user's activities and even infer passwords or credentials. Common user interaction interfaces with the device include hand gestures (*e.g.*, air tap to select a UI element, pinch and grab to move a virtual object), as well as hands-free user interactions (*e.g.*, voice commands, eye gaze). User interactions may cause changes in the performance counters due to changes in computations to track hands/head/eyes and recognize pre-defined hand/voice patterns, as well as from rendering virtual hands or gaze pointers.

- **Spying on concurrent applications**: The user may not wish to reveal information about other concurrently running apps, such as their launch, or detailed information about their operation such as the currently open website in a virtual browser. These concurrent apps may cause changes in the performance counters due to their typical rendering workload during launch, or based on their appearance as well as memory allocations.

- **Spy on the real-world (or virtual in the case of VR) environment**: The environment in which the user is operating the AR/VR device may contain private information, such as how many bystanders are passing by or the amount of clutter in the scene (*e.g.*, a blank hallway vs. a messy lab), which enables deriving the location of the user. The real-world environment may cause changes in the performance counters due to AR devices' continuous spatial mapping background process [46], which helps blend virtual objects with the real world in a realistic way.

In this work, we demonstrate five specific attacks from Fig. 2 (shaded boxes): hand gestures, voice input, keystrokes, app launch, and bystander ranging. These five attacks occur under three scenarios, shown in Fig. 3.

3.2 Available leakage vectors

Unity [18] and Unreal Engine [59] are two leading platforms for developing AR/VR applications, and they are supported on both the Hololens and Meta Quest headsets. They both provide resource monitoring APIs which provide access to various performance statistics (performance counters) to assist developers in improving the performance of their applications. These values are updated with every frame. The APIs are accessible to all applications, including the spy application, requiring only user-level permissions even when they are in the background. We exploit these APIs to obtain the side-channel information that enables us to carry out the attacks and recover sensitive information. We describe these APIs next.

Memory allocation API. After an application is launched on an AR/VR device, the system will allocate memory for it, based on the size and content of its visible objects. On Unity, we can use *AppMemoryUsage* [44] to track an app's memory usage, including current memory usage, memory usage peak, and memory usage limit. While on Unreal Engine, Rendering Hardware Interface (RHI) [69] provides *CurrentRendertargetMemorySize* and *CurrentTextureMemorySize*, two APIs to help developers understand the memory allocation for render targets and textures.

Although each application can only access the memory allocation information corresponding to its own usage, we notice that these numbers are affected by contention from user inputs or other applications. For example, when the built-in voice commands are triggered, the observed memory usage of

Categories	Counters
Frame rate	CPU frame rate, GPU frame rate, Frame time, Refresh rate, GPU input latency time
Thread	Game thread time, Render thread time, Working RHI thread time, Render thread idle count, RHI thread time, Swap buffer time
Render task	Number of draw calls, Number of primitives, Draw up call index counts, Vertex count, Texture pool size, GPU max texture mip count, Graph pool size percentage

Table 1: Performance counters used by our spy program, available from the Unity and/or Unreal game engines.

the spy program increases (discussed in Section 4.2); we conjecture that this is because the voice commands used shared buffers that are also treated as part of the spy program, resulting in memory usage increasing. Conversely, when concurrent applications are launched, the resident memory usage of the spy program decreases because of pressure on the physical memory arising from resource contention. By observing these contention-related patterns, the spy can infer the behavior of other concurrent applications (discussed in Section 5).

Performance counters. Performance counters are provided by both the Unity and Unreal XR high-level SDKs. Table 1 summarizes the available performance counters categorized by three groups:

- **Frame rate:** Since a sudden decrease in frame rate has been shown as a major factor underlying AR/VR sickness [78], it is important for developers to maintain a high frame rate. Towards this, Unity provides a *FrameTimingManager* API [67] to track the *CPU frame rate* and *GPU frame rate*. *CPU frame rate* is calculated as the time between the start of the frame and the next frame on the main thread. *GPU frame rate* reflects the GPU's work time between the work submitted to the GPU and finished. These two counters help developers determine where are the performance bottlenecks in AR/VR applications. If developers find it is CPU-bound, they can control workloads on the CPU by reducing how often some game objects' logic is updated [63]. If the frame rate is GPU-bound, developers can reduce the rendering resolution to limit the amount of work done by the GPU.

- **Thread:** Multi-threading is widely used in AR/VR applications, which support processing multiple rendering threads simultaneously. However, poor synchronization of multiple threads would bring in resource waste, which harms the immersive experience on AR/VR devices. Thus, thread-related counters assist developers by providing timing information for parallel threads. For example, Unreal XR SDK offers *Render thread time* and *Game thread time*. These two threads typically run in parallel. By tracking the timing information of these two threads, developers can synchronize the rendering work between the game thread and the render thread to make the application faster.

- **Render task:** The rendering-related counters represent the complexity of 2D/3D objects shown on the AR/VR display. For instance, *Vertex count* indicates the number of vertices in existing 2D/3D scenes. Primitives are predefined 3D objects, like cubes, spheres, etc. Each type of primitive corresponds to a different *Number of primitives*. With the help of render-related counters, developers can avoid involving too many or too complex objects, which may slow down the application's frame rate.

Although each application can only access its own performance counters, we notice that these numbers are affected by other applications or user interactions. This allows us to bridge a series of side-channel attacks. In general, those counters relating to graphical rendering tend to show increased load when user behaviors generate new display elements (*e.g.*, *Vertex count* increases when users make gestures that are rendered on the screen), while non-graphics-related counters tend to show increased load during non-rendering tasks (*e.g.*, *AppMemoryUsage* increases during parsing of voice commands, *CPU frame rate* decreases while processing changes in the real-world environment). More details are provided in the following sections.

3.3 Experimental setup

We demonstrate our attacks on two devices: a Microsoft Hololens 2 [41] (a Windows-based AR headset) and a Meta Quest 2 [38] (an Android-based VR headset). We chose these devices for the following reasons: Firstly, they serve as two of the most representative headsets for the AR and VR domains, respectively. Secondly, the low-level SDK of the two devices is distinct: Hololens is built upon the Windows XR SDK while the Quest is based on the Oculus XR. We seek to show that our side-channel attacks generally work on different AR/VR systems. The malicious spy applications were designed using both Unity version 2020.3.16f1 and Unreal Engine version 4.27.2. The spy runs as a normal user-space application; it does not require special permissions (*e.g.*, research mode [66], which provides access to raw sensor streams) to operate. We typically conduct experiments with only the spy and any victim running; the impact of noise from other background apps is examined in § 4.1.

Data collection. In the experiments, we got IRB approval from our institution and solicited ten volunteers with a diversity of ages, heights, weights, and gender from our university

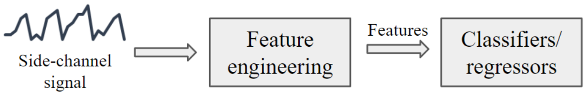


Figure 4: Attack overview.

community. Before the data collection experiments, we introduced these volunteers to a brief training session on Hololens 2 and Meta Quest 2 interaction models. We collect data for the 5 different attack scenarios from each volunteer. Data collection for each attack took around 30 minutes per volunteer. We record performance counter readings at 60 Hz. All collected features are accessible by the spy background program with user-level permissions.

