Side-channel Inference of User Activities in AR/VR Using GPU Profiling

Seonghun Son* Chandrika Mukherjee◇ Reham Mohamed Aburas† Berk Gulmezoglu* Z. Berkay Celik◇
*Iowa State University ◇Purdue University †American University of Sharjah
Emails:{seonghun, bgulmez}@iastate.edu, {cmukherj, zcelik}@purdue.edu, raburas@aus.edu

***Abstract*—Over the past decade, AR/VR devices have drastically changed how we interact with the digital world. Users often share sensitive information, such as their location, browsing history, and even financial data, within third-party apps installed on these devices, assuming a secure environment protected from malicious actors. Recent research has revealed that malicious apps can exploit such capabilities and monitor benign apps to track user activities, leveraging fine-grained profiling tools, such as performance counter APIs. However, app-to-app monitoring is not feasible on all AR/VR devices (e.g., Meta Quest), as a concurrent standalone app execution is disabled. In this paper, we present OVRWATCHER, a novel side-channel primitive for AR/VR devices that infers user activities by monitoring low-resolution (1Hz) GPU usage via a background script, unlike prior work that relies on high-resolution profiling. OVRWATCHER captures correlations between GPU metrics and 3D object interactions under varying speeds, distances, and rendering scenarios, without requiring concurrent app execution, access to application data, or additional SDK installations. We demonstrate the efficacy of OVRWATCHER in fingerprinting both standalone AR/VR and WebXR applications. OVRWATCHER also distinguishes virtual objects, such as products in immersive shopping apps selected by real users and the number of participants in virtual meetings, thereby revealing users' product preferences and potentially exposing confidential information from those meetings. OVRWATCHER achieves over 99% accuracy in app fingerprinting and over 98% accuracy in object-level inference.**

I. INTRODUCTION

Augmented Reality (AR) and Virtual Reality (VR) platforms have reshaped industries and are redefining how we interact with the digital and physical worlds [68], [72]. These immersive systems integrate virtual elements into real-world environments (AR) or create entirely simulated environments (VR). This offers new ways of interaction, visualization, and engagement with other people and the environment. Several companies have designed their own AR/VR devices, e.g., Meta Quest [70], [71], Microsoft HoloLens [54], and Apple Vision Pro [6]. These devices comprise CPUs, GPUs, neural engines, various sensors (e.g., visible light cameras and infrared cameras), and audio components for real-time processing of multimodal sensory information and rendering [76], [87].

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.231302
www.ndss-symposium.org

The myriad of sensors and powerful processors in AR/VR devices enable users to become deeply immersed in digital content, whether in fully virtual environments or blended physical–digital scenes. However, recent research has shown that these capabilities also introduce novel side-channel vulnerabilities in these devices. These works include virtual keystroke inference using WiFi signals [5] or embedded 2D infrared sensors [63], as well as avatar typing observation [96], head movement analysis [78], or a combination of computer vision methods and motion sensors [41]. Additionally, other studies have explored the inference of visual and audio activities during device charging [40], and the reconstruction of high-quality vital signals and speech content, employing embedded motion sensors in AR/VR headsets [11], [97].

A recent work [98] proposed side-channel attacks that recover gestures, voice commands, keystrokes, and detect bystanders using a concurrent malicious app on AR/VR devices. These attacks exploit memory allocation APIs and performance counters exposed through SDKs provided by game engines such as Unity and Unreal. Although the aforementioned work [98] uses CPU and GPU frame rates obtained via SDKs as attack vectors, it faces three key limitations.

First, it requires a concurrently running standalone background app to access these performance counters. However, devices like the Meta Quest restrict concurrent app execution, apart from a few Meta-approved apps (e.g., Messenger and Meta Chat), rendering the attack infeasible. Second, their approach relies on high-resolution profiling (60 Hz), which makes it detectable and easily prevented by lowering the resolution of the profiling tool. Lastly, their work focuses on a small set of attack attributes and performance counters without modeling the detailed relationship between 3D object rendering and GPU usage. Hence, limited sensitive data, such as built-in voice commands and simple gestures, can be captured.

In this paper, we revisit the threat landscape of GPU-based side-channel attacks in immersive environments by focusing on Meta devices, which expose GPU metrics with unusually high and fine-grained correlation to application behavior. We demonstrate that even low-resolution (1Hz) profiling of these metrics can reveal fine-grained information and leak sensitive application activity. To this end, we introduce OVRWATCHER which leverages the built-in GPU profiling tool [16] to fingerprint user activity with over 98% accuracy, without relying on concurrent app execution or additional SDKs. We show that even profiling a single GPU metric is sufficient to leak

fine-grained information, which underscores the practicality and stealthiness of this attack vector. These findings thus reveal critical gaps in existing mitigation strategies [61], and highlight the urgent need for more robust protections on current and future XR platforms.

We therefore quantify these gaps by evaluating OVRWATCHER on XR platforms. OVRWATCHER achieves over 99% accuracy in fingerprinting standalone AR/VR applications, which are primarily rendered in 3D or occupy the full immersive space, thus exerting a stronger impact on GPU metrics. Despite the constrained setting of 2D, it achieves 99% accuracy in fingerprinting WebXR apps, and with two GPU metrics yielding more than 94% accuracy.

As suggested by prior works in the mobile domain [58], [59], identifying user interests can be leveraged for targeted advertising. Motivated by this, we demonstrate the feasibility of identifying virtual objects within immersive environments. Through case studies, we show that OVRWATCHER can fingerprint realistic products in immersive shopping apps and detect participant counts in virtual meeting apps with over 98% accuracy. We further validate the effectiveness of OVRWATCHER through a user study, where participants interact with the real-world application Meta Layout [52], achieving accuracy of up to 88%. These attacks broaden our understanding of the threat models faced by emerging immersive AR/VR systems.

Our attack operates even on platforms such as the Meta Quest by leveraging built-in GPU profiling tools, specifically, Meta's `ovrgpuprofiler` tool [16]. This attack requires no elevated privileges, physical access, or additional SDK installations, and it overcomes the key limitation of prior work that depends on concurrent application execution. In contrast to existing AR/VR side-channel attacks that primarily detect keystrokes or application usage, OVRWATCHER is the first to reveal low-level GPU metrics that can expose fine-grained information within an app's immersive environment.

In summary, we make the following contributions:

- We present OVRWATCHER, a method that uses a built-in GPU profiler with low (1Hz) resolution to perform side-channel attacks on AR/VR devices, without requiring additional SDKs or concurrent app execution.
- We systematically analyze how virtual object rendering in AR/VR affects GPU metrics, revealing indicators correlated with scene complexity and user interactions.
- Through case studies, we show that OVRWATCHER achieves near 100% accuracy in tracking foreground standalone AR/VR and WebXR apps, and inferring fine-grained user activities, such as identifying products in a shopping app and participant counts in a private meeting app, highlighting the significant privacy risks posed by low-resolution GPU side channels in AR/VR.
- We demonstrate that virtual objects selected by real participants can be distinguished with up to 88% accuracy despite the noise introduced by the surrounding environment.
- We evaluate OVRWATCHER in a cross-device setting, showing its effectiveness even with limited GPU metrics, achieving over 93% accuracy across all case studies.

Responsible Disclosure. We have disclosed our findings with the Meta Quest development team through Meta Bug Bounty [51]. The Meta Security Team acknowledged these findings in the Meta Quest series and recognized our contribution with a Meta Bounty Award (Detailed in Section X).

II. BACKGROUND

A. Extended Reality (XR)

Extended Reality (XR) encompasses technologies such as Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR) that integrate digital elements into the real-world environment to varying extents. XR is rapidly advancing with applications in gaming, healthcare, education, training, and data visualization. VR fully immerses users in an alternate reality, isolating them from their physical surroundings, as demonstrated by gaming apps such as Beat Saber [48]. In contrast, AR overlays digital elements onto the user's physical environment, usually without allowing direct interaction between the digital and physical elements, as seen in apps such as Pokémon Go [64] and IKEA Place [35]. MR, a subset of XR, merges VR and AR, enabling users to view and interact with virtual objects within their physical environment. For example, Figmin XR [49] allows users to create, collect, and play in an MR environment. In addition to these standalone apps, both Meta Quest and HoloLens 2 support WebXR [33], a web-based immersive environment.

B. GPU Performance Counter

To assist developers in tracking and optimizing the performance of their apps, XR frameworks provide GPU performance counters. Metrics provided by the GPU performance counter reflect the graphical complexity of objects rendered on the display. Similarly, to analyze GPU usage, development frameworks offer tools to access real-time metrics and GPU profiling data provided for Meta Quest headsets. For example, Meta's `ovrgpuprofiler` tool exposes 72 GPU metrics (78 for Quest 3S) at 1 Hz sampling resolution. Apple Vision Pro's Xcode Instruments [8] offer over 150 GPU counters at up to 60Hz, and Microsoft HoloLens supplies memory and GPU utilization metrics via Windows Performance Recorder [57] and PIX [55] at 10Hz, accessible in the background.

Because maintaining high frame rates and low latency is critical to user comfort and immersion in AR/VR, GPU profilers are essential to identify performance bottlenecks. This makes them impossible to remove without severely degrading the experience. Each metric provides insight into different GPU subsystems. For instance, in `ovrgpuprofiler`, GPU counter cover utilization, memory bandwidth, and geometry throughput.

Users can also install the OVR Metrics Tool [50] from the Meta Horizon Store, which runs entirely in user-space and exposes various performance counters. This app provides basic `ovrgpuprofiler` metrics, such as GPU utilization and frame timing, to users either as an in-app HUD overlay or as CSV reports, while developers can access advanced metrics [18] within their apps.

TABLE I: Comparison of related AR/VR side-channel attacks and **OVRWATCHER**. Each row presents (i) the type of side channel, (ii) the specific sensor or API used, (iii) the extracted attributes and the corresponding number of labels, (iv) the sampling resolution (Hz), (v) whether a standalone or concurrent app is required, and (vi) the tested AR/VR environment. Symbols are used to denote an AR (🥽) and a VR (🕹). The attack denoted by † is not feasible on the Meta Quest series.

Side-Channel Type	Side-Channel Primitive	Extracted Attributes (# Labels)	Resolution (Hz)	Standalone	AR/VR
Physical	Facial Vibration Monitoring Belt [97]	Gender (2), User (27), Body fat (-)	203	×	🕹
	Power Monitoring device [40]	App (10), Website (10), Audio (5)	100	✓	🕹
Motion/Gesture Sensors	Controller [95]	Keystrokes (38)	60	✓	🕹
	IMU [78]	Keystrokes (60)	72	×	🕹
	Camera [93]	Keystrokes (4)	30	✓	🕹
	IMU (Accelerometer/Gyro) [11]	Digits (10)	1000	×	🕹
System-level APIs	Performance Counters (Unity/Unreal) [98]	Gestures (5), Voice (5), Digits (10), App†(12)	60	×	🥽, 🕹
OVRWATCHER	**GPU Profiler (in-built)**	**VR Object (35), App (100), Website (100), Avatar (10)**	**1**	✓	🥽, 🕹

III. Related Work

Side-Channel Attacks on GPU. Researchers [24], [31], [61] have introduced new side-channel attacks exploiting the parallelism and resource sharing of modern GPUs in desktop and mobile environments. Naghibijouybari et al. [61] exploited GPU performance counters for website fingerprinting, keystroke detection, and neural network recovery. These attacks, however, can be mitigated by lowering the sampling rate and reducing website fingerprinting accuracy to less than 40% at 2 samples per second. Dutta et al. [24] extended side-channel attacks to multi-GPUs using a Prime+Probe method targeting L2 cache contention to infer sensitive data transfer patterns in multi-GPU interconnects. These cross-GPU attacks can be mitigated by disabling peer memory access between untrusted GPUs and enforcing cache and memory partitioning across GPUs.

Privacy Leakage Attacks in XR. Prior works [27], [41], [62], [77], [78], [98] have demonstrated various privacy leakage attacks in XR environments. Zhang et al. [98] leveraged leakage vectors, including memory allocation APIs and performance counters from Unity [85] and Unreal [29], to recover hand gestures, voice commands, virtual keyboard keystrokes, perform application fingerprinting, and estimate bystander distance. Slocum et al. [78] proposed a system to infer words or characters typed by a victim on an XR device using a concurrent application by extracting IMU motion signals. Ling et al. [41] used computer vision and motion sensors to infer keystrokes in virtual environments, while Shi et al. [77] exploited AR/VR motion sensors to infer sensitive information from facial dynamics associated with speech (e.g., identity, gender).

Beyond keystroke recovery and app fingerprinting, XR also leaks higher-level attributes such as location and identity. Farrukh et al. [27] presented a location inference attack for MR devices, which employs geometric and semantic features from 3D spatial maps. In an analysis of a publicly available dataset, Nair et al. [62] found that hand and head motion data captured in a VR environment can uniquely identify a large number of users. Tricomi et al. [88] developed a generic profiling framework leveraging machine learning on behavioral data, including head, controller, and eye movements, to identify users and infer attributes such as age and gender. Unlike previous works, our approach uses GPU metrics without relying on motion sensors or camera data to analyze user activity in XR.