To demonstrate the generalizability of attacks across SDKs, we tested some attacks on Unity and others on Unreal, the two primary SDKs for AR/VR devices. It is possible for any attack to be conducted against either SDK. Specifically, attacks 3 (keystroke monitoring) and 4 (concurrent app fingerprinting) use Unreal, while all the other attacks target Unity applications. The set of counters offered by each SDK is different. For example, in attack 1 (hand gestures inference), we target Unity applications and collect five leakage vectors, including *CPU frame rate*, *GPU frame rate*, *Number of draw calls*, *Vertex count* and *AppMemoryUsage*. Volunteers perform each hand gesture five times for each trial. Attacks 2 (voice commands inference) and 5 (bystander ranging) also target Unity and collect the same performance counters. For attack 2, volunteers trigger each voice command five times for each trial. For attack 5, the experiments were held in the same room with an unchanged surrounding environment. The headset was kept in the same position on the table. Each of the ten volunteers is requested to walk with constant speed in front of the device five times at different preset distances (0.5, 1, 2, 3, 4, and 5 meters).

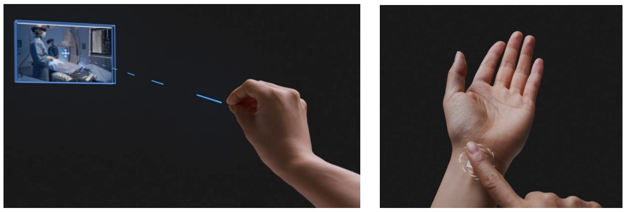
Attacks 3 and 4 (concurrent app fingerprinting) target Unreal. They collect the following performance counters: *Frame time*, *Game thread time*, *Render thread time*, *Number of draw calls* and *Number of primitives* counters from the Unreal Engine RHI. For attack 3, the volunteers are asked to type each digit (0-9) ten times each, while for attack 4, each volunteer opens and closes each foreground application 10 times as the spy collects data.

3.4 Attack workflow

The general attack workflow is shown in Fig. 4, and we expound on each stage below. We split the data into training/testing sets with 80/20%, respectively. 80% of the data were used to do feature engineering and training, and the remaining 20% for evaluation for all attacks.

Feature engineering. After recording side-channel leakages from the malicious app on AR/VR device, our next step is to extract useful features from the time-series data. We use `tsfresh` [11], a Python library that automatically generates hundreds of statistical features from time-series data. We used the Benjamini-Yekutieli procedure [5] to rank all the features and filter the strongly relevant ones based on the p-value.

Classification/Regression. We use these features to train several standard classifier or regressor candidates, namely K Nearest Neighbors (KNN) [26], Decision Tree (DT) [62], Random Forest (RF) [7], Light Gradient Boosting Machine (LightGBM) [24] and weighted majority rule voting (Voting) [54] based classifier/regressor. The voting ensemble method combines a decision tree and a random forest as candidates. The weights of the two candidates are set as identical. The hyperparameter settings for classifiers and regressors are listed in Table 11 and Table 12, respectively. To evaluate the performance of the classifiers, we compute the F1 score (F1), Precision (Prec), and Recall (Rec). Also, to select the best regressor, we calculate the Mean Absolute Error (MAE).



(a) Air tap [45]   (b) Start gesture [45]

Figure 5: Examples of hand gestures.

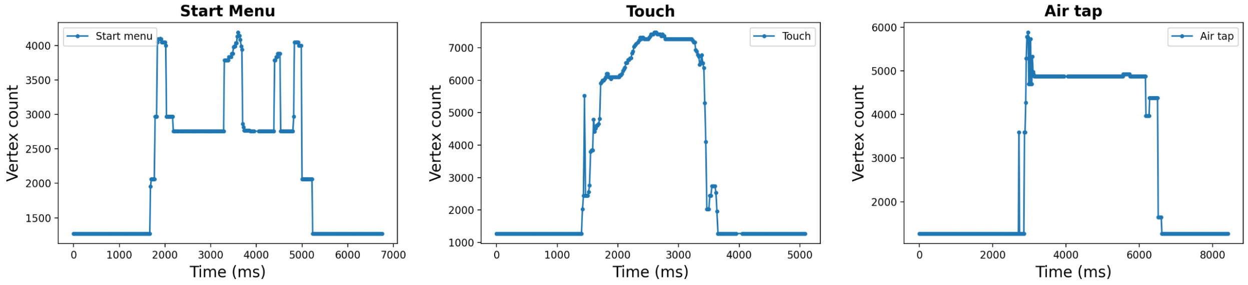
4 Attack Scenario 1: User Interactions

In the first attack scenario, we assume a spy application runs in the background to keep profiling performance counters and memory allocation APIs, while a victim user is interacting with AR/VR device. We will use performance counters and memory allocation APIs to infer hand gestures (both system commands and keystrokes) and voice input behavior.

4.1 Attack 1: Hand gestures inference

Users can interact with digital artifacts (*e.g.*, holograms) in the environment either directly or with hand gestures. We first describe hand gesture inputs on the Hololens 2 and Meta Quest 2, and then our attacks and evaluations.

Hand gestures on Hololens 2. We focus on inferring five basic system-level hand gestures on the Hololens 2 that are part of the standard interaction interface for most applications. The first gesture is *Touch*, which means touching holographic contents directly using the user's index finger. A white touch cursor will be displayed on the user's index finger, which helps the user touch and interact with the item precisely. Second, *Air tap* gesture allows users to interact with distant holograms



(a)

(b)

(c)

Figure 6: Performance counter traces for hand gestures on Hololens 2: (a) *Start menu*; (b) *Touch*; and (c) *Air tap*.

through the help of a hand ray, which emanates from the palm of a user's hand, as shown in Fig. 5a. To select distant holograms pointed to by the hand ray, the user needs to pinch her thumb and index finger together and release them quickly after that. The third interaction is *Start menu* gesture, which opens the start menu. To perform it, the user points at her inner wrist with her palm facing outward as shown in Fig. 5b. The fourth and fifth gestures are *Scale up* and *Scale down* gestures, which enable resizing the hologram by grabbing and stretching its corners.

Hand gestures on Meta Quest 2. Similar to the Hololens 2, users can interact with 3D objects using hand gestures on Meta Quest 2. We track five different hand gestures, including *Palm pinch*, *Point and pinch*, *Touch*, *Scale up* and *Scale down* [37]. *Palm pinch* requires the user to hold her thumb and index finger together to trigger the start menu. *Point and pinch* is similar to *Air tap* on the Hololens – it allows users to select far away holograms by using thumb point holograms and pinch index finger to select. The *Touch*, *Scale up*, and *Scale down* gestures are identical to those on the Hololens 2.

Selected leakage vectors. We notice that different hand gestures introduce distinct rendering contention patterns with the spy program running in the background. Specifically, as a user interacts with holograms using hand gestures, various rendering tasks will be dispatched to the device CPU depending on which hand gesture is being performed. In turn, these rendering workloads leave signatures visible through rendering performance counters: for example, *Vertex count* reveals the number of vertices submitted to the graphics API for rendering.

Fig. 6 shows traces of the rendering performance counter *Vertex count* illustrating unique signatures when the *Start menu*, *Touch*, and *Air tap* gestures are performed. Specifically, before three hand gesture inputs are sent to the device, the value of *Vertex count* is stable at around 1000. However, when the user starts to interact with the device via hand gestures, the AR/VR device needs to render the hands in the immersive world, resulting in the sudden increase of *Vertex count*. Also, since *Touch* and *Air tap* involve interaction with a 3D object while *Start menu* does not, the peak of *Vertex count* of *Start menu* is less than other two gestures. In addition, the *Touch* gesture generates more vertices than *Air tap* gesture, since *Touch* gesture interacts with the object directly while *Air tap* gesture interacts remotely.