Table I compares our attack with existing side-channel attack vectors and their target environments. While prior work relies on high-resolution profiling for accuracy, our results show that even 1Hz GPU sampling can effectively compromise user privacy. We evaluate our attack in AR and VR environments, demonstrating its effectiveness in both these settings.

IV. Motivation

Our goal is to demonstrate that sensitive user information in XR environments, such as active apps, virtual object properties, and the number of participants in private meetings, can be exploited by analyzing GPU metrics collected from the built-in GPU profiler. The extracted sensitive data could enable an adversary to launch various attacks, including targeted advertising or the leakage of personal and organizational information [9], [38], [39].

For example, in a virtual shopping app, an adversary could correlate the GPU usage metrics patterns with the rendering of complex virtual product assets, such as furniture items, generating fingerprints for each item. These fingerprints can be exploited to target users with recommendations or malicious advertisements based on their preferences. Similarly, a malicious background script could track periodic fluctuations in GPU metrics corresponding to the rendering of participant avatars in a virtual meeting environment. This allows inference of the number of participants and their spatial arrangement, which could be exploited by an adversary to craft tailored phishing messages based on the inferred context of the meeting.

Prior AR/VR side-channel attacks have used high-frequency performance counters (e.g., from Unity or Unreal SDKs) to infer common gestures, voice commands, numeric inputs, and application usage with up to 95% accuracy [98]. However, their effectiveness drops below 40% at sampling rates under 10 Hz, and they rely on a malicious app running concurrently with the target app, a model infeasible on Meta Quest devices due to strict runtime isolation.

Beyond XR platforms, similar high-resolution side-channel attacks have been demonstrated on desktops and smartphones, such as website fingerprinting via GPU utilization [61], achieving around 90% accuracy with high-resolution timers and

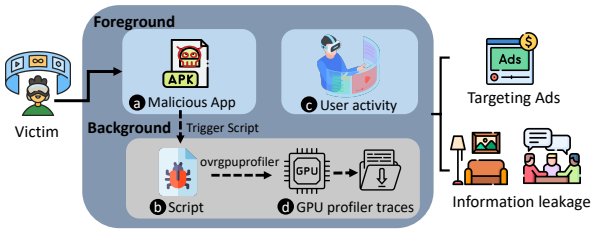


Fig. 1: Illustration of OVRWATCHER's threat model.

GPU rasterization. However, their success rate drops to around 59% when the performance counter resolution is reduced or rasterization is disabled.

Therefore, in this work, we aim to explore GPU profilers on AR/VR devices, operating at a low sampling rate of 1Hz without requiring additional SDKs, elevated privileges, or concurrent background app execution. We further aim to investigate the relationship between GPU metrics and user activity in AR/VR environments. To assess the extent to which such activity can be recognized, we aim to evaluate the efficacy of the attack through lab-controlled studies in both AR and VR settings. Finally, we aim to examine the practicality of the attack through a user study in which participants interact with a real-world application.

Design Challenges. Achieving high accuracy in predicting user activity at low sampling rates, however, entails overcoming several technical challenges, as described below.

(C1) Disabled Concurrent Applications. Prior work [98] shows that malicious apps can run concurrently with benign apps, which enables profiler-based side-channel data collection. However, unlike Microsoft HoloLens and Apple Vision Pro, Meta Quest devices restrict concurrent app execution, permitting only selected Meta apps like Messenger or Meta Chat. This limitation in Meta Quest devices makes prior attacks infeasible for third-party applications and presents a particularly challenging environment for side-channel attacks.

(C2) Efficient GPU Metric Selection. Accessing the full spectrum of GPU performance metrics (72 available GPU metrics on Quest2 and 78 on Quest3) using the built-in GPU profiler introduces significant overhead due to high resource consumption. This often results in missing or inconsistent metric values. Consequently, Meta's official documentation [47] recommends not to request more than 30 real-time metrics simultaneously. This limitation poses a challenge for reliably profiling the GPU metrics required for the side-channel attacks.

(C3) Low-Resolution Attack Channel. Prior GPU side-channel attacks rely on high-frequency sampling to achieve high accuracy [61], [98]. In contrast, Meta Quest devices expose GPU metrics at a low sampling rate of 1Hz, making fine-grained activity inference more challenging.

V. THREAT MODEL

We consider an attacker whose goal is to infer a user's privacy-sensitive activities by analyzing GPU performance metrics on an AR/VR device, as shown in Figure 1. To achieve

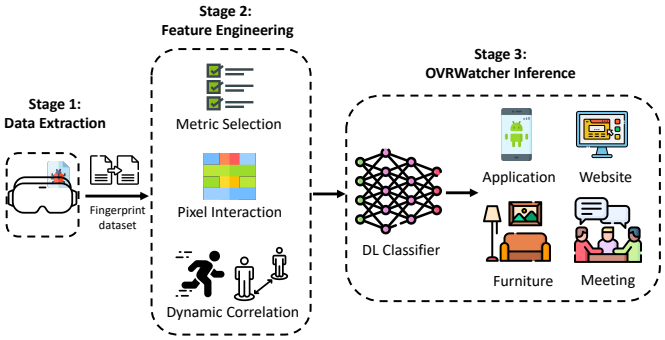


Fig. 2: Architecture of OVRWATCHER.

this, we assume that a user installs a standalone AR/VR app, such as games or productivity tools, provided by the adversary (a). While running in the foreground, this app initiates the device's GPU profiler tool to monitor and record GPU usage patterns (e.g., frame timing, shading statistics) as the user engages in user activity with other benign AR/VR apps (b, c). We load the adversary app in developer mode, which allows installation of APKs and invokes shell commands via Android's API to execute a script file. Similarly, cross-compiling and bundling the `ovrgpuprofiler` *Executable and Linkable Format* (ELF) binary with the Android NDK [15], the user space app can launch the profiler directly at runtime [4], [73], which makes it accessible without root or special privileges. We note that AR/VR app stores (e.g., Meta Store) allow apps with a GPU profiler for legitimate performance analysis [50] and, once the adversary app is activated, the profiler operates in the background even after the app is terminated (See Section VI-C).

The captured GPU metrics are then stored within the app or transmitted to a server. Both methods incur minimal energy and computational overhead, which allows the attack to remain undetected by the user and the OS (See Section VI-B). By analyzing the resulting GPU profiler traces, an adversary infers sensitive information about the user's activities (d). This information includes the apps a user is using, e.g., gaming, chat, or shopping apps, and interactions with 3D objects, e.g., the products that the user is browsing in a shopping app, as well as in-app contextual information, e.g., the number of participants in a meeting (See Section VII).

VI. OVRWATCHER DESIGN

We present OVRWATCHER, a side-channel attack that infers a user's augmented and virtual reality environment by leveraging the built-in GPU performance monitoring tool. The tool not only allows OVRWATCHER to monitor rendered content and user interactions but also circumvents the need for elevated privileges or direct sensor access from XR devices.

A. System Overview

Figure 2 illustrates three main stages in the OVRWATCHER attack. First, OVRWATCHER creates a malicious script file that runs in the background even though the malicious app is

TABLE II: A total of 30 selected GPU performance counters leveraged by OVRWATCHER.

Category	Metric
GPU Utilization	GPU Frequency, GPU Bus Busy, Preemptions / second, Avg Preemption Delay
Stalls	Vertex Fetch Stall, Texture Fetch Stall, Texture L2 Miss, Stalled on System Memory
Memory Access	Vertex Memory Read (Bytes/Second), SP Memory Read (Bytes/Second), Global Memory Load Instructions, Global Buffer Data Read Request BW (Bytes/sec), Global Buffer Data Read BW (Bytes/sec), Global Image Uncompressed Data Read BW (Bytes/sec), Bytes Data Write Requested, Bytes Data Actually Written
Shader/Instruction	Vertex Instructions / Second, Local Memory Store Instructions, Avg Load-Store Instructions Per Cycle, Avg Bytes / Fragment, L1 Texture Cache Miss Per Pixel
Geometry/Rasterization	Pre-clipped Polygons/Second, Prims Trivially Rejected, Prims Clipped, Average Vertices/Polygon, Average Polygon Area
Texture/Filtering	Nearest Filtered, Anisotropic Filtered, Non-Base Level Textures

terminated. The script file collects GPU metrics (i.e., frame timing, shading performance, GPU memory usage and texture details) for a certain period of time to capture the user activity fingerprint dataset, which overcomes **C1** by bypassing Meta Quest's restriction on side-by-side VR app execution.

In the second stage, OVRWATCHER performs reverse engineering on GPU metrics using the collected fingerprints to analyze how they manifest in fully immersive (VR) and pass-through (AR) environments. For example, an increase in GPU memory usage or texture usage obtained from GPU metrics can indicate that a new virtual object has been rendered in the immersive scene. We address **C2** by selecting only the most informative GPU metrics that capture real-time user interactions. This reduces the profiling overhead of accessing all available metrics, which would otherwise stall the GPU through frequent counter reads, and also mitigates potential data loss.

The last stage illustrates how OVRWATCHER applies machine learning classifiers such as Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), Random Forest (RF), and Support Vector Machine (SVM) models to the 1-second sampled metric sequences. By learning patterns in low-resolution data, it overcomes **C3** and accurately infers user activities such as app usage, virtual object interaction, and virtual meeting inference.

B. GPU Metrics for AR/VR Scene Analysis

We explore the relationship between GPU metrics and user activity in AR/VR environments. First, we identify GPU metrics that can be exploited to infer such activity. Then, the selected metrics are analyzed to determine their behavior based on different pixel rendering in AR/VR scenes. This analysis process lays the groundwork for understanding how low-level GPU behavior can reveal sensitive activity in AR/VR apps.

GPU Metric Selection. Simultaneous monitoring of all 72 metrics using the built-in `ovrgpuprofiler` tool generates substantial computational overhead, often straining the GPU profiling pipeline. As a result, metric data may be missing or inconsistent due to the excessive number of data buffer access requests. We use a controlled baseline of rendering three basic 3D objects (cube, cylinder, and sphere) in Unity [85] and monitor each metric during the rendering process. This approach enables us to systematically evaluate each metric and its effectiveness in accurately classifying rendered objects.

Specifically, each 3D object is set to Unity's default size of 1 $unit$[1], placed on the screen for 5 secs. We repeat the process 20 times per object, creating a comprehensive dataset covering all 72 metrics across 20 measurements for each of the 3D objects. We then train a Convolutional Neural Network (CNN) model to distinguish which virtual object is being rendered.

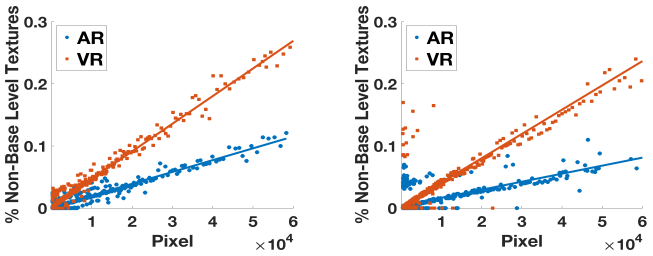
Based on our preliminary analysis, we identified 30 individual metrics that, consistently achieved over 60% classification accuracy and exhibited variation across different rendering objects within the AR/VR scene. These 30 selected metrics, shown in Table II, reduce computational overhead and ensure capturing fingerprints relevant to user interactions with higher accuracy. Specifically, limiting profiling to this subset prevents missing or inconsistent values as Meta recommends fewer than 30 simultaneous real-time counters [47]. Consequently, we further refine these metrics for each case study (Section VII).

Beyond selecting metrics from the accuracy, we perform a pairwise correlation analysis on GPU metric traces from a basic 3D cube. We compute Pearson correlation coefficients [10] over each metric and measure highly redundant pairs ($|r| > 0.90$). This process reduces the metric set from 30 to 11 while preserving the most informative traces (Appendix Table VII).

Non-Base Level Textures Metric. We systematically compared all 72 metrics from `ovrgpuprofiler` under various rendering cases. Our experiment involves rendering basic 3D objects such as a cube and observing how each metric responds to changes in object size and position from the user's point of view. From this evaluation, we found that `Non-Base Level Textures` metric achieves the highest correlation among 72 metrics provided by the `ovrgpuprofiler`.

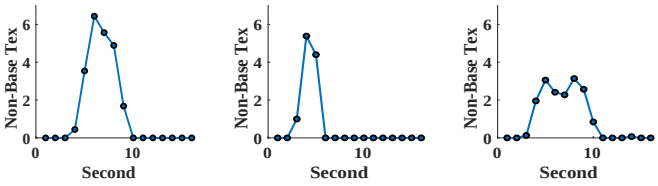
The `Non-Base Level Textures` metric represents the percentage of textures that are not at the base mipmap [94] level (level 0), which is simply the original, full-resolution texture image that the application uploads into GPU memory. The base mipmap level corresponds to the highest resolution texture, delivering maximum detail when a 3D object is viewed up close. For example, when a user interacts with large 3D objects that occupy a significant portion of the screen in AR/VR devices, the rendering process prioritizes visual clarity by using lower mipmap levels (closer to the base level) to ensure the textures of the objects appear sharp and detailed.

[1]Unity's $unit$ [90] corresponds to one meter in real-world space as a basic measurement in the scene



(a) Meta Quest 2 (b) Meta Quest 3

Fig. 3: Relation between pixel value and the `Non-Base Level Textures` metric in both (a) Meta Quest 2 and (b) Meta Quest 3.



(a) $v = 1, z = 2$ (b) $v = 2, z = 2$ (c) $v = 1, z = 3$

Fig. 4: Correlation between `Non-Base Level Textures` metric and VR Cube object rendering with (a) the speed, v = 1 $unit/second$ and distance coordinate z = 2 $unit$, (b) increasing speed to v = 2 $unit/second$, and (c) move object further from the point of view, $z = 3$ $unit$.

As a result, a higher percentage of textures near the base mipmap level leads to an increase in the `Non-Base Level Textures` metric value, which indicates that the GPU is processing more detailed textures to render the 3D object accurately. Conversely, smaller objects occupying fewer pixels on the screen require less texture detail, utilizing higher mipmap levels (further from the base level). Hence, there is a positive correlation between object size and the `Non-Base Level Textures` metric for 3D objects.

Correlating Pixel and GPU Metric. In AR/VR devices, virtual 3D objects are rendered as pixels within the immersive environment. Therefore, we posit that pixel-level analysis is a critical component for understanding user interactions with 3D content. We further characterize the exact number of pixels rendering on the screen with the selected `Non-Base Level Textures` metric. This process assigns the target object to a specific layer to isolate it from the black background, $R, G, B = (0, 0, 0)$. Next, we apply the `RenderTexture` library to convert the 3D images into `Texture2D` objects since it allows us to obtain the number of pixels for each object. We then apply a color threshold technique, which filters out the black background and retains only the pixels belonging to the target object, resulting in an accurate measure of the object's rendered pixels.

We collect 1,000 GPU metric readings by varying the size of the 3D object to observe a wide range of pixel coverage. To achieve the correlation between pixels and the `Non-Base Level Textures` metric, we employ `LinearRegression` from the Python `scikit-learn` library [67] and separately compute Pearson correlation coefficient, $\rho_{X,Y} = cov(X, Y)/\sigma_X\sigma_Y$, where $cov(X, Y) = \mathbb{E}[(X-\mu_X)(Y-\mu_Y)]$. We employ linear regression to quantify how well our chosen metric predicts pixel coverage and to validate that the selected GPU metric remains robust across both AR and VR scene configurations.

Our analysis yields a strong correlation in both AR and VR scenes. We evaluate our method on both Meta Quest 2 and Meta Quest 3 to demonstrate it applies to different hardware generations (Section VI-C). Specifically, Meta Quest 3 incorporates an upgraded system-on-chip (SoC) and improved display hardware compared to Meta Quest 2, potentially influencing GPU performance metrics. In AR mode, Meta Quest 2 achieves an R^2 score of 0.90 with a correlation coefficient of 0.95, while Meta Quest 3 achieves an R^2 score of 0.71 with a correlation coefficient of 0.84. Similarly, in VR mode, Meta Quest 2 achieves an R^2 score of 0.98 and a correlation coefficient of 0.99, whereas Meta Quest 3 records an R^2 score of 0.89 with a correlation coefficient of 0.94.

The higher VR correlations came from the fully rendered virtual environment, where every pixel change reflects GPU work. However, AR passthrough mode mixes camera feed and overlays, introducing additional noise into the metric. These values indicate a strong relationship between pixel coverage and the `Non-Base Level Textures` metric in both AR and VR scenes, as illustrated in Figure 3.

GPU Metrics in Relation to Speed and Depth. In this experiment, we automatically launch the basic Unity application, generating a single cube moving at two different speeds (1 and 2 $unit/second$). The cube travels left to right over a total horizontal distance of 30 $units$ (x-coordinates from -15 to 15). We also vary the cube's depth from the camera by placing it at three distinct z-coordinates (2, 2.5, and 3 $unit$). Each experiment runs for 30 seconds and is repeated 20 times to ensure sufficient data capture. This setup allows us to measure how moving speed and object depth affect the `Non-Base Level Textures` metric. More metrics relation is depicted in Appendix Figure 11.

When the cube moves slowly across the scene, the GPU's workload remains high for a longer duration, resulting in a broader width in the fingerprint created from `Non-Base Level Textures`. Conversely, when any 3D object moves faster in the scene, the GPU metric value spikes appear narrower. For instance, at a speed of 1 $unit/second$, a broader peak width is observed, whereas at 2 $unit/second$, the peak appears narrower, as illustrated in Figure 4a and 4b. However, due to the low resolution of the GPU profiler (1Hz), our attack only detects objects that traverse the scene in more than one second.

When the 3D object is rendered closer to the camera and occupies a substantial portion of the screen, it causes an increase in the `Non-Base Level Textures` metric value, along with other `ovrgpuprofiler` metrics that are affected by

Listing 1: Cross-compile to create binary.

```
export NDK=<Path to your NDK>/Android_ndk
export TOOLCHAIN=$NDK/toolchains/llvm/prebuilt/darwin-
    x86_64  # MacOS

$TOOLCHAIN/bin/aarch64-linux-android21-clang++ \
    -std=c++11 -O2 -fPIE -pie \
    -I"$PWD/include" \
    -L"$PWD/libs/arm64-v8a" -lOVRMetricsTool -pthread \
    -o ovrgpuprofiler \
    my_profiler.cpp             # Wrapper code
```

Listing 2: In-app profiler example.

```
# Copy and prepare the profiler ELF
dst = getFilesDir() + "/ovrgpuprofiler"
copyFile(src, dst)
setExecutable(dst)

# Option A: via Java ProcessBuilder
args = [ dst, "-r", <counters> ]
ProcessBuilder(args).start()

# Option B: Unity JNI helper double-fork
nativeStartDetached(dst, ["-r", <counters>])

function nativeStartDetached(path, args):
    pid1 = fork()                # first fork
    if pid1 > 0: return          # parent returns
    setsid()                     # child detach
    pid2 = fork()                # second fork
    if pid2 > 0: exit(0)         # first child exits
    execv(path, [path] + args)   # Execute in grandchild
```

the depth between the viewpoint and 3D objects. This visibly higher peak within the sampled dataset indicates that the GPU actively renders more detailed textures in both AR/VR scenes. In contrast, if a 3D object is farther away and occupies fewer screen pixels due to the smaller sizes, it creates minor changes in the GPU metrics. For example, placing the cube object at coordinate $z = 2$ significantly increases `Non-Base Level Textures` values, resulting in higher peaks, whereas $z = 3$ produces smaller values, as shown in Figure 4a and 4c. Consequently, these observations serve as a foundation for the detailed case studies discussed in Section VII.

C. Experiment Setup

Software Configuration. We launch our experiments on the most recently updated software of Meta Quest devices, specifically, Meta Quest builds version 72.0, Android OS version 12 [32], and Unity [34] version 2022.3.34f1.

Hardware Configuration. Meta Quest 2 is equipped with a Qualcomm Snapdragon XR2 system on a chip (SoC). This chip is an advanced product of the Snapdragon 865 model, specifically designed for AR/VR devices. The SoC integrates the Adreno 650 GPU operating at 587 MHz. Moreover, Meta Quest 3 utilizes the Snapdragon XR2 Gen 2, an enhanced version with Adreno 740 running at 599 MHz. Meta Quest 3S also employs a Qualcomm Snapdragon XR2 Gen 2 processor with 8GB of RAM. Our experiments are conducted on different dates and at multiple locations by different authors to validate cross-device and cross-environment. Furthermore, our experiments are conducted across all available Meta Quest series, which demonstrates the feasibility of our attack.

Attack Workflow. We explain the detailed step-by-step process of OVRWATCHER on the victim's device. As an in-app attack workflow, the adversary downloads the OVRMetric Tool packages [19]. Then uses the Android NDK to cross-compile a `ovrgpuprofiler` by linking `libOVRMetricTool.so` [73] with wrapper code, as shown in Listing 1.

When the app launches, it copies the binary into the private data directory and then starts `ovrgpuprofiler` by either invoking `ProcessBuilder` [65] in Java or calling a Unity JNI helper [86] that performs a double-fork method [66] as depicted in Listing 2. This double-fork sequence (line 13-19) creates a detached background process. First, fork creates a child process and `setsid()` makes that child run its own session, separating it from the app. The second fork creates a grandchild while the first child exits immediately. Because the first child exists, the grandchild process is left without a parent and automatically adopted by Android's main init process (PID1). Therefore, the profiler is no longer tied to the app's process and will continue writing GPU counter metrics to a file even within the sandbox environment.

This file can be later analyzed or transmitted when the malicious app is relaunched by the user. Our attack does not need any privilege escalation or access to sensitive sensors, as the GPU profiler is directly accessible from the XR devices.

Classification. OVRWATCHER tracks the pre-selected 30 GPU metrics from the built-in GPU profiler. The real-time metric values, M_i are collected for n seconds, where i is the metric index. The collected values generate an individual fingerprint, where each metric is represented as $M_i = \{t_1, t_2, \ldots, t_n\}$. The fingerprint uniquely characterizes the behavior of specific AR/VR apps and user interactions over time. Then, the fingerprint is preprocessed by normalizing the metrics using standard normalization to remove the baseline and noise.

We evaluate the collected dataset using four classification models: Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), Random Forest (RF), and Support Vector Machine (SVM). For CNN and LSTM models, we input the entire time series by padding multiple metrics for each execution to ensure a total size of $n \times i$ across all samples. In contrast, RF and SVM models require a fixed-length feature vector. Therefore, we compute statistical values (μ, σ, max, min) for each metric across all time steps, then concatenate these values into a single feature vector.

The preprocessed fingerprint is fed into pre-trained ML models to classify the standalone AR/VR or WebXR app currently running in the foreground. Once the app is identified, the collected GPU metric values are leveraged to infer the VR object render in both AR/VR scenes. To ensure the generalizability of OVRWATCHER, we conduct experiments across a diverse set of devices within the Meta Quest family, including cross-series models such as Quest 2, 3, and 3S.

VII. Evaluation Through Case Studies

Through four case studies, we evaluate the practical implications of OVRWatcher framework on Meta Quest devices, which prohibit concurrent standalone app execution (as detailed in Section IV, **C1**), a challenge largely unexplored in prior AR/VR side-channel research. Each case study compromises a distinct privacy and security aspect of AR/VR interaction. Collectively, they address and overcome the primary design challenges (**C1**-**C3**) posed by our threat model. In the following sections, we detail each case study. OVRWatcher proceeds in two stages: first, the attacker determines which (i) AR/VR standalone app or (ii) WebXR website has been launched; and second, it targets user privacy within these environments to extract sensitive information, including (iii) rendered VR objects and (iv) meeting room participants.

A. Case Study I: AR/VR Standalone App Fingerprint

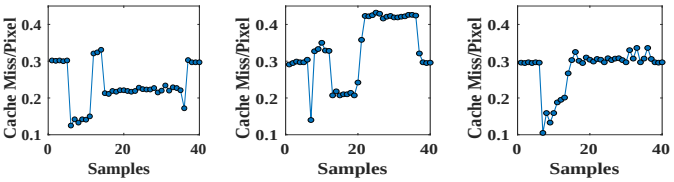
The first stage of our attack performs AR/VR application fingerprinting as the victim launches a standalone application within the immersive environment of the Meta Quest series, without requiring any user interaction.

Experiment Design. We target the top 100 most popular free applications from Meta Quest App Store[2], which consist of six different categories: Gaming (31%), Entertainment (21%), Fitness (10%), Social (7%), Productivity (27%), and Mixed Reality (4%). Although these categories encompass a diverse set of applications, gaming remains the most popular use case for AR/VR headsets [45], where it dominates a large portion of the App Store. We list those 100 apps with their main purpose categories in Appendix Table XVI.

Data Collection. After selecting the target applications, we employ a malicious script file to run the OVRWatcher framework. Specifically, we automatically launch the target applications to collect the app fingerprinting dataset. For each measurement, we run an app for 30 seconds and simultaneously monitor the 30 selected GPU metrics as outlined in Section VI-B. Each app is profiled 20 times, resulting in a dataset consisting of 2000 measurements across the Meta Quest family (Quest 2 and 3). For each device, we randomly split the dataset into balanced subsets using 80% for training and 20% for testing.

Results. We observe that each AR/VR app exhibits a unique fingerprint due to its immersive graphics and resource-intensive design. Such high rendering complexity leads to distinct GPU usage patterns. Examples of these fingerprints are shown in Figure 5. In particular, before any application is launched, there is a constant baseline of GPU usage attributed to the system due to the background load, such as AR (passthrough) view or VR (immersive) view. Once an AR/VR app starts, GPU usage rises sharply due to the demands of rendering immersive 3D graphics. During the app's runtime, GPU usage remains elevated to handle real-time interactions and dynamic scene updates. When the user closes the application, GPU usage returns to its baseline level associated with either the AR view or the VR view.