Feature engineering and classification. Table 2 summarizes the top 10 features for classifying hand gestures on the Hololens 2 and Meta Quest 2, respectively. Table 15 defines these statistics [10]. The top 3 features for Hololens are benford_correlation (*Vertex count*), approximate_entropy (*CPU frame rate*) and approximate_entropy (*GPU frame rate*). For Quest 2, the top 3 features are sum_values, mean and root_mean_square (*AppMemoryUsage*). We noticed that all top 3 features of the Quest 2 are generated from the *AppMemoryUsage* counter. We conjecture that this is because the hardware configurations of Hololens and Meta are different. For instance, the Quest 2 has a CPU and GPU to accelerate render tasks but the Hololens 2 has an extra HPU to help process render tasks. The *AppMemoryUsage* is more sensitive when different hand gestures are triggered on the Quest than the Hololens.

Table 3 presents the performance results for each classifier using the previously selected features. We found that the voting-based classifier achieves over 89% accuracy for hand gestures inference on the Hololens 2. Similarly, we noticed that the random forest classifier attains the best performance among all classifier candidates for hand gesture inference on Quest 2, with over 93% for F1 score, precision, and recall.

Robustness to Noise: Since the preceding experiments were tested in a relatively quiet environment, other applications may also run on the same AR/VR device, which harms the quality of side-channel leakages. Signal to Noise Ratio (SNR) is often used to evaluate the quality of a side-channel [36]. We use SNR to check whether there is leakage in the presence of other applications. We use the SNR equation: $SNR = \frac{Var_{(Signal)}}{Var_{(Noise)}}$. [1]

Specifically, we collect five traces of *CPU frame rate* and compute the average SNR with other apps (Microsoft Store, Microsoft Edge, and 3D Viewer) running in the background.

[1] $Var_{(Signal)}$ is the variance of the *CPU frame rate* when the victim uses the *Start menu* gesture to interact with the device, and $Var_{(Noise)}$ is the variance of *CPU frame rate* when no hand gestures involved.

	Features	
	Hololens 2	**Quest 2**
CPU frame rate	**approximate_entropy**, sample_entropy, permutation_entropy	median
Number of draw calls	benford_correlation	minimum, quantile
GPU frame rate	**approximate_entropy**, sample_entropy, permutation_entropy	root_mean_square
AppMemoryUsage	maximum, abs_energy	**sum_values**, **mean**, **root_mean_square**, abs_energy, c3
Vertex count	**benford_correlation**	minimum

Table 2: Top 10 features for classifying hand gestures on the Hololens and Quest (top 3 features are bolded).

	Hololens 2			**Quest 2**		
	F1	**Prec**	**Rec**	**F1**	**Prec**	**Rec**
KNN	53.6	55.4	54.2	57.9	58.3	58.8
DT	80.0	80.5	80.0	91.3	91.7	91.3
RF	86.6	86.6	86.7	**93.7**	93.8	93.7
LightGBM	84.7	86.7	85.0	89.0	91.9	90.0
Voting	**89.2**	89.3	89.2	91.3	91.9	91.3

Table 3: Hand gesture inference performance: F1 (%), Precision (%), and Recall (%) on Hololens and Quest.

	Hololens 2			**Quest 2**		
	F1	**Prec**	**Rec**	**F1**	**Prec**	**Rec**
KNN	87.5	87.7	87.5	65.9	73.3	62.0
DT	**93.7**	93.8	93.8	88.1	89.6	88.0
RF	91.2	91.3	91.2	86.0	89.3	86.0
LightGBM	88.9	90.9	89.5	90.3	93.0	90.8
Voting	91.3	92.4	91.3	**93.9**	94.0	94.0

Table 4: Voice command inference performance: F1 (%), Precision (%), and Recall (%) on Hololens and Quest.

Without interference, for the *Start menu* gesture, the measured SNR is 10.51 indicating a high-quality channel. While in the presence of noise the calculated SNR decreases to 6.90, this value is still high, indicating resilience to noise. In particular, an SNR value greater than 1 is considered to be exploitable in recent literature [75].

4.2 Attack 2: Voice commands inference

Users can also communicate with the headset through voice input commands. We launch a similar attack as Section 4.1; however, this attack infers a user's voice commands on both the Quest and Hololens headsets. We first briefly describe the built-in voice commands for each.

Built-in voice input commands on the Hololens 2. We aim to infer five HoloLens-specific commands [43]: *“Go to Start”*, *“Take a picture’*, *“Start/stop recording”*, *“Turn up/down the volume”* and *“Show/hide hand ray”*.

Built-in voice input commands on the Meta Quest 2. To initiate a voice command, the user double-clicks the home button on the right controller. We aim to infer the five Meta built-in voice commands [39]: *“Take a photo”*, *“Start/stop recording”*, *“Turn up/down the volume”*, *“Reset view’* and *“Start casting”*.

Selected leakage vectors. We notice that different voice commands (*“Go to Start”*, *“Take a picture’*, *“Start/stop recording”*) result in different memory usage of the spy program, which is shown in Fig. 7. The sudden increase in memory usage indicates the voice command is triggered. We conjecture it is because the triggered voice command may be allocated to a shared buffer with the background spy program. Also, the peaks of three voice commands are different: *“Start/stop recording”* is the highest, around 780 MB, while *“Go to Start”* only triggers 610 MB.

Feature engineering and classification. We use the same methodology to extract the salient time-series features, shown in the Appendix in Table 13 for Hololens 2 and Meta Quest 2, respectively. Table 4 presents the attack accuracy on the two devices. The Hololens-specific voice commands can be successfully inferred with F-1 score of 93.7% with the decision tree classifier. Using the voting classifier, we can identify the correct type of voice command with high accuracy (F-1 score of 93.9%) on the Meta Quest 2.

4.3 Attack 3: Keystroke monitoring

The next attack targets keystroke monitoring on the virtual keyboard. We noticed that when the user enters keystrokes using the virtual keyboard in the immersive environment, the user has to move her hand and fingers differently for each keystroke based on the key location. This movement in turn generates distinguishable side-channel leakage patterns for each keystroke [70]. Fig. 9 shows example patterns for three different digit inputs collected by the spy program on the Hololens 2. As can be seen, each of the digits has a distinctive pattern. We note that prior work [4] has studied keystroke monitoring in VR using WiFi Channel State Information (CSI) monitored by an external attacker with access to the wireless channel. In contrast, our attack does not require any physical access to the wireless infrastructure.

Attack scenario. Fig. 8 illustrates the attack overview. The victim user switches to a new foreground application (1) and

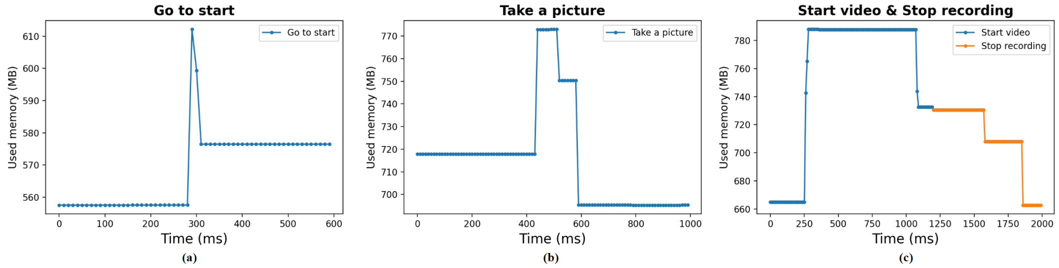


Figure 7: Performance counter traces as a user speaks different voice commands: (a) Go to start; (b) Take a picture; and (c) Start video and Stop recording.