[2]https://www.meta.com/experiences/view/1321443348416166



(a) Gorilla Tag (b) Bigscreen Beta (c) Multiverse

Fig. 5: Example of fingerprints collected from Meta Quest 2 for AR/VR standalone apps based on the `L1 Texture Cache Miss Per Pixel` metric: (a) Gorilla Tag, (b) Bigscreen Beta, and (c) Multiverse.

TABLE III: AR/VR standalone app fingerprinting performance on 100 standalone apps on Quest 2, using a combination of 30 GPU metrics (average scores).

Model	F1 (%)	Precision (%)	Recall (%)	Accuracy (%)
CNN	98.3	98.5	98.4	99.3
LSTM	98.7	98.9	98.9	98.6
RF	99.4	99.5	99.6	99.5
SVM	85.1	86.9	87.6	86.8

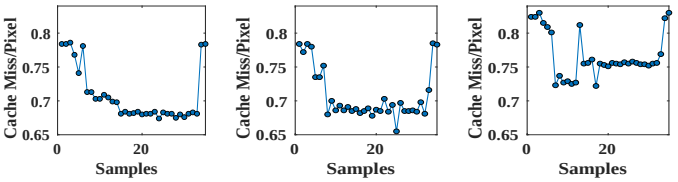
As shown in Table III, CNN, LSTM, and RF models achieve accuracies of 99.3%, 98.6%, and 99.5% , respectively, in classifying 100 different AR/VR standalone apps while utilizing a selected combination of 30 metrics. Specifically, test accuracy shows the overall percentage of correctly classified samples across all class labels. For each class label, we compute the F-1 score, precision, and recall metrics individually, and we report the average values. Additionally, we evaluated the classification performance using each GPU metric individually. Notably, each 10 metrics achieves more than 90% F-1 score (Appendix Table VIII) using the RF model, indicating that even a single metric from the `ovrgpuprofiler` tool is feasible to perform an AR/VR app fingerprinting attack.

Overall, OVRWatcher successfully identifies the foreground app with 99.5% accuracy using 30 metrics on Quest 2 and maintains over 90% accuracy even with a single metric. Furthermore, we achieve similar 95.8% classification accuracy throughout the cross-device setting (Appendix Table XV). This indicates that the pre-trained model is feasible to identify user rendering AR/VR apps across different device setups.

B. Case Study II: WebXR App Fingerprint

WebXR API enables users to explore and engage with virtual environments directly through the browser of any HMD. Users can access a WebXR app in a manner similar to accessing a standard website through a browser. Before entering the 3D mode, the app loads within the 2D browser screen displayed on the HMD. By default, WebXR enables the rendering of WebGL scenes, allowing websites and browsers to utilize the GPU for rendering. This capability makes our attack feasible for fingerprinting WebXR apps.

Experiment Design. We selected 100 popular and free WebXR apps based on recommendations from social media lists [69],



(a) VARTISTE (b) Moon Rider (c) Towermax Fitness

Fig. 6: Example fingerprints for WebXR apps based on the `L1 Texture Cache Miss Per Pixel` metric. (a) Vartiste, (b) Moon Rider, and (c) Towermax Fitness.

[79], [83], [84] and example showcases from development platforms such as A-Frame [12], WebXR [33], as well as free apps hosted on Glitch [82]. We selected applications from five distinct categories: Gaming & Entertainment (19%), Art & Creativity (11%), Tours & Exploration (18%), Health & Fitness (8%), and Demonstration (44%) as detailed in Appendix Table XVII. Although most of the applications are showcase examples from standard WebXR sites (Demonstration category (44%)), they effectively demonstrate the diverse capabilities offered by WebXR. These capabilities include anchors [37], positional audio [36], animation [1], as well as minimal demonstrations such as shopping [3] and reading [2].

Data Collection. After selecting the target applications, we execute the OVRWATCHER framework. Similar to the standalone AR/VR apps, we monitor the metrics provided by `ovrgpuprofiler` for 30 seconds. We collect 30 metrics out of the 72 available GPU metrics using a similar methodology, as detailed in Section VI-B, repeating the process 20 times for each selected WebXR app.

TABLE IV: WebXR app fingerprinting performance on 100 standalone apps on Quest 2, using a combination of 30 GPU metrics (average scores).

Model	F1 (%)	Precision (%)	Recall (%)	Accuracy (%)
CNN	97.5	98.3	97.6	97.6
LSTM	96.1	96.7	96.4	96.4
RF	99.2	99.4	99.0	99.0
SVM	72.8	74.8	74.5	74.5

Results. We observe distinct fingerprints for each app based on the selected 30 metrics. When a WebXR app is launched, GPU usage spikes sharply from its baseline and returns to normal upon termination. For example, based on `L1 Texture Cache Miss Per Pixel` metric, Figure 6 shows distinct fingerprints for three applications, VARTISTE [91], Moon Rider [60], and Towermax Fitness [28], which represent the categories Gaming & Entertainment, Art & Creativity, and Health & Fitness, respectively.

As shown in Table IV, the RF and CNN models achieves accuracies of 99.0% and 97.6%, respectively, in classifying 100 WebXR applications using 30 metrics, while the LSTM model achieves 96.4%. Moreover, the top four metrics listed in Appendix Table IX individually achieve over 72% accuracy in classifying 100 WebXR apps using the RF model. Additionally, OVRWATCHER achieves over 94% accuracy in the RF model using only the top two metrics and 98.7% accuracy combining the top four metrics, demonstrating the effectiveness of the attack with a reduced set of metrics. Furthermore, across Quest 2, 3 and 3S, the RF model maintains an accuracy of 97.15% (Appendix Table XV) demonstrating cross-device robustness.

Unlike the fingerprinting of standalone AR/VR applications rendered in 3D mode, this case study demonstrates that even rendering a WebXR application within a 2D browser interface is sufficient for OVRWATCHER to achieve high classification accuracy, despite operating at a significantly lower sampling resolution compared to prior GPU profiler-based website fingerprinting attacks [61].

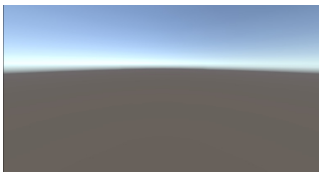
C. Case Study III: Virtual Object Detection

After OVRWATCHER infers the standalone AR/VR or WebXR app, it can further compromise the user's privacy by identifying the rendered objects within the immersive environment. For example, applications like IKEA Place [35] enable users to virtually place furniture, allowing them to visualize how different pieces would fit into their existing environment using mobile devices. Similarly, immersive virtual showrooms developed by Demodern enable users to explore and arrange furniture in simulated environments, providing a realistic and interactive platform for home design [14], [21].

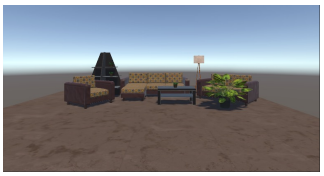
Inferring details about a user's home layout and the types of products they explore within a shopping app can be exploited for targeted advertising, unauthorized surveillance, and more advanced privacy attacks. In this case study, we evaluate the ability of the OVRWATCHER framework to accurately detect and identify virtual product placement within both AR and VR scenes. Specifically, we focus on realistic product prefabs designed to resemble IKEA-style furniture, which are commonly used in XR environments to help customers virtually visualize and place products within their homes or offices.

Experiment Design. We evaluate the object detection attack in both AR and VR environments. We selected product prefabs from five free asset packages available on Meta Asset Store [46], including Toon Furniture [26], Free Furniture Set [20], HDRP Furniture Pack [89], Apartment Kit [80], and Chair and Sofa Set [30]. From these assets, we chose a representative subset of 35 items, spanning large pieces (sofas, beds, desks) to smaller home accessories (lamps, coffee machines, toasters) to ensure coverage of diverse shapes, sizes and usage contexts in a shopping scenario. We also select multiple items from the same furniture category. This 3D object selection will demonstrate OVRWATCHER's ability to distinguish between visually analogous 3D objects.

We design four experimental scenes to simulate real-world usage scenarios: (i) an empty VR environment as a baseline, (ii) a VR living room setup mimicking a typical living room layout with multiple VR objects, (iii) an AR living room where actual objects are placed within the physical living room environment, and (IV) an AR office environment with an actual office space



(a) Default VR Scene



(b) VR Living Room Scene

(c) AR Office Scene

(d) AR Living Room Scene

Fig. 7: Experiment scenes for furniture classification. Two VR scene setups (a) without VR objects and (b) with VR objects (living room layout). Two AR scene setups in an (c) office and (d) living room.

and real furniture such as desks, chairs, and monitors. Figure 7 illustrates these four experimental setups.

Data Collection. We simulate typical user interactions with a virtual product, such as moving items to explore optimal placements or moving the viewpoint (headsets) to better visualize the overall arrangement. In each AR/VR scene, the selected prefab is rendered with speeds of $v = 1,\ 10,\ 50$ $unit/second$ from left to right. Furthermore, we vary the distance values between the user's origin and the virtual product position. Specifically, the selected object distance coordinates are set at $z = 0,\ 5,\ 10$ $unit$, with the camera positioned at $z = -3$ $unit$ relative to the user's default origin. The variations in distance and movement speed mimic realistic user interaction with virtual objects and exploration of product placement. We collect 20 iterations for each combination of speed and distance values with 35 prefabs.

To automate this process, we developed a Unity code that reads a configuration file, which is saved as a text file in the application directory. This text file specifies the prefab name, position, movement speed, and distance for each iteration. A bash script then saves this configuration into the app's working folder and invokes `ovrgpuprofiler` to start GPU profiling, while the Unity code reads the configuration text file and moves the object. For each iteration, this unified automation workflow logs GPU metrics to a local file over a total duration of 40 seconds. Specifically, the first 10 seconds allow the AR/VR scene to fully load and stabilize to ensure consistent tracking and rendering, while the remaining 30 seconds capture the user moving the selected virtual furniture object at one of the predefined speed values to simulate realistic interaction.

Results. We deploy the same ML model types used in Case Study I and II, which are CNN, LSTM, RF with 100 estimators, and SVM, to ensure consistency in our evaluation. As shown in Table V, CNN, LSTM, and RF models achieve the highest accuracies of 96.5%, 92.6%, and 98.1%, respectively, in the default VR scene with 17 selected metrics (Appendix Table XI). Averaging across all four scenes, these three models maintain robust performance with 93.6%, 87.4% and 97.1%, respectively. Specifically, the selected 17 metrics (out of the 30 collected metrics) individually achieve more than 80% classification accuracy (CNN) applied in the default VR scene with $v = 1$ $unit/second$ speed and distance coordinate with $z = 0$. Notably, `Non-Base Level Textures` and `Global Buffer Read L2 Hit` metrics individually achieve 100% classification accuracy, which demonstrates that a single metric is feasible to classify the VR objects.

TABLE V: Fingerprinting performance of virtual products on Quest 2 using a combination of 17 GPU metrics (average scores in %).

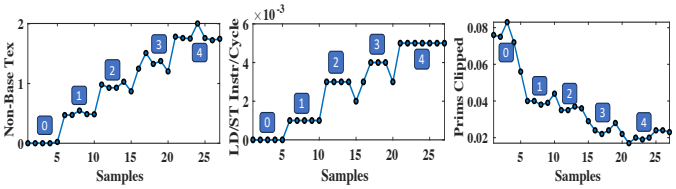
	VR Scene						AR Scene					
	Default			Living room			Office			Living room		
Model	**F1**	**Pre**	**Rec**	**F1**	**Pre**	**Rec**	**F1**	**Pre**	**Rec**	**F1**	**Pre**	**Rec**
CNN	96.5	96.6	94.3	88.3	90.6	90.2	95.1	94.6	96.7	94.4	94.6	94.5
LSTM	92.6	92.3	94.3	75.8	80.1	80.9	89.8	90.9	91.3	91.4	91.1	93.0
RF	98.1	98.2	98.2	95.6	96.4	96.2	98.1	98.1	98.2	96.6	97.0	97.0
SVM	50.6	50.7	59.6	36.7	39.6	43.3	50.6	50.7	59.6	56.0	57.6	62.6

The VR living room scene (Figure 7b) presents the most challenging environment due to the presence of default VR objects (living room furniture) within the scene. This introduces significant complexity and noise since several overlapping objects and complex textures make it more challenging to distinguish GPU activities caused by an individual furniture object. In this challenging VR living room scenario, we still maintain 87% classification accuracy with a single metric, `Prims Trivially Rejected`.

The GPU metrics show minimal impact due to the scene complexity in AR environments because the objects in the AR are not rendered through GPU components. The real-world background is directly taken from the headset's built-in camera feed (passthrough mode) rather than being fully rendered by the GPU. Hence, the GPU only handles the virtual objects overlaid on top of the camera feed. Consequently, OVRWATCHER achieves nearly 100% accuracy in both AR scenes (Figure 7c and 7d) by leveraging only the `Non-Base Level Texture` percentage metric that captures the subtle pixel changes as demonstrated in Section VI-B.