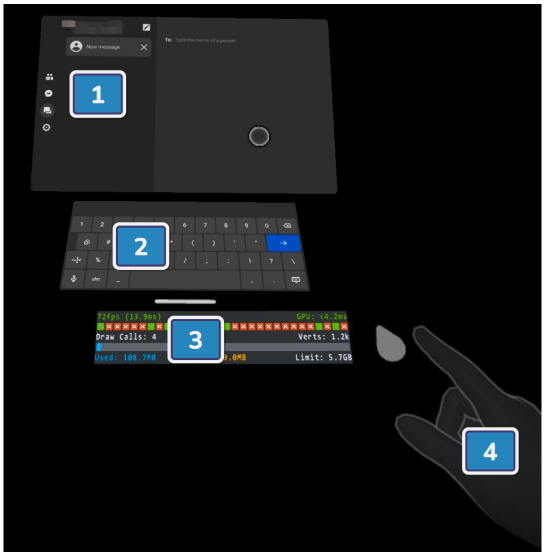


Figure 8: Keystroke monitoring attack scenario. (1) The foreground app (Meta Messenger) (2) Victim user types messages using the built-in board (3) The malicious app records performance counters in the background (4) Victim uses the hand gesture *Point and pinch* to type.

starts typing some sensitive text via the system keyboard (2). In the meantime, the malicious application is recording the performance counters in the background (3). The user enters the sensitive keystrokes via the hand gesture *Point and pinch* directly (4). As proof of concept, we focus on the ten digits (0-9) on the virtual keyboard. The digits can contain sensitive information, for example, credit card numbers, for online payment.

Selected leakage vectors. Fig. 9 shows the values of the counters for these two threads for three digits (0, 2, and 9). These three digits all have different patterns since the hand movements are distinct. The counters *Game thread time* and *Render thread time* track the execution time of two primary threads present in applications using this engine. The game thread is responsible for most application logic, while the render thread is activated to render new frames. We notice that the *Game thread time* counter increases while the *Render thread time* counter decreases simultaneously. A possible explanation of this counter behavior pattern is that when AR/VR devices detect the *touch* gesture, it consumes some resources to keep tracking the hand movement. Since the game thread time reflects the time used for spy application logic, it may be preempted by the hand-tracking task. Also, rendering threads of the spy application may skip updating because of resource contention. As a result, the game thread time increases while the rendering time decreases.

	Hololens 2			Quest 2		
	F1	Prec	Rec	F1	Prec	Rec
KNN	43.3	49.6	44.3	44.1	49.4	44.0
DT	88.7	89.8	88.6	92.1	93.7	92.0
RF	52.1	54.0	52.9	73.7	75.5	75.0
LightGBM	87.5	88.0	88.8	**93.8**	94.8	94.0
Voting	**91.4**	91.7	91.4	90.1	91.6	90.0

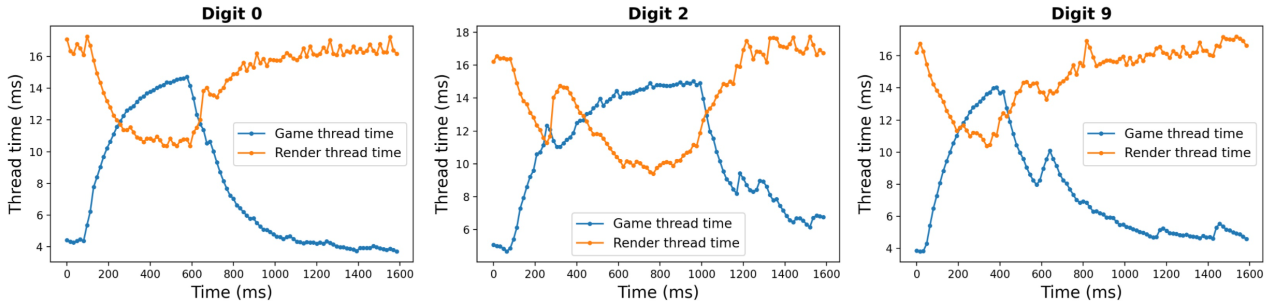
Table 5: Keystroke monitoring performance: F1 (%), Precision (%), and Recall (%) on Hololens and Quest.

Feature engineering and classification. Table 14 in the Appendix summarizes the top performance counters for the two AR/VR devices. Results are presented in Table 5, showing that the voting method performs best with an F-1 score of around 91% on Hololens 2, and the LightGBM classifier achieves 93.8% on Quest 2.

5 Attack Scenario 2: Concurrent Apps

In this scenario, the attacker exploits side-channel leakage to infer information about other applications running on the device. Specifically, we show that a spy can fingerprint an application being opened.

Attack 4: Concurrent app fingerprinting: Information about other concurrent applications can potentially be sensitive: it discloses information about the user activity and could be used to provide context for other attacks such as phishing attacks [14]. The goal of this attack is to infer the launch and identify a concurrent application. We observe that

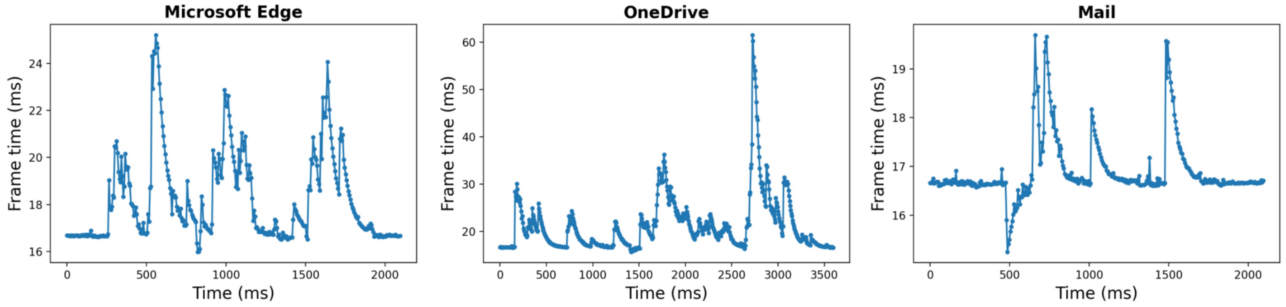


(a)

(b)

(c)

Figure 9: Performance counter traces when a user inputs different digits on a virtual keyboard: (a) 0, (b) 2, and (c) 9.



(a)

(b)

(c)

Figure 10: Performance counter traces when launching applications: (a) Microsoft Edge; (b) OneDrive; and (c) Mail.

different applications create different side-channel patterns when launched on Hololens, which can be observed by the spy program running in the background. This attack was not possible on the Quest 2 headset, where only a few applications (such as Facebook Messenger or Meta chat) are allowed to open in the foreground with other applications, although this may change in the future.

Application launch process and attack preliminaries. When the user launches an application, a system process named *backgroundTaskHost* will handle the launch of the selected application. The user waits until the application renders on the display. We observe that launching an application typically takes several seconds. Meanwhile, the foreground application execution leads to resource contention with the spy program, causing a detectable signature. For instance, we observe that when the user launches another application, the CPU/GPU frame latency of the spy program increases. We use such leakage to fingerprint the application.

Application fingerprinting demonstration. Fig. 10 shows traces of the *Frame time* counter during the launch of three applications. Each application exhibits a distinct pattern. Although applications are rendered in fixed-size windows, they each have different graphics content to generate during launch, resulting in different traces.

Feature engineering and classification. Table 9 in the Appendix summarizes the top-ranked features for the selected rendering performance counters for classification. Table 6 shows the set of evaluated applications and Table 7 presents the classification results. Voting classifier performs best with an F-1 score of 89.3%.

Category	**Application**
Productivity	Calendar, File explorer, OneDrive
Recreation	3D Viewer, Microsoft Store, Movies&TV, Photos
Communication	Cortana, Captive portal flow, Mail, Microsoft Edge, Microsoft Pay

Table 6: Applications tested for the fingerprinting attack.

	F1	**Prec**	**Rec**
KNN	33.7	39.4	35.0
DT	84.7	86.5	85.0
RF	51.3	53.0	50.8
LightGBM	85.8	87.4	86.8
Voting	**89.3**	91.0	89.2

Table 7: Concurrent application inference performance on Hololens 2: F1 (%), Precision (%), and Recall (%).