When all 17 metrics are combined to create a single fingerprint, the classification models achieve consistently high accuracy across all four experimental scenes. In particular, RF surpasses 95% accuracy even in the VR living room scene ($v = 1$, $z = 0$), while CNN and LSTM each maintain accuracies above 90% in all four scenes. However, SVM shows significantly lower accuracy, ranging from 37% to 60% in all scenes, as shown in Table V. While SVM requires extensive tuning and struggles to handle complex data effectively, the RF model leverages an ensemble of decision trees to handle multidimensional data, and CNN/LSTM models efficiently capture important features in a sequential dataset.

Beyond the accuracy-based metric pruning, we apply Pearson



(a) (b) (c)

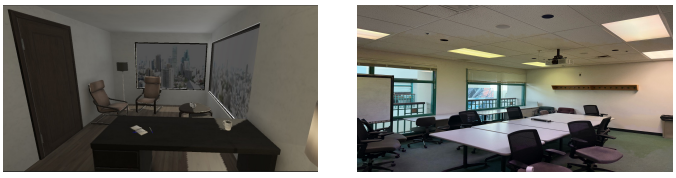
Fig. 8: Example of fingerprint dataset of meeting participant inferences in default VR scene setup (Figure 7a). (a) `Non-Base Level Texture`, (b) `Avg Load-Store Instructions Per Cycle`, and (c) `Prims Clipped` metric in AR scene. The number on the graph represents the number of participants joining the target meeting room.

correlation-driven selection (Section VI-B). We narrow 5 core metrics (Appendix XII) by taking the intersection with our selected 17 metrics in Case Study III (Appendix Table XI) and 11 metrics from correlation-driven metric selection (Appendix Table VII). We retrain our object detection models on a 5-pruned metric set and achieve 91%. We gain only a slight drop (4%) in the RF model in the VR living room scene, while CNN still maintains 90% accuracy. This additional comparison demonstrates that a minimal subset of uncorrelated, high information metrics can obtain robust performance with lower computational overhead. The cross-device setting across Quest 2, 3, and 3S also demonstrates a high accuracy of 93% (Appendix Table XV).

Our classification accuracy in the variation of speeds ($v = 1,\ 10,\ 50$) and distance coordinates ($z = 0,\ 5,\ 10$) for the default VR scene reveals that the attacker can detect virtual objects in different scenarios, as shown in the Appendix Table X. The accuracy drops by roughly 25% at the extreme speed, $v = 50$ with the same distance $z = 0$, but only 4% drop at the furthest distance $z = 10$. This is because the speed and size of the object are directly related to its distance from the origin. When the object is closer to the user (smaller z values), it occupies a larger portion of the screen, spanning a greater number of units within a specific time frame. This relationship can be expressed as $v_{screen} \propto \frac{s \times v}{z}$, where s is the size of the object. Although high speed changes may reduce distinctive GPU changes due to OVRWATCHER's low resolution, the results demonstrate that the high speed and small size virtual objects are still detectable.

D. Case Study IV: Meeting Room Inference

As XR platforms such as Spatial [81], AltspaceVR [53], Horizon Workrooms [44], and VRChat [92] become popular venues for hosting social and professional gatherings, concerns over potential leakage of private information related to these meetings become more critical. The use of realistic AR/VR meeting applications, featuring detailed 3D representations of participants, introduces significant GPU workloads. These GPU usage patterns can inadvertently expose sensitive information, creating new vectors for privacy leakages.



(a) Office VR Scene (b) Conference Room AR Scene

Fig. 9: Experiment scenes for meeting room inference. (a) Office VR scene and (b) conference room scene.

In this case study, we simulate both AR and VR meeting rooms to infer the number of participants by analyzing GPU performance metrics. OVRWATCHER demonstrates the ability to infer the number of meeting participants accurately, revealing meeting attendance without any user interaction.

Experiment Design. For this case study, we designed three different AR/VR scenes to simulate various meeting environments: (i) the default VR empty environment as shown in Figure 7a, (ii) office room-like VR scene [25] with limited space size as shown in Figure 9a and (iii) actual conference room as depicted in Figure 9b.

To simulate different numbers of participants in a virtual meeting, we use standard humanoid avatar [23] prefabs obtained from the Unity Asset Store. These avatars are positioned within each scene to ensure they remain visible from the user's point of view. In the simulated AR environment (Figure 9b), the relatively larger room accommodates up to four humanoid avatars at a distance coordinate of $z = 3$, all within the user's field of view. However, the more confined VR scene, resembling a small office room (Figure 9a), can fit only two avatars without overlap, while maintaining visibility from the user's view.

To further mimic real-world scenarios, we introduce an additional five humanoid avatars alongside the initial four, resulting in a total of nine avatars within the AR and VR meeting scene. These additional avatars create overlapping situations that commonly occur in practical AR/VR meetings, e.g., when participants move within the environment and some avatars become partially obscured by others. We selected a total of 9 humanoid avatars to simulate these overlapping situations, ensuring they are positioned to fit entirely within the AR/VR scene without exceeding the spatial constraints.

Data Collection. We use the same set of 30 metrics selected during the feature selection process (Section VI-B). During the data collection, we execute OVRWATCHER across different scene setups and render up to 9 humanoid avatars (two avatars in the VR office scene).

To simulate participants joining a virtual meeting, avatars are incrementally rendered within the scene. Each avatar remains within the user's field of view for 5 seconds before the next avatar appears in the scene. In practice, scenes with up to four avatars require a total rendering duration of approximately 30 seconds, accounting for app initialization, avatar transitions, and a final stabilization period. For scenarios involving all nine avatars, the total rendering time extends to 55 seconds to

ensure the complete sequence is captured.

We construct datasets for the two scenes, the default VR scene (Figure 7a) and the AR meeting scene (Figure 9b). Each dataset is formed of different scenarios, with each scenario representing a specific number of participants in the scene, ranging from 0 to 9. For each scenario, we collect 10 measurements, each with a duration of 70 seconds. For the VR office scene (Figure 9a), we construct the scenarios with 0 to 2 participants due to spatial constraints, and similarly, 10 measurements are collected per scenario.

Results. Analysis of the 30 GPU metrics reveals distinct changes based on the number of participants across three distinct scenes. In the AR office scene, 22 metrics show noticeable jumps from the baseline values as the number of participants increases, as illustrated in Figure 8. These jumps indicate a direct correlation between GPU workload and the number of participants. Interestingly, four specific metrics, `Prims Trivially Rejected`, `Prims Clipped`, `Average Vertices / Polygon`, and `Average Polygon Area`, show decreasing step behavior as the number of participants increases in the scene (Figure 8c).

Specifically, the `Prims Trivially Rejected` metric measures the percentage of basic shapes (primitives) the GPU can immediately ignore from the rendering pipeline if they do not contribute to rendering in the immersive scene. For example, in the empty scene, many primitives are rejected (ignored), resulting in a high baseline value. As more participants join, more primitives become relevant, so the GPU rejects fewer primitives. This causes the metric value to decrease in noticeable steps. The `Prims Clipped` indicates the percentage of primitives cut off by the camera view, `Average Vertices / Polygon` captures geometric complexity by measuring the average number of vertices per polygon, and the `Average Polygon Area` measures the screen space each shape covers. Therefore, these metrics exhibit decreasing values if more participants join the scene.

We train the same types of machine learning models to accurately determine the number of participants in VR and AR meeting scenarios and perform a 5-fold cross-validation on our dataset to ensure robust evaluation and avoid the overfitting risk. We achieve 100% accuracy in detecting the number of participants by monitoring 20 metrics individually with the RF model in the default VR scene and 18 metrics in both the VR office scene and AR meeting room setup. We identified 13 common individual metrics resulting in 100% accuracy in inferring the number of participants across all three scenes by employing the RF classifier. Furthermore, from cross-device setup, we achieve 100% accuracy as mentioned in the Appendix Table XV. These overall results demonstrate that OVRWATCHER can perfectly infer the number of participants in both complex VR and the realistic AR meeting setup.

Across all four case studies, OVRWATCHER achieves consistently high accuracy on Meta Quest 2, 3, and 3S by leveraging the GPU metrics. This demonstrates that our method bypasses Meta Quest's strict concurrent app policy (**C1**), operates effectively with a minimal set (1-3) of metrics with avoiding profiling overhead and data loss (**C2**), and extracts user activity at a low 1Hz sampling rate (**C3**).

VIII. Open World User Interaction

AR/VR applications depend heavily on natural user movements and gestures, yet our earlier case studies rely on developer-designed applications and evaluation in a controlled environment. Furthermore, AR/VR shopping and home-design apps such as IKEA Place [35] let users preview virtual furniture in their own living spaces, but they are only accessible in selected retail locations [14]. To demonstrate that OVRWATCHER remains effective when users drive the experience, we evaluate its performance in an open-world AR setting using the pre-built Meta application called Meta Layout [52].

This app enables users to browse, scale, rotate, and position realistic furniture models directly within their physical environment via AR passthrough, which mirrors the IKEA experience. We leverage the Layout app with real users as they freely place, hold, and remove virtual objects. We demonstrate that OVRWATCHER can distinguish virtual objects under real-use conditions with the 1Hz, low-resolution GPU profiling tool.

Data Collection. We obtained Institutional Review Board (IRB) approval and recruited six volunteers from our university. Participants are diverse in age, height, weight, and prior AR/VR experience, with more details provided in the Ethical Considerations section. Prior to the experiment, they received a brief training session on Quest 3S and the Meta Layout app. We design two separate scenarios for data collection.

(i) In the static scenario, participants hold a Quest controller button to generate one of the five 3D furniture items (chair, sofa, table, desk, and bed) into the user's view for 5 seconds, then make the 3D object disappear for another 5 seconds by pressing the button on the controller. This render-remove cycle takes 10 seconds and is repeated 10 times for each furniture item, a total of 100 seconds of GPU profiling traces per participant.

(ii) As users are expected to interact with virtual objects in real-world VR applications, we also designed a dynamic scenario in which the participant opens the in-scene panel in the Layout app, selects a furniture item, and drags it into their field of view for 10 seconds. After 10 seconds, the participant drags the 3D object out of their view. This scenario takes around 40 seconds and is repeated 5 times for each 3D object. Including brief rest breaks and menu navigation, the dynamic scenario requires approximately 25 minutes per participant to complete all 3D object interactions.

Results. We leverage the 17 selected metrics from Case Study III VII-C (Appendix Table XI) and employ RF, XGBoost, and SVM models to distinguish the objects. During data collection, we noticed that participants sometimes pressed or released the controller button early or late. Therefore, GPU readings didn't line up exactly with our intended 5-second cycles. To handle these misalignments in the dynamic scenario, we first count how many samples each trace actually captures per second, then extract the 10-second window as the object fingerprint. For

TABLE VI: Accuracy of RF, XGB, and SVM in the static and dynamic user-study scenarios on Meta Quest 3S.

Scenario	Model	Test (%)	5-Fold (%)
Static Interaction	RF	83	86 ± 6
	XGB	82	88 ± 6
	SVM	30	38 ± 4
Dynamic Interaction	RF	81	77 ± 5
	XGB	67	66 ± 9
	SVM	78	80 ± 10

each window, we compute the per-metric mean and standard deviation and concatenate these values into our feature vector.

In the static scenario, all the fingerprints from each user are combined to generate the dataset, where 20% of the dataset is used for the test dataset while the remaining is used for training. The RF model achieves 83% accuracy while XGBoost reaches 82%. We then train all models and perform hyperparameter tuning via grid search with 5-fold cross-validation. Under this evaluation, the RF model shows the robustness by achieving $86\% \pm 4$ and $88\% \pm 6$ accuracy for the XGB model.

In the dynamic scenario, where users incorporate additional real-time movement, including dragging, dropping, head movement, and continuous cursor movement, the overall classification accuracy slightly declines. We achieve 81% accuracy with the RF model on the same 80/20 test split. Applying the same 5-fold cross-validation approach, the RF achieves $77\% \pm 5$ as shown in Table VI. The accuracy drop from the static to dynamic scenario shows the impact of extensive user interaction, which introduces noise in each object's fingerprint.

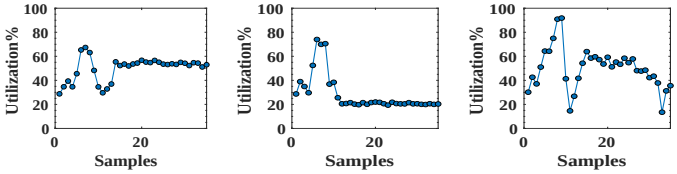
Additionally, we evaluate the accuracy for unseen users by applying the leave-one-participant-out (LOPO) cross-validation method. We achieve an accuracy of 72% using the transformer model. These results demonstrate the effectiveness of the OVRWATCHER attack in practice, which can be further improved through training on a larger and more diverse user base.

IX. Countermeasures and Limitations

A. Countermeasures

Restrict GPU Profiler Access. The `ovrgpuprofiler` tool utilizes Performance Interface Library (PIL), a low-level on-device library within the Oculus OS that exposes real-time GPU metrics. Because developers depend on this profiler for legitimate performance tuning, we cannot simply remove or disable it without breaking valid workflows. Instead, PIL calls should be restricted to apps running in developer mode and the app store's vetting process should block unapproved uses of these APIs. This aligns with previous works to prevent access to OS-level sensors such as frequency scaling [22], [43], power consumption [42], [99], and API accesses to performance counters [61], [98].

Dynamic Detection. A defense mechanism can monitor system calls using tools such as *perf trace* to detect repetitive access patterns to the GPU profiler. Upon detection, the system can inform the user with a warning on the screen. Since the profiler samples GPU metrics at a low resolution, this detection approach incurs minimal performance overhead.



(a) Algorithmic Nature (b) Graffiti 3D (c) Insight Heart

Fig. 10: Example fingerprints for standalone MR apps on HoloLens 2 based on the `GPU Utilization` metric. (a) Algorithmic Nature, (b) Graffiti 3D, and (c) Insight Heart.

Noise Injection. Introducing artificial noise into the GPU components by executing dummy instructions can disrupt fingerprinting attacks. Such a mechanism can be built by rendering random, non-intrusive objects, similar to noise injection strategies used against website fingerprinting in network traffic [75] and browsers [13], [74]. However, noise injection method introduces additional overhead, potentially slowing down object rendering. Hence, the trade-off between the attack success rate reduction and performance overhead needs to be considered for the noise injection countermeasure.

B. Limitations

Applicability to Diverse Devices. XR devices from other vendors expose different profiling interfaces. For example, Apple Vision Pro [7] supports concurrent app execution and offers extensive GPU counters via Xcode *Instruments* (over 150 metrics) [8], while Microsoft HoloLens [56] provides profiling through *Windows Performance Recorder* (WPR) [57] and PIX [55] for detailed GPU profiling.

To validate OVRWATCHER's broader applicability, we also tested it on Microsoft HoloLens 2 by running publicly available MR apps from the Microsoft Store. By collecting the device's built-in GPU metrics such as `Utilization`, `Dedicated Memory`, `System Memory`, and `System Memory Used`, we confirmed that our methodology can capture rendered app behavior as shown in the Figure 10. In future work, we aim to expand our attack framework to encompass various AR/VR devices, including the Apple Vision Pro's Xcode *Instruments* and other vendor-specific tools.

Generalizability. Our case studies (Sections VII-A and VII-B) utilize publicly available standalone AR/VR and WebXR applications, demonstrating their generalizability. In Sections VII-C and VII-D, we selected a diverse set of objects and varied parameters (speed and distance) to prove the concept of object and avatar identification. Moreover, our setup focuses on a single-app scenario. However, in real-world scenarios, users may open multiple browsers or switch between apps, which can affect attack accuracy. Additionally, rapidly moving ($< 1s$) virtual objects within the scenes might influence attack effectiveness. Future work will extend our evaluation to multi-app and highly dynamic environments.

X. Conclusion

We introduce OVRWatcher that exploits real-time GPU profiling metrics to extract sensitive user activities from AR/VR users. This finding invalidates the common defense of simply lowering profiler resolution. Across app detection, website classification, object detection, meeting participant counting, and open-world user interaction with 3D objects, OVRWatcher achieves high accuracy with a 1Hz sampling rate. Our findings underscore the need for more restrictive permission models in AR/VR operating systems to safeguard user privacy against sophisticated side-channel attacks.

Acknowledgment

This work was partially supported by the National Science Foundation (NSF) under grants CNS-2144645 and IIS-2229876. Grant 2229876 was also supported in part by funds from the Department of Homeland Security and IBM. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF or its federal and industry partners.

Ethical Considerations

We introduce OVRWatcher, a GPU-based side-channel attack on AR/VR headsets that abuses vendor-provided GPU metrics and exposes sensitive user activities, undermining both user privacy and service-provider trust. To address these ethical concerns, we responsibly disclosed our findings to Meta via their bug bounty program [51] on January 15, 2025.

In March 2025, Meta confirmed that user-space applications can indeed access basic GPU performance counters and awarded us a Bug Bounty for identifying this side-channel vulnerability. We are waiting for their response on the upcoming firmware update.

As of June 4, 2025. Meta has tightened developer access requirements for native Android AR/VR development [17]. Developer mode now requires users to join or create a verified developer organization via the Meta Horizon Developer Dashboard. While these changes may not directly prevent misuse of performance metrics, they raise the barrier for arbitrary access by requiring stronger identity validation for developer mode access.

We conducted a user study to validate the attack's effectiveness in a real-world scenario, recruiting six participants (age 18+) from our university through an IRB-approved process. To ensure participants' well-being during Mixed Reality interaction, interested individuals were asked to complete an online screening survey assessing medical conditions such as vision or hearing impairments, seizures, motion sickness, and neurological disorders. Only individuals without any of the listed conditions were invited to the in-lab study. Throughout the study, participants were monitored for safety and informed of their right to withdraw at any time. To protect participant privacy, no personally identifiable information (PII) was collected during the in-lab sessions.

References

[1] A-Frame. Animation raw example - a-frame. https://aframe.io/aframe/examples/performance/animation-raw/. Accessed: 2025-01-17.

[2] A-Frame. Comic book showcase example - a-frame. https://aframe.io/aframe/examples/showcase/comicbook/. Accessed: 2025-01-17.

[3] A-Frame. Shopping showcase example - a-frame. https://aframe.io/aframe/examples/showcase/shopping/. Accessed: 2025-01-17.

[4] Vitor Afonso, Antonio Bianchi, Yanick Fratantonio, Adam Doupé, Mario Polino, Paulo De Geus, Christopher Kruegel, Giovanni Vigna, et al. Going native: Using a large-scale analysis of android apps to create a practical native-code sandboxing policy. In *The Network and Distributed System Security Symposium (NDSS)*, 2016.

[5] 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.

[6] Apple. Apple vision pro. https://www.apple.com/apple-vision-pro/, 2024. Accessed: 2024-12-30.

[7] Apple Inc. Apple Vision Pro. https://www.apple.com/apple-vision-pro/, 2025. Accessed: 2025-01-16.

[8] Apple Inc. *Optimizing GPU Performance*, 2025. Accessed: 2025-01-16.

[9] Natã M Barbosa, Gang Wang, Blase Ur, and Yang Wang. Who am i? a design probe exploring real-time transparency about online and offline user profiling underlying targeted ads. *ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*, 2021.

[10] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*. Springer, 2009.

[11] Derin Cayir, Reham Mohamed, Riccardo Lazzeretti, Marco Angelini, Abbas Acar, Mauro Conti, Z Berkay Celik, and Selcuk Uluagac. Speak up, i'm listening: Extracting speech from zero-permission vr sensors. In *The Network and Distributed System Security Symposium (NDSS)*, 2025.

[12] A-Frame Community. A-frame examples: Interactive webxr demos and use cases. https://aframe.io/aframe/examples/, 2025. Accessed: 2025-01-14.

[13] Jack Cook, Jules Drean, Jonathan Behrens, and Mengjia Yan. There's always a bigger fish: a clarifying analysis of a machine-learning-assisted side-channel attack. In *International Symposium on Computer Architecture (ISCA)*, 2022.

[14] Demodern. Ikea vr showroom. https://demodern.com/projects/ikea-vr-showroom, 2020. Accessed: 2025-1-5.

[15] Android Developers. Android ndk. https://developer.android.com/ndk, 2025. Accessed: 2025-07-18.

[16] Meta Developers. Horizon unity documentation: Ovrgpu profiler. https://developers.meta.com/horizon/documentation/unity/ts-ovrgpuprofiler, 2024. Accessed: 2024-12-05.

[17] Meta Developers. Mobile device setup for native android development. https://developers.meta.com/horizon/documentation/native/android/mobile-device-setup/?utm_source=chatgpt.com, 2024. Accessed: 2025-07-17.

[18] Meta Developers. Monitor performance with ovr metrics tool. https://developers.meta.com/horizon/documentation/unity/ts-ovrmetricstool/#accessing-metrics-data, 2024. Accessed: 2025-07-17.

[19] Meta Developers. Ovr metrics tool package download. https://developers.meta.com/horizon/downloads/package/ovr-metrics-tool/, 2025. Accessed: 2025-07-18.

[20] Dexsoft. Free furniture set. https://assetstore.unity.com/packages/3d/props/furniture/furniture-set-free-242389, 2023. Accessed: 2025-1-5.

[21] Present Digital. Present digital ikea projects. https://present.digital/ikea/, 2021. Accessed: 2025-1-5.

[22] Debopriya Roy Dipta and Berk Gulmezoglu. Df-sca: dynamic frequency side channel attacks are practical. In *Annual Computer Security Applications Conference (ACSAC)*, 2022.

[23] doxygen. Humanoid control for unity v4. https://humanoidcontrol.com/. Accessed: 2025-1-8.

[24] Sankha Baran Dutta, Hoda Naghibijouybari, Arjun Gupta, Nael Abu-Ghazaleh, Andres Marquez, and Kevin Barker. Spy in the gpu-box: Covert and side channel attacks on multi-gpu systems. In *International Symposium on Computer Architecture (ISCA)*, 2023.

[25] Elcanetay. Office room furniture. https://assetstore.unity.com/packages/3d/props/furniture/office-room-furniture-70884. Accessed: 2025-1-8.

[26] Elcanetay. Toon furniture. https://assetstore.unity.com/packages/3d/props/furniture/toon-furniture-88740, 2023. Accessed: 2025-1-5.

[27] Habiba Farrukh, Reham Mohamed, Aniket Nare, Antonio Bianchi, and Z Berkay Celik. {LocIn}: Inferring semantic location from spatial maps in mixed reality. In *USENIX Security Symposium*, 2023.

[28] TowerMax Fitness. Towermax fitness - tower. https://towermax.fitness/tower/, 2025. Accessed: 2025-01-21.

[29] Epic Games. Unreal engine: Real-time 3d creation platform. https://www.unrealengine.com/en-US, 2025. Accessed: 2025-01-14.

[30] Geniuscrate Games. Chair and sofa set. https://assetstore.unity.com/packages/3d/props/furniture/chair-and-sofa-set-263004, 2023. Accessed: 2025-1-5.

[31] Lukas Giner, Roland Czerny, Christoph Gruber, Fabian Rauscher, Andreas Kogler, Daniel De Almeida Braga, and Daniel Gruss. Generic and automated drive-by gpu cache attacks from the browser. In *ACM Asia Conference on Computer and Communications Security (Asia CCS)*, 2024.

[32] Google. Android 12. https://www.android.com/android-12/. Accessed: 2024-12-30.

[33] Immersive Web Working Group. Webxr samples: Interactive examples for immersive web development. https://immersive-web.github.io/webxr-samples/, 2025. Accessed: 2025-01-14.

[34] John K Haas. A history of the unity game engine. *Journal of Game Development*, 2014.

[35] IKEA. Ikea place. https://www.ikea.com/global/en/newsroom/innovation/ikea-launches-ikea-place-a-new-app-that-allows-people-to-virtually-place-furniture-in-their-home-170912/, 2021. Accessed: 2025-1-5.

[36] Immersive Web Working Group. Webxr samples: Positional audio. https://immersive-web.github.io/webxr-samples/positional-audio.html. Accessed: 2025-01-17.

[37] Immersive Web Working Group. Webxr - anchors sample. https://immersive-web.github.io/webxr-samples/anchors.html, n.d. Accessed: 2025-01-22.

[38] Yucheng Jin, Karsten Seipp, Erik Duval, and Katrien Verbert. Go with the flow: effects of transparency and user control on targeted advertising using flow charts. In *International Working Conference on Advanced Visual Interfaces*, 2016.

[39] Tami Kim, Kate Barasz, and Leslie K John. Why am i seeing this ad? the effect of ad transparency on ad effectiveness. *Journal of Consumer Research*, 2019.

[40] Jiachun Li, Yan Meng, Yuxia Zhan, Le Zhang, and Haojin Zhu. Dangers behind charging vr devices: Hidden side channel attacks via charging cables. *IEEE Transactions on Information Forensics and Security*, 2024.

[41] 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)*. IEEE, 2019.

[42] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. Platypus: Software-based power side-channel attacks on x86. In *IEEE Symposium on Security and Privacy (S&P)*, 2021.

[43] Chen Liu, Abhishek Chakraborty, Nikhil Chawla, and Neer Roggel. Frequency throttling side-channel attack. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[44] Meta. Horizon workrooms by meta. https://forwork.meta.com/horizon-workrooms/. Accessed: 2025-1-8.

[45] Meta. Most popular apps in meta qeust app store. https://www.meta.com/experiences/section/554169918379884/?srsltid=AfmBOooZZS9xp7-GoobOpDaDjuwzQm9AyNN0qHfHzBS_mqU0gPxF6EOG. Accessed: 2025-01-20.