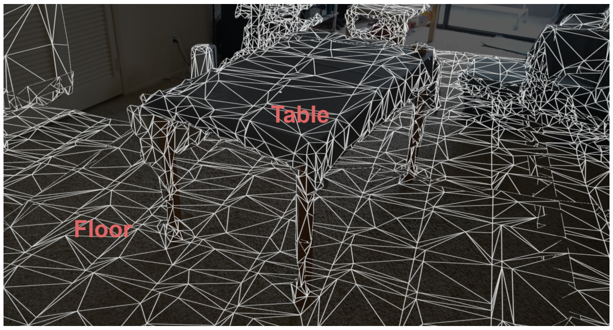


Figure 11: Example of a constructed spatial mesh.

6 Attack Scenario 3: Environment

An attacker can attempt to infer information about the environment viewed by an AR device. The presence of certain items or events in the field of view may result in different processing patterns that can be used to infer information about what is being seen. We illustrate an example of this type of attack by showing that we are able to estimate the distance of a walking bystander from the head-mounted display. This attack turns the AR device into a surveillance device to monitor the surrounding real-world environment.

Privacy concerns of the bystander. AR/VR devices, by definition, interface with the real world and may capture information about a bystander who steps into the field of view of the device. However, the spatial information of the surrounding space is typically encrypted and its raw data cannot be easily accessed [60]. For instance, the Hololens 2 provides a research mode [66] to let the developer access the raw sensor data stream, but this mode is turned off by default, making it impossible to recover visible spatial information via direct sensor-based or vision-based leakages. In contrast, our work shows the possibility of learning about the distance of the bystander using the side-channel leakages via performance counters.

Attack 5: Bystander ranging: Our intuition is that these environmental events may cause the headset to carry out additional rendering or other processing tasks that create signatures in the performance counters, allowing a spy to learn information about the environment. Prior work has raised privacy concerns about the presence or location of bystanders on AR/VR devices [56]. Thus, we build an attack to show that the distance of a walking bystander may be vulnerable to estimation via side-channel leakage.

Attack model. In this attack, a spy program running in the background keeps profiling the rendering performance counters. We consider a scenario when a bystander steps into the field of view of a Hololens.[2] The spy program is instrumented with a Mixed Reality Toolkit Scene Understanding observer [47], which forces the Hololens to generate spatial mesh representations of the real-world environment (no explicit permissions are needed from the user to do so, since spatial mapping is a fundamental part of mixed reality). This spatial processing is typically happening continuously in the background of a MR device [31]; what our instrumentation does is create additional contention by forcing the results of the spatial processing to be rendered. As a result, the spy program turns the Hololens into a monitoring device, which keeps perceiving the surrounding space. On the Hololens, the spatial surfaces appear to be covered by triangle meshes, as shown in Fig. 11. Note that Quest 2 disallows access to the spatial mapping data, so we could not perform these experiments with it.

The spy program is developed via the Unity platform, and the spatial meshes are set to update at interval of 1 second. In Fig. 12a, we plot the *CPU frame rate* as a bystander passes by. We observe that *CPU frame rate* decreases briefly when the bystander steps into the field of view, and also again when the bystander steps out of view. We believe that this effect occurs because when the bystander steps into the frame, the Hololens' spatial processing detects the bystander and generates a mesh to cover the bystander, causing *CPU frame rate* to decrease from around 60 to 18. Then when the bystander steps out of the frame of Hololens, the mesh is discarded immediately for the removed spatial surfaces. During this mesh removal process, we notice that *CPU frame rate* also decreases, from 60 to 37. Thus by tracking the side-channel leakages, the adversary can monitor when the bystander steps into the field of view and when she leaves.

Impact of bystander distance. We also examined the impact of bystander distance on the side-channel fingerprint. The intuition is that new spatial objects near and in front of the user would be given priority, and their meshes would be rendered first. Also, due to the perspective projection, near objects would appear larger in the field of view than far objects and thus would be covered with more meshes than a distant object, creating a location-dependent fingerprint.

We performed experiments to see how distance influences the *CPU frame rate*. Volunteer bystanders stood in front of Hololens at different distances, ranging from 0.5 meters to 5 meters (0.5, 1, 2, 3, 4, and 5 meters). The bystander passes by the Hololens at a constant walking speed. Fig. 12 shows the performance counter traces of *CPU frame rate* for different distances. When the distance is smaller, the *CPU frame rate* tends to drop more. For instance, *CPU frame rate* dips to 18 when the distance is 0.5 meters in Fig. 12 (a), while it only drops to 35 when the distance increases to 2 meters in Fig. 12 (b). In some scenarios where the bystander is far from the Hololens, the spatial mesh is not triggered to cover the bystander, and the bystander is not detected.

Feature engineering and regression. The filtered top features are summarized in Table 10 in the Appendix. We trained several standard regression models to infer the distance be-

[2] According to [42], the Hololens can still work for several minutes even if the user removes the headset.

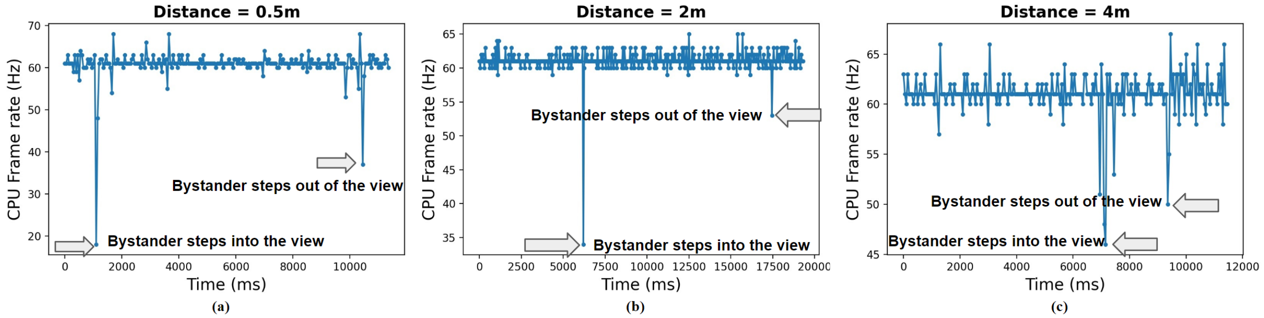


Figure 12: Side-channel leakage pattern for a bystander moving into view at different distances: (a) Distance = 0.5m; (b) 2m; (c) 4m. As the bystander becomes closer, the CPU frame rate drops more.

	KNN	DT	RF	LightGBM	Voting
MAE	0.401	**0.103**	0.257	0.279	0.164

Table 8: Bystander ranging mean absolute error (meters).

tween the device and the bystander. Table 8 shows that the decision tree performs best, with mean absolute error of 10.3 cm, showing that the attack can estimate bystander distance with high accuracy.

Limitations: We note that this is a proof-of-concept attack that identifies walking-bystander distance in a known environment. It would be interesting to explore how well this attack can be generalized to arbitrary environments and with headset movement.

7 Potential Mitigations

We consider two general classes of mitigations: (1) managing access to the performance counters and (2) detecting suspicious contention. These solutions' directions are similar to prior side-channel attacks that rely on tracked states such as performance counters.

Managing access to performance counters: We note that MR presents unique challenges in that part of the state is relevant to multiple applications, including the model of the surrounding environment and the inputs from the sensors. Thus, it is important to think about managing access to the performance counters and other system resources in a way that does not interfere with an application's legitimate access to the needed state.