[46] Meta. Meta asset store. https://assetstore.unity.com/, 2023. Accessed: 2025-1-5.

[47] Meta Horizon. Meta developer tools. https://developers.meta.com/horizon/unity/ts-ovrgpuprofiler/?doc_root=documentation. Accessed: 2025-03-12.

[48] Inc. Meta Platforms. Beat saber on meta quest. https://www.meta.com/experiences/pcvr/beat-saber/1304877726278670/?utm_source=beatsaber.com&utm_medium=oculusredirect, 2025. Accessed: 2025-01-14.

[49] Inc. Meta Platforms. Figmin xr: Mixed reality experience. https://www.meta.com/experiences/figmin-xr-mixed-reality/6849182851823457/?srsltid=AfmBOoo0KyTDPEkYb3YhKCueTFfCj3EH4wNBsKNfH5KvL1U5HYv2g6eh, 2025. Accessed: 2025-01-14.

[50] Inc. Meta Platforms. Ovr metrics tool. https://www.meta.com/experiences/ovr-metrics-tool/2372625889463779/?srsltid=AfmBOorTjESKXfP1Qp4B5menguU8wKd5_zF5VmUp5NSE5r9alopgbh_o#reviews, 2025. Accessed: 2025-04-22.

[51] Meta Platforms, Inc. Meta bug bounty program. https://bugbounty.meta.com/. Accessed: 2025-01-20.

[52] Meta Platforms, Inc. Meta experience: Layout. https://www.meta.com/experiences/layout/9298251876913852/?srsltid=AfmBOoqVclH-mSB-1G-o5fQiTVouJK8xM30FDhbtJ89bshTOdVMmQ9kZ, 2025. Accessed: 2025-07-16.

[53] Microsoft. Altspacevr-a social vr platform. https://altvr.com/. Accessed: 2025-1-8.

[54] Microsoft. Microsoft hololens. https://learn.microsoft.com/en-us/hololens/, 2024. Accessed: 2024-12-30.

[55] Microsoft. GPU Captures in Microsoft PIX. https://devblogs.microsoft.com/pix/gpu-captures/, 2025. Accessed: 2025-01-16.

[56] Microsoft. Microsoft HoloLens Documentation. https://learn.microsoft.com/en-us/hololens/, 2025. Accessed: 2025-01-16.

[57] Microsoft. Windows Performance Recorder. https://learn.microsoft.com/en-us/windows-hardware/test/wpt/windows-performance-recorder, 2025. Accessed: 2025-01-16.

[58] Reham Mohamed, Arjun Arunasalam, Habiba Farrukh, Jason Tong, Antonio Bianchi, and Z Berkay Celik. {ATTention} please! an investigation of the app tracking transparency permission. In *USENIX Security Symposium*, 2024.

[59] Reham Mohamed, Habiba Farrukh, Yidong Lu, He Wang, and Z. Berkay Celik. iStelan: Disclosing Sensitive User Information by Mobile Magnetometer from Finger Touches. *Privacy Enhancing Technologies (PETS)*, 2023.

[60] Moonrider. Moonrider. https://moonrider.xyz, 2025. Accessed: 2025-01-21.

[61] 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 (CCS)*, 2018.

[62] Vivek Nair, Wenbo Guo, Justus Mattern, Rui Wang, James F O'Brien, Louis Rosenberg, and Dawn Song. Unique identification of 50,000+ virtual reality users from head & hand motion data. In *USENIX Security Symposium*, 2023.

[63] Tao Ni, Yuefeng Du, Qingchuan Zhao, and Cong Wang. Non-intrusive and unconstrained keystroke inference in vr platforms via infrared side channel. *arXiv preprint arXiv:2412.14815*, 2024.

[64] Inc. Niantic. Pokémon go official website. https://pokemongolive.com/?hl=en, 2025. Accessed: 2025-01-14.

[65] Oracle. Processbuilder (java platform se 8). https://docs.oracle.com/javase/8/docs/api/java/lang/ProcessBuilder.html, 2025. Accessed: 2025-07-18.

[66] Stack Overflow. How to make a detached fork process? https://stackoverflow.com/questions/62837610/how-to-make-a-detached-fork-process, 2020. Accessed: 2025-07-18.

[67] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 2011.

[68] Alexander Pertus. The future is virtual: How ar and vr are redefining tourism. https://www.reportlinker.com/article/10399, 2025. Accessed: 2025-04-23.

[69] Kevin Picchi. Webxr games: Collection of webxr-based games. https://github.com/PicchiKevin/WebXR-games, 2025. Accessed: 2025-01-14.

[70] Meta Platforms. Meta quest 2 specifications. https://www.meta.com/quest/products/quest-2/tech-specs/, 2024. Accessed: 2024-12-30.

[71] Meta Platforms. Meta quest 3 specifications. https://www.meta.com/quest/quest-3/, 2024. Accessed: 2024-12-30.

[72] Program-Ace. Top 5 virtual reality trends of 2025 — the future of vr. https://program-ace.com/blog/virtual-reality-trends/, 2025. Accessed: 2025-04-23.

[73] Antonio Ruggia, Andrea Possemato, Savino Dambra, Alessio Merlo, Simone Aonzo, and Davide Balzarotti. The dark side of native code on android. *ACM Transactions on Privacy and Security*, 2025.

[74] Son Seonghun, Dipta Debopriya Roy, and Gulmezoglu Berk. Defweb: Defending user privacy against cache-based website fingerprinting attacks with intelligent noise injection. In *Computer Security Applications Conference (ACSAC)*, 2023.

[75] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y Zhao. A real-time defense against website fingerprinting attacks. *arXiv preprint arXiv:2102.04291*, 2021.

[76] Jeff Shepard. What sensors are used in AR/VR systems? https://www.sensortips.com/featured/what-sensors-are-used-in-ar-vr-systems-faq/, 2022. Accessed: 2025-04-23.

[77] Cong Shi, Xiangyu Xu, Tianfang Zhang, Payton Walker, Yi Wu, Jian Liu, Nitesh Saxena, Yingying Chen, and Jiadi Yu. Face-mic: inferring live speech and speaker identity via subtle facial dynamics captured by AR/VR motion sensors. In *International Conference on Mobile Computing and Networking (MobiCom)*, 2021.

[78] Carter Slocum, Yicheng Zhang, Nael Abu-Ghazaleh, and Jiasi Chen. Going through the motions: AR/VR keylogging from user head motions. In *USENIX Security Symposium*, 2023.

[79] Sindre Sorhus. Awesome webxr: A curated list of webxr projects and resources. https://project-awesome.org/msub2/awesome-webxr#standout-projects, 2025. Accessed: 2025-01-14.

[80] Brick Project Studio. Apartment kit. https://assetstore.unity.com/packages/3d/environments/apartment-kit-124055, 2023. Accessed: 2025-1-5.

[81] Spatial System. Spatial - the future of work in virtual reality. https://spatial.io/. Accessed: 2025-1-8.

[82] Glitch Team. Glitch: The friendly community for building the web. https://glitch.com/, 2025. Accessed: 2025-01-14.

[83] VRSites Team. Vrsites: Explore virtual reality websites and experiences. https://vrsites.com/, 2025. Accessed: 2025-01-14.

[84] WebXR Metaverse Team. Webxr metaverse: Explore web-based xr experiences. https://www.webxr-metaverse.com/, 2025. Accessed: 2025-01-14.

[85] Unity Technologies. Unity: Real-time development platform. https://unity.com/, 2025. Accessed: 2025-01-14.

[86] Unity Technologies. Unity scripting api: Androidjnihelper. https://docs.unity3d.com/ScriptReference/AndroidJNIHelper.html, 2025. Accessed: 2025-07-18.

[87] The Business Research Company. AR/VR chip global market report 2025. https://www.thebusinessresearchcompany.com/report/ar-or-vr-chip-global-market-report, 2025. Accessed: 2025-04-23.

[88] Pier Paolo Tricomi, Federica Nenna, Luca Pajola, Mauro Conti, and Luciano Gamberini. You can't hide behind your headset: User profiling in augmented and virtual reality. *IEEE Access*, 2023.

[89] Tridify. Hdrp furniture pack. https://assetstore.unity.com/packages/3d/props/furniture/hdrp-furniture-pack-153946, 2023. Accessed: 2025-1-5.

[90] Unity. Preparing assets for unity. https://docs.unity3d.com/2019.3/Documentation/Manual/Glossary.html#Unityunit, 2025. Accessed: 2025-01-17.

[91] Vartiste. Vartiste. https://vartiste.xyz, 2025. Accessed: 2025-01-21.

[92] VRChat. Vrchat. https://hello.vrchat.com/. Accessed: 2025-1-8.

[93] Hanqiu Wang, Zihao Zhan, Haoqi Shan, Siqi Dai, Maximilian Panoff, and Shuo Wang. Gazeploit: Remote keystroke inference attack by gaze estimation from avatar views in vr/mr devices. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024.

[94] Lance Williams. Pyramidal parametrics. In *Conference on Computer graphics and interactive techniques*, 1983.

[95] Yi Wu, Cong Shi, Tianfang Zhang, Payton Walker, Jian Liu, Nitesh Saxena, and Yingying Chen. Privacy leakage via unrestricted motion-position sensors in the age of virtual reality: A study of snooping typed input on virtual keyboards. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2023.

[96] Zhuolin Yang, Zain Sarwar, Iris Hwang, Ronik Bhaskar, Ben Y Zhao, and Haitao Zheng. Can virtual reality protect users from keystroke inference attacks? In *USENIX Security Symposium*, 2024.

[97] Tianfang Zhang, Zhengkun Ye, Ahmed Tanvir Mahdad, Md Mojibur Rahman Redoy Akanda, Cong Shi, Yan Wang, Nitesh Saxena, and Yingying Chen. Facereader: unobtrusively mining vital signs and vital sign embedded sensitive info via AR/VR motion sensors. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023.

[98] Yicheng Zhang, Carter Slocum, Jiasi Chen, and Nael Abu-Ghazaleh. It's all in your head (set): Side-channel attacks on AR/VR systems. In *USENIX Security Symposium*, 2023.

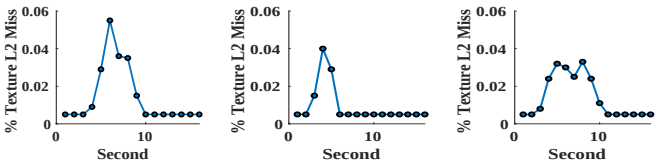
[99] Zhenkai Zhang, Sisheng Liang, Fan Yao, and Xing Gao. Red alert for power leakage: Exploiting intel rapl-induced side channels. In *ACM Asia Conference on Computer and Communications Security (Asia CCS)*, 2021.

APPENDIX A
GPU METRICS SELECTION BY PEARSON CORRELATION

TABLE VII: Pruned 11 GPU metrics and reasons after pairwise Pearson correlation analysis. If the threshold is redundant, drop the second metric in each pair. Metrics that never appear in a high-correlation pair are kept (Uncorrelated).

Metric Name	Justification
GPU Bus Busy	Prims Clipped (r = -0.995) Texture Fetch Stall (r = 0.911) Texture L2 Miss (r = 0.907) Stalled on System Memory (r = 0.913)
Vertex Fetch Stall	Prims Trivially Rejected (r = 0.908) Nearest Filtered (r = 0.906) Avg Bytes / Fragment (r = 0.907) Global Image Uncompressed Data Read (r = 0.918)
Anisotropic Filtered	Uncorrelated
Non-Base Level Textures	Uncorrelated
SP Memory Read (Bytes/Second)	Global Buffer Read L2 Hit (r = -0.932) Bytes Data Actually Written (r = -0.909) Global Buffer Data Read BW (r = 1.000)
Preemptions / second	Global Buffer Data Read Request BW (r = 0.919)
Avg Preemption Delay	Uncorrelated
Global Memory Load Instructions	Uncorrelated
Local Memory Store Instructions	Uncorrelated
Avg Load-Store Instructions Per Cycle	Uncorrelated
Bytes Data Write Requested	Uncorrelated

APPENDIX B
GPU METRICS IN RELATION TO SPEED AND DEPTH



(a) $v = 1, z = 2$ (b) $v = 2, z = 2$ (c) $v = 1, z = 3$

Fig. 11: Correlation between `Texture L2 Miss` metric and VR Cube object rendering with (a) the speed, v = 1 $unit/second$ and distance coordinate $z = 2$ $unit$, (b) increasing speed to v = 2 $unit/second$, and (c) move object further from the point of view, $z = 3$ $unit$.

APPENDIX C
AR/VR STANDALONE APP FINGERPRINT

TABLE VIII: Accuracy and F1 scores for the top-performing metrics employing a Random Forest classifier on 100 standalone AR/VR apps.