Completely blocking access to potentially leaky APIs and counters is impractical since legitimate applications sometimes rely on these APIs for online performance tuning. One possibility is to allow performance counter access only to the current foreground or active app that the user is focusing on, thereby preventing a spy from listening in. However, determining which app is active in MR is challenging because multiple MR apps could be interspersed throughout the field of view, providing different services (*e.g.*, a gallery app displays multiple virtual paintings while simultaneously a virtual board game app is visible). It is difficult to determine which app(s) the user is focused on at any given time. This is in contrast to traditional displays such as monitors or tablets, where apps typically have a single contiguous window rendered on a confined rectangular viewpoint. Eye gaze could be potentially used as a proxy for attention and hence to determine the active MR app.

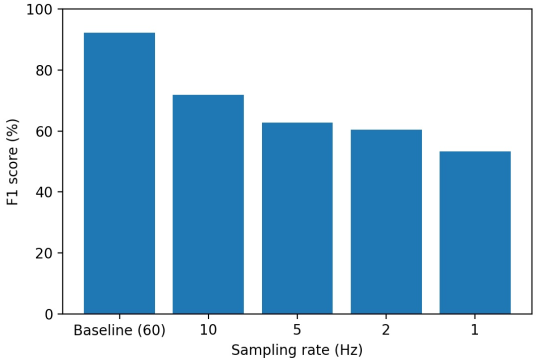


Figure 13: Hand gesture classification accuracy with reduced sampling rate.

Reducing the precision or rate of performance counters could reduce the effectiveness of the spy [50]. However, this could result in side effects on legitimate applications, interfering with their ability to operate correctly. Fig. 13 shows the effect of limiting the sampling rate on the hand gesture recognition attack. While the accuracy drops significantly, there remains some signal even at low sampling rates. We conjecture that since the event we are capturing (the hand gestures) takes multiple performance counter features as input, even sampling at such a low rate retains some useful signal.

Another possibility is to delegate responsibility to the user to set permissions for what APIs each app can access through an explicit permissions management system. Such permissions systems have been used for other types of MR data such as MR content shared between multiple users [58], visual camera data [20], or general sensor streams [21, 57], but not for mixed reality performance counters. However, this pushes the responsibility of determining when these performance counters can be accessed by different apps to the user, who may be unclear on the consequences of granting such permission and lack informed consent. Moreover, there may be technical difficulties in implementing a permissions system and providing the appropriate user interface.

Monitoring for abnormal monitoring and/or contention: Several of the proposed attacks rely on creating contention on the shared resources. Moreover, the attacks rely on repeated accesses to the monitoring APIs to collect the time series data. One possibility is to monitor this behavior to detect the presence of such behavior (for example, by hooking the leaky APIs [73] or using hardware support [52]). Once such behavior is detected, different defenses could be triggered to interfere with the ability of the spy to carry out the attack. As with other intrusion detection systems, false positives and overhead are potential drawbacks of this approach.

8 Related Work

Software-based information leakage on AR/VR devices. Some work has been done on stealing sensitive information from MR devices through software vulnerabilities. Prior works show that malicious applications can probe the network traffic information to infer the location of the victim user in a crowd-sourced AR app [40, 60]. Other works [13, 17, 34] also focus on location estimation based on 3D cloud point data collected by AR devices. Recently, Casey et al. [8] demonstrate that the configuration files of OpenVR and SteamVR API leak information about the user and can be modified to disorient the user. Yarramreddy et al. [74] and OVRSeen [65] show that sensitive information can be reconstructed from VR network traces. Arya [29] prevents visual conflicts between holograms and a user's perception of the real world. Several works also focus on the privacy of input data streams [9, 19–22]. Several works [35, 61] show that malicious AR applications can infer keystrokes by tracking a user's head motions. In the multi-app scenario, Lebeck et al. [28] discuss challenges in visual conflicts between multiple apps. However, none of these works investigate performance counters as a source of information leakages as we do.

MR information leakages requiring physical access: Another class of information leakage attacks requires physical access to the MR hardware or the environment where the hardware is being used. Several works infer the keystrokes in the immersive environment by WiFi CSI side-channel information [4] and vision and sensor-based side-channel leakages [32]. Kohno et al. [25] investigate whether a malicious bystander can exploit the information leakage from the visual channel on the augmented reality headset to infer sensitive information that an augmented reality user is seeing. In contrast, the attacks we propose in this work do not require physical access to the hardware.

Side-channel attacks via micro-architectural hints. Our work relies on performance counters provided by software interface (Unity/Unreal XR SDK) to construct side-channel on the AR/VR devices. It differs from the low-level hardware performance counters used in the prior works under CPU [1, 6], GPU resources [50, 71]. Moreover, the hardware components (*e.g.*, CPU, GPU, HPU, system memory) on AR/VR devices can be potential attack targets through high-precision prime-probe attacks on data caches [23, 33], rowhammer attacks on DRAM [49]. We leave this as future work.

9 Concluding Remarks

This paper explored side-channel attacks on AR/VR devices. We present a classification of the types of attacks possible based on the target information that the attacker seeks to compromise. We identify three general classes of targets: the behavior of the user by inferring their interactions, the behavior of other applications, or information about the visible environment. We demonstrate a number of successful end-to-end attacks from these three categories. Our attacks demonstrate that this attack vector is a threat to AR/VR systems.

Acknowledgments

The authors would like to thank the insightful reviews and helpful suggestions from our anonymous shepherd and reviewers. The authors also thank the user study participants for volunteering their time. This work was partially supported by the NSF grants CNS-1942700, CNS-2053383, CCF-2212426, and a Meta faculty research award.

References

[1] Manaar Alam, Astikey Singh, Sarani Bhattacharya, Kuheli Pratihar, and Debdeep Mukhopadhyay. In-situ extraction of randomness from computer architecture through hardware performance counters. In *International Conference on Smart Card Research and Advanced Applications*. Springer, 2019.

[2] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri. Port contention for fun and profit. In *IEEE Symposium on Security and Privacy (S&P)*, 2019.

[3] Marco Alecci, Riccardo Cestaro, Mauro Conti, Ketan Kanishka, and Eleonora Losiouk. Mascara: A novel attack leveraging android virtualization. *arXiv preprint arXiv:2010.10639*, 2020.

[4] Abdullah Al Arafat, Zhishan Guo, and Amro Awad. Vr-spy: A side-channel attack on virtual key-logging in vr headsets. In *IEEE Virtual Reality and 3D User Interfaces (VR)*, 2021.

[5] Yoav Benjamini and Daniel Yekutieli. The control of the false discovery rate in multiple testing under dependency. *Annals of statistics*, pages 1165–1188, 2001.

[6] Sarani Bhattacharya and Debdeep Mukhopadhyay. Utilizing performance counters for compromising public key ciphers. *ACM Transactions on Privacy and Security (TOPS)*, 21(1):1–31, 2018.

[7] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[8] Peter Casey, Ibrahim Baggili, and Ananya Yarramreddy. Immersive virtual reality attacks and the human joystick. *IEEE Transactions on Dependable and Secure Computing*, 18(2):550–562, 2019.

[9] Song Chen, Zupei Li, Fabrizio Dangelo, Chao Gao, and Xinwen Fu. A case study of security and privacy threats from augmented reality (ar). In *IEEE International Conference on Computing, Networking and Communications (ICNC)*, 2018.

[10] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W Kempa-Liehr. Overview on extracted features from tsfresh library. https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html.

[11] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77, 2018.

[12] HTC Corporation. HTC Vive. https://www.vive.com/us/.

[13] Jaybie A de Guzman, Kanchana Thilakarathna, and Aruna Seneviratne. A first look into privacy leakage in 3d mixed reality data. In *European Symposium on Research in Computer Security*. Springer, 2019.

[14] Wenrui Diao, Xiangyu Liu, Zhou Li, and Kehuan Zhang. No pardon for the interruption: New inference attacks on android through interrupt timing analysis. In *IEEE Symposium on Security and Privacy (S&P)*, 2016.