Metric Name	Accuracy	F-1 Score
Average Vertices / Polygon	0.945	0.935
Vertex Memory Read (Bytes/Second)	0.940	0.931
Prims Trivially Rejected	0.935	0.921
Average Polygon Area	0.920	0.905
Vertex Instructions / Second	0.920	0.912
L1 Texture Cache Miss Per Pixel	0.915	0.900
Prims Clipped	0.910	0.885
Global Image Uncompressed Data Read BW	0.902	0.868
Texture L2 Miss	0.900	0.881
Pre-clipped Polygons/Second	0.897	0.890
Avg Bytes / Fragment	0.895	0.860
GPU Bus Busy: Accuracy	0.890	0.860

APPENDIX D
WEBXR APP FINGERPRINT

TABLE IX: Accuracy and F1 scores for the top-performing metrics employing a Random Forest on 100 WebXR apps.

Metric Name	Accuracy	F-1 Score
Average Polygon Area	75.7	75.2
Average Vertices / Polygon	73.25	72.9
% Texture L2 Miss	72.75	72.21
% Texture Fetch Stall	72.0	71.1
% Prims Clipped	68.5	67.8
L1 Texture Cache Miss Per Pixel	68	68.1
GPU % Bus Busy	65.7	65.6

APPENDIX E
VIRTUAL OBJECT DETECTION

TABLE X: CNN F1 score (%) across varying speeds (v) and distances (z) in the default VR scene (Figure 7a).

Speed (v)	Distance (z)		
	0	5	10
1	96.5	97.1	95.7
10	72.1	86.7	93.1
50	68.5	78.9	89.1

TABLE XI: Accuracy and F1 scores for the top-performing 17 metrics employing a Random Forest classifier on 35 virtual products.

Metric Name	Accuracy	F-1 Score
Avg Bytes / Fragment	0.993	0.992
Global Image Uncompressed Data Read BW	0.979	0.978
Global Memory Load Instructions	0.964	0.961
% Non-Base Level Textures	0.964	0.968
% Texture L2 Miss	0.957	0.960
Global Buffer Data Read Request BW	0.943	0.941
SP Memory Read	0.893	0.866
Global Buffer Data Read BW	0.893	0.866
% Prims Clipped	0.864	0.806
Reused Vertices / Second	0.864	0.855
% Prims Trivially Rejected	0.836	0.779
Pre-clipped Polygons/Second	0.779	0.744
Vertex Memory Read	0.750	0.743
% Vertex Fetch Stall	0.736	0.686
Vertices Shaded / Second	0.736	0.695
Vertex Instructions / Second	0.714	0.651
% Anisotropic Filtered	0.621	0.649

TABLE XII: Core GPU metrics retained after correlation-driven pruning in Case Study III VII-C

Metric Name
% Vertex Fetch Stall
SP Memory Read (Bytes/Second)
Global Memory Load Instructions
% Anisotropic Filtered
% Non-Base Level Textures

APPENDIX F
PER-DEVICE AND CROSS-DEVICE EVALUATION

To verify that OVRWATCHER generalizes across the entire Meta Quest family, we repeated all four case studies (AR/VR standalone (I) and WebXR (II) app fingerprint, virtual object detection (III), and meeting room participant inference (IV)) on Quest 2, Quest 3, and the most recent Quest 3S headsets. For each device, we collected the same GPU metrics under identical scene setups, focusing on the top 3-4 metrics identified based on our prior analysis.

First, we performed per-device evaluation by splitting each device's dataset into 80% for training and 20% for testing. Then we trained four classifiers (CNN, LSTM, RF, and SVM) independently on each device (Table XIII). Because Quest 3 and Quest 3S share the same GPU specification (Section VI-C), we report standalone (I) app fingerprinting only on Quest 2 and Quest 3. Next, to account for hardware variability even within the same model, we conducted a two-device test on Quest 2 with Case Study 1 to demonstrate effectiveness on app fingerprinting attacks. We trained our models on data from one Quest 2 headset and saved the pretrained weights. Then we evaluated 15 standalone apps without retraining the ML models (Table XIV). Finally, we merged 80% of the training and 20% of the testing data from all three headsets to train a single cross-series model (Table XV).

Our results show that OVRWATCHER remains highly effective across hardware generations and headset variations, using as few as 3–4 GPU metrics.

TABLE XIII: Per-device accuracies for each model across Quest 2, Quest 3, and Quest 3S.

Case Study (# of GPU Metrics)	Model	Accuracy (%)		
		Quest 2	Quest 3	Quest 3S
I. AR/VR Standalone App (4)	LSTM	95.70	89.25	—
	CNN	97.31	87.90	—
	RF	**98.39**	**95.97**	—
	SVM	94.89	80.65	—
II. WebXR App (3)	LSTM	83.74	83.25	83.50
	CNN	87.62	87.75	91.00
	RF	**92.72**	**96.25**	**93.50**
	SVM	40.29	38.00	36.00
III. Virtual Object Detection (3)	LSTM	90.71	60.00	72.14
	CNN	94.29	68.57	95.71
	RF	**99.29**	**92.86**	**100.00**
	SVM	75.71	19.29	28.57
IV. Meeting Room Inference (3)	LSTM	80.00	80.00	70.00
	CNN	**100.00**	**100.00**	**100.00**
	RF	**100.00**	**100.00**	**100.00**
	SVM	100.00	95.00	90.00

TABLE XIV: Two-device cross-evaluation on Quest 2 for AR/VR standalone app fingerprinting (Case Study I).

Case Study (# of GPU Metrics)	Model	Quest 2			
		Accuracy (%)	Precision	Recall	F1
I. AR/VR Standalone (4)	LSTM	70.67	0.596	0.707	0.621
	CNN	**80.00**	**0.693**	**0.800**	**0.730**
	RF	37.33	0.306	0.373	0.310
	SVM	53.33	0.435	0.533	0.461

TABLE XV: Cross-device evaluation of OVRWATCHER: final test results (Accuracy, Precision, Recall, F1) for four classifiers (CNN, LSTM, RF, SVM) across the four case studies on the entire Meta Quest series.

Case Study (# of GPU Metrics)	Model	Accuracy	Precision	Recall	F1
I. AR/VR Standalone App (4)	LSTM	89.38%	0.913	0.894	0.889
	CNN	89.92%	0.924	0.899	0.899
	RF	**95.83%**	**0.963**	**0.958**	**0.958**
	SVM	72.72%	0.755	0.727	0.716
II. WebXR App (3)	LSTM	68.09%	0.668	0.667	0.639
	CNN	70.66%	0.734	0.703	0.672
	RF	**97.15%**	**0.968**	**0.969**	**0.966**
	SVM	13.39%	0.124	0.153	0.127
III. Virtual Object Detection (3)	LSTM	67.62%	0.731	0.657	0.649
	CNN	37.38%	0.398	0.407	0.331
	RF	**93.57%**	**0.933**	**0.935**	**0.930**
	SVM	24.76%	0.209	0.263	0.209
IV. Meeting Room Inference (3)	LSTM	90.00%	0.850	0.900	0.867
	CNN	**100.00%**	**1.000**	**1.000**	**1.000**
	RF	**100.00%**	**1.000**	**1.000**	**1.000**
	SVM	88.33%	0.878	0.883	0.876

APPENDIX G
AR/VR STANDALONE APPLICATION LIST

TABLE XVI: Categorization of top 100 AR/VR standalone applications from the Meta Quest app store.

Category	Application Name
Gaming (31)	Gorilla Tag, PokerStars VR, GunRaiders, CardsTankards, Supernatural Shootout, ForeVR Cornhole, Roblox, Yeeps, Innerworld, SHARKS, Big Ballers VR, Noclip VR, Monkey Doo, HyperDash, Primal Apes, Beastcraft, Hell Horde:Mixed Reality Survival, Dissection Simulator:Frog Edition, Truck Parking Simulator VR Demo, Gun Shooting Range with Pistol, Running Monkeys, A2RL VR, Animal Company, Scary Baboon, Untangled, Gorilla Warzone, Modified Bed Bath Simulator, WIN Reality Baseball, Machine Shop Simulator, NeVR Fear The Dentist, Preflight Simulator
Entertainment (21)	Bigscreen Beta, Anne Frank House VR, Epic Roller Coasters, Amazon Prime Video, Oktoberfest, 4XVRVideoPlayer, Pianogram, Campfire, VENTA X, Notre-Dame de Paris, StartVR Streaming Video Player, FloatVR Relaxation and Focus, Mobile VR Station, PlayAniMaker Beta, Wolvic, ecosphere, Fluid, Spaceframe, Prism, Maloka, Viso
Fitness (10)	Xtadium, Alcove, TRIPP, FitXR, Shado Running, VRFS, Litesport, XRWorkout, Baseball Softball Training, Hoame
Social (7)	Horizon Worlds, Multiverse, VRChat, WhatsApp, Twitch:Live Streaming, MeetinVR, Flipside
Productivity (27)	Immersed, Horizon Workrooms, Meta Quest Browser, Engage, Spatial, Human Anatomy VR, Lab Monkey, GRAB, vSpatial, ALVR, Hard Drive, Nanome, VR Anatomy Lab, SketchUp Viewer, IMMERSE–Language Learning, Arkio, ShapesXR, Doodle Board, Gravity Sketch, Resolve, Mistika VR Connect, DelTrain Adult ICU Delirium, Arthur, Noda, Naer, Masterpiece, Open Brush–3D Painting
Mixed Reality (4)	A2RL VR, Hell Horde:Mixed Reality Survival, Hello Dot, Hololight Space

APPENDIX H
WEBXR APPLICATION LIST

TABLE XVII: Categorization of 100 WebXR applications.

Category	Application Name
Gaming & Entertainment(19)	moonrider.xyz/, webvr.soundboxing.co/, plockle.com/play, spiderman.webvr.link/, jorgefuentes.net/projects/halloVReen/, aboveparadowski.com/, anumberfromtheghost.com/, jorgefuentes.net/projects/vuppets/vuppets_6DOF, spacerocks.moar.io, blocksarcade.xyz/, xrpet.me/, crossthestreet.fun/game/, aframe.io/a-blast/, micosmo.com/trajectilecommand, beatknightxr.web.app, play.js13kgames.com/human-not-found/, worldsdemolisher.totalviz.com, slime-freighter.glitch.me/?autoplay=true, play.js13kgames.com/teleport/
Art & Creativity (11)	aframe.io/a-painter/, brushworkvr.com/paint, vartiste.xyz/, esc.art/, flowerbed.metademolab.com/, artsalad.net/, castle.needle.tools/, cecropia.github.io/thehallaframe, demos.littleworkshop.fr/track, hatsumi.netlify.app, framevr.io/frame-tutorial
Tours & Exploration (18)	immersive-web.github.io/webxr-samples/360-photos.html, immersive-web.github.io/webxr-samples/stereo-video.html, aframe.city/, jorgefuentes.net/projects/puppetrilla, a-frame–360-vr-tour.glitch.me, forestwave.glitch.me, 3xr.space, travisbarrydick.github.io/vr-planets/dist, xrdinosaurs.com, msub2.github.io/hello-webxr, live.arrival.space/welcome, codercat.xyz/ma, anvropomotron.com, codercat.xyz/dying-to-find, a.flow.gl/flow/i2nxns/display, codercat.xyz/three-body, new-cesium-a-frame-test.glitch.me, modelo-3d-a-frame.glitch.me
Health & Fitness (8)	towermax.fitness/tower/, towermax.fitness/peakfreak/, towermax.fitness/reaction/, towermax.fitness/punchingballgames/, towermax.fitness/slingsurf/, towermax.fitness/towerhockey/, towermax.fitness/stepuprun/, towermax.fitness/drilltrack/
Demonstration (44)	From aframe.io/aframe/examples/—anime-UI, shopping, comicbook, spheres-and-fog, dynamic-lights, hand-tracking-grab-controls, boilerplate/ar-hello-world, test/fog, test/visibility, performance/animation-raw, animation/arms, animation/pivots, animation/unfold, performance/multiview-extension/?multiview=on, test/shadows, primitives/torus, docs/aincraft; From immersive-web.github.io/webxr-samples—Reduced-bind-rendering.html, room-scale.html, input-tracking.html, input-selection.html, controller-state.html, positional-audio.html, anchors.html; From https://glitch.com/—a-frame-particules-rain.glitch.me, tree-by-a-frame-by-jenjira-santaw.glitch.me, a-frame-goofs.glitch.me, ma3120-a-frame-space-image-02.glitch.me, ma3120-a-frame-image-layered-repetition.glitch.me, ma3120-a-frame-texture-and-grab.glitch.me, military-heathered-creator.glitch.me, fluttering-robust-kiwi.glitch.me, a-frame-spinosaurus-for-vr.glitch.me, a-frame-physics-entities-balls-fall-from-the-sky.glitch.me, a-frame-wolf.glitch.me, a-frame-stationary-caustics.glitch.me, celda-de-trabajo-automatizada-en-a-frame.glitch.me, a-frame-snowman-rv.glitch.me, simple-solarsystem-a-frame.glitch.me, city-whit-a-frame-5a.glitch.me, lmc-2700-a-frame-project.glitch.me, a-custom-a-frame-scene.glitch.me, dune-a-frame-navmesh.glitch.me, a-frame-tree-model-loading.glitch.me