[15] Unreal Engine. Unreal engine xr sdk. https://docs.unrealengine.com/4.26/en-US/SharingAndReleasing/XRDevelopment/.

[16] Lucas Silva Figueiredo, Benjamin Livshits, David Molnar, and Margus Veanes. Prepose: Privacy, security, and reliability for gesture-based programming. In *IEEE Symposium on Security and Privacy (S&P)*, 2016.

[17] Jaybie Agullo de Guzman, Aruna Seneviratne, and Kanchana Thilakarathna. Unravelling spatial privacy risks of mobile mixed reality data. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*, 5(1):1–26, 2021.

[18] John Haas. A history of the unity game engine (dissertation). *Worcester Polytechnic Institute*, 2014.

[19] Jassim Happa, Mashhuda Glencross, and Anthony Steed. Cyber security threats and challenges in collaborative mixed-reality. *Frontiers in ICT*, 6:5, 2019.

[20] Jinhan Hu, Andrei Iosifescu, and Robert LiKamWa. Lenscap: split-process framework for fine-grained visual privacy control for augmented reality apps. In *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2021.

[21] Suman Jana, David Molnar, Alexander Moshchuk, Alan Dunn, Benjamin Livshits, Helen J Wang, and Eyal Ofek. Enabling fine-grained permissions for augmented reality applications with recognizers. In *USENIX Security Symposium*, 2013.

[22] Suman Jana, Arvind Narayanan, and Vitaly Shmatikov. A scanner darkly: Protecting user privacy from perceptual applications. In *IEEE Symposium on Security and Privacy*, 2013.

[23] Mehmet Kayaalp, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Aamer Jaleel. A high-resolution side-channel attack on last-level cache. In *Design Automation Conference (DAC)*, 2016.

[24] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[25] Tadayoshi Kohno, Joel Kollin, David Molnar, and Franziska Roesner. Display leakage and transparent wearable displays: Investigation of risk, root causes, and defenses (technical report). Technical report, Microsoft Research, 2015.

[26] Oliver Kramer. K-nearest neighbors. In *Dimensionality reduction with unsupervised nearest neighbors*, pages 13–23. Springer, 2013.

[27] Magic Leap. Magic leap 2. https://www.magicleap.com/en-us/.

[28] Kiron Lebeck, Tadayoshi Kohno, and Franziska Roesner. Enabling multiple applications to simultaneously augment reality: Challenges and directions. In *ACM International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2019.

[29] Kiron Lebeck, Kimberly Ruth, Tadayoshi Kohno, and Franziska Roesner. Securing augmented reality output. In *IEEE symposium on security and privacy (S&P)*, 2017.

[30] Seungsoo Lee, Changhoon Yoon, and Seungwon Shin. The smaller, the shrewder: A simple malicious application can kill an entire sdn environment. In *ACM Workshop on Security in Software Defined Networks & Network Function Virtualization*, 2016.

[31] Peiliang Li, Tong Qin, Botao Hu, Fengyuan Zhu, and Shaojie Shen. Monocular visual-inertial state estimation for mobile augmented reality. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2017.

[32] Zhen Ling, Zupei Li, Chen Chen, Junzhou Luo, Wei Yu, and Xinwen Fu. I know what you enter on gear vr. In *IEEE Conference on Communications and Network Security (CNS)*, 2019.

[33] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *IEEE symposium on security and privacy*, 2015.

[34] Hongbin Liu, Jinyuan Jia, and Neil Zhenqiang Gong. Pointguard: Provably robust 3d point cloud classification. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[35] Shiqing Luo, Xinyu Hu, and Zhisheng Yan. Hololologger: Keystroke inference on mixed reality head mounted displays. In *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2022.

[36] Stefan Mangard. Hardware countermeasures against dpa–a statistical analysis of their effectiveness. In *Cryptographers' Track at the RSA Conference*, pages 222–235. Springer, 2004.

[37] Meta. Getting started with hand tracking on meta quest 2 and meta quest. https://www.meta.com/help/quest/articles/headsets-and-accessories/controllers-and-hand-tracking/hand-tracking-quest-2/.

[38] Meta. Meta quest 2. https://store.facebook.com/quest/products/quest-2/.

[39] Meta. What you can say with Voice Commands on Meta Quest. https://www.meta.com/help/quest/articles/in-vr-experiences/oculus-features/what-can-I-say-with-voice-commands/, 2022.

[40] Gabriel Meyer-Lee, Jiacheng Shang, and Jie Wu. Location-leaking through network traffic in mobile augmented reality applications. In *IEEE International Performance Computing and Communications Conference (IPCCC)*, 2018.

[41] Microsoft. Microsoft hololens 2. https://www.microsoft.com/en-us/hololens/buy.

[42] Microsoft. Microsoft hololens 2 sleep behavior. https://learn.microsoft.com/en-us/hololens/hololens2-setup#sleep-behavior.

[43] Microsoft. Voice input. https://learn.microsoft.com/en-us/windows/mixed-reality/design/voice-input.

[44] Microsoft. Window memorymanager api. https://docs.microsoft.com/en-us/uwp/api/windows.system.memorymanager?view=winrt-22000.

[45] Microsoft. Getting around hololens 2. https://docs.microsoft.com/en-us/hololens/hololens2-basic-usage, 2021.

[46] Microsoft. Spatial mapping - mixed reality. https://docs.microsoft.com/en-us/windows/mixed-reality/design/spatial-mapping, 2021.

[47] Microsoft. Scene understanding observer - Mixed Reality Toolkit. https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/features/spatial-awareness/scene-understanding?view=mrtkunity-2021-05, 2022.

[48] Charlie Miller. Mobile attacks and defense. *IEEE Security & Privacy*, 9(4):68–70, 2011.

[49] Onur Mutlu and Jeremie S Kim. Rowhammer: A retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(8):1555–1571, 2019.

[50] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. Rendered insecure: Gpu side channel attacks are practical. In *ACM SIGSAC conference on computer and communications security*, 2018.

[51] Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L Davidson, Sameh Khamis, Mingsong Dou, et al. Holoportation: Virtual 3d teleportation in real-time. In *ACM Symposium on user interface software and technology (UIST)*, 2016.

[52] Meltem Ozsoy, Caleb Donovick, Iakov Gorelik, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Malware-aware processors: A framework for efficient online malware

detection. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[53] Riccardo Paccagnella, Licheng Luo, and Christopher W Fletcher. Lord of the ring (s): Side channel attacks on the cpu on-chip ring interconnect are practical. In *USENIX Security Symposium*, 2021.

[54] Robi Polikar. Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, 6(3):21–45, 2006.

[55] Attia Qamar, Ahmad Karim, and Victor Chang. Mobile malware attacks: Review, taxonomy & future directions. *Future Generation Computer Systems*, 97:887–909, 2019.

[56] Franziska Roesner and Tadayoshi Kohno. Security and privacy for augmented reality: Our 10-year retrospective. In *International Workshop on Security for XR and XR for Security (VR4Sec)*, 2021.

[57] Franziska Roesner, David Molnar, Alexander Moshchuk, Tadayoshi Kohno, and Helen J Wang. World-driven access control for continuous sensing. In *ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[58] Kimberly Ruth, Tadayoshi Kohno, and Franziska Roesner. Secure multi-user content sharing for augmented reality applications. In *USENIX Security Symposium*, 2019.

[59] Andrew Sanders. *An introduction to Unreal engine 4*. CRC Press, 2016.

[60] Jiacheng Shang, Si Chen, Jie Wu, and Shu Yin. Arspy: Breaking location-based multi-player augmented reality application for user location tracking. *IEEE Transactions on Mobile Computing*, 2020.

[61] Carter Slocum, Yicheng Zhang, Nael Abu-Ghazaleh, and Jiasi Chen. Going through the motions: Ar/vr typing inference using head motion tracking. In *USENIX Security Symposium*, 2023.

[62] Yan-Yan Song and LU Ying. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130, 2015.

[63] Bob Tellez. How to improve game thread cpu performance in unreal engine. https://www.unrealengine.com/en-US/blog/how-to-improve-game-thread-cpu-performance, 2014.

[64] Elene Terry. Silicon at the heart of hololens 2. In *IEEE Hot Chips 31 Symposium (HCS)*, 2019.

[65] Rahmadi Trimananda, Hieu Le, Hao Cui, Janice Tran Ho, Anastasia Shuba, and Athina Markopoulou. OVRSEEN: Auditing Network Traffic and Privacy Policies in Oculus VR. *USENIX Security*, 2022.

[66] Dorin Ungureanu, Federica Bogo, Silvano Galliani, Pooja Sama, Casey Meekhof, Jan Stühmer, Thomas J Cashman, Bugra Tekin, Johannes L Schönberger, Pawel Olszta, et al. Hololens 2 research mode as a tool for computer vision research. *arXiv preprint arXiv:2008.11239*, 2020.

[67] Unity. Unity engine FrameTimingManager API. https://docs.unity3d.com/ScriptReference/FrameTimingManager.html.

[68] Unity. Unity xr sdk. https://docs.unity3d.com/Manual/xr-sdk.html.

[69] Unreal. RHI (rendering hardware interface). https://docs.unrealengine.com/4.26/en-US/API/Runtime/RHI/.

[70] Daimeng Wang, Ajaya Neupane, Zhiyun Qian, Nael B. Abu-Ghazaleh, Srikanth V. Krishnamurthy, Edward J. M. Colbert, and Paul L. Yu. Unveiling your keystrokes: A cache-based side-channel attack on graphics libraries. In *Network and Distributed System Security Symposium (NDSS)*, 2019.

[71] Junyi Wei, Yicheng Zhang, Zhe Zhou, Zhou Li, and Mohammad Abdullah Al Faruque. Leaky DNN: Stealing deep-learning model secret with GPU context-switching side-channel. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020.

[72] Tao Wei, Min Zheng, Hui Xue, and Dawn Song. Apple without a shell–ios under targeted attack. In *Virus Bulletin Conference*, 2014.

[73] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, 5:32–39, 2007.

[74] Ananya Yarramreddy, Peter Gromkowski, and Ibrahim Baggili. Forensic analysis of immersive virtual reality social applications: a primary account. In *IEEE Security and Privacy Workshops*, 2018.

[75] Yicheng Zhang, Rozhin Yasaei, Hao Chen, Zhou Li, and Mohammad Abdullah Al Faruque. Stealing neural network structure through remote fpga side-channel analysis. *IEEE Transactions on Information Forensics and Security*, 16:4377–4388, 2021.

[76] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *ACM Conference on Data and Application Security and Privacy*, 2012.

[77] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In *Network and Distributed System Security Symposium (NDSS)*, volume 25, pages 50–52, 2012.

[78] David J Zielinski, Hrishikesh M Rao, Mark A Sommer, and Regis Kopper. Exploring the effects of image persistence in low frame rate virtual environments. In *IEEE Virtual Reality (VR)*, 2015.

Appendix

We show the top 10 features extracted from performance counters in concurrent application inference attack, bystander ranging inference attack, voice commands inference attack, and keystrokes monitoring attack in Tables 9, 10, 13, and 14 respectively. Also, the hyper-parameter settings for classifiers and regressors are listed in Table 11 and Table 12 respectively. Besides, we provide definitions for the extracted features in Table 15.

Performance counters	**Features**
Frame time	agg_linear_trend
Game thread time	longest_strike_above_mean, autocorrelation
Render thread time	quantile, agg_linear_trend, median
Number of draw calls	benford_correlation, quantile
Number of primitives	benford_correlation, agg_autocorrelation

Table 9: Top 10 features for classifying concurrent application inference.

Performance counters	**Features**
CPU frame rate	maximum
Number of draw calls	minimum
GPU frame rate	maximum, agg_linear_trend
AppMemoryUsage	quantile, cwt_coefficients
Vertex counts	minimum, quantile, agg_linear_trend, cwt_coefficients

Table 10: Top 10 features for inference bystander ranging on Hololens 2.

Classifiers	**Settings**
KNN	$n\_neighbors = 3, leaf\_size = 50, p = 1, weights = uniform$
DT	$criterion = entropy, splitter = random, max\_depth = 10, min\_samples\_split = 2$
RF	$criterion = entropy, max\_depth = 10, n\_estimators = 100$
LightGBM	$boosting\_type = gbdt, num\_leaves = 10, max\_depth = 10, n\_estimators = 100$
Voting	$classifiers = \{DecisionTree, RandomForest\}, voting = soft, weights = [1, 1]$

Table 11: Hyper-parameter settings for classifiers

Regressors	**Settings**
KNN	$n\_neighbors = 5, leaf\_size = 30, p = 2, weights = uniform$
DT	$criterion = squared\_error, splitter = best, min\_samples\_split = 1$
RF	$criterion = squared\_error, n\_estimators = 100$
LightGBM	$boosting\_type = gbdt, num\_leaves = 31, n\_estimators = 100$
Voting	$regressors = \{DecisionTree, RandomForest\}, voting = soft, weights = [1, 1]$

Table 12: Hyper-parameter settings for regressors

	Features	
	Hololens 2	**Quest 2**
CPU frame rate	quantile, root_mean_square, abs_energy	quantile, maximum
Number of draw calls	minimum, quantile	fft_coefficient
GPU frame rate	quantile, root_mean_square, abs_energy	quantile, maximum,
Used Memory	benford_correlation	quantile, median, abs_energy
Vertex counts	fft_aggregated	sum_values, mean

Table 13: Top 10 features for classifying voice commands on the Hololens 2 and Meta Quest 2.

	Features	
	Hololens 2	**Quest 2**
Frame time	quantile, minimum	fft_coefficient, median
Game thread time	benford_correlation, change_quantiles, median, c3, sum_values	maximum, agg_linear_trend, quantile
Render thread time	fft_coefficient	mean, root_mean_square
Number of draw calls	maximum	cwt_coefficients, quantile
Number of primitives	quantile	root_mean_square

Table 14: Top 10 features for classifying keystrokes on the Hololens 2 and Meta Quest 2.

Features	**Definition**
abs energy	Returns the absolute energy of the time series.
agg autocorrelation	Descriptive statistics on the autocorrelation of the time series.
agg linear trend	Calculates a linear least-squares regression for values of the time series.
approximate entropy	Implements the approximate entropy algorithm on the time series.
autocorrelation	Calculates the autocorrelation of the specified lag.
benford correlation	Useful for anomaly detection applications.
c3	Uses c3 statistics to measure non linearity in the time series.
change quantiles	First fixes a corridor given by the quantiles ql and qh of the distribution of time series.
cwt coefficients	Calculates a Continuous wavelet transform for the Ricker wavelet.
fft aggregated	Returns descriptive statistics of the absolute Fourier transform spectrum.
fft coefficient	Calculates the Fourier coefficients of the time series.
longest strike above mean	Returns the length of the longest consecutive subsequence that is bigger than the mean.
maximum	Calculates the highest value of the time series.
mean	Returns the mean of the time series.
median	Returns the median of the time series.
minimum	Calculates the lowest value of the time series.
permutation entropy	Calculate the permutation entropy of the time series.
quantile	Calculates the q quantile of the time series.
root mean square	Returns the root mean square of the time series.
sample entropy	Calculate and return sample entropy of the time series.
sum values	Calculates the sum over the time series.

Table 15: Definitions of performance counter features.