XRgo: Design and Evaluation of Rendering Offload for Low-Power Extended Reality Devices

Steven Gao*
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
hongyig3@illinois.edu

Jeffrey Liu*
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
jliu179@illinois.edu

Qinjun Jiang
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
qinjunj2@illinois.edu

Finn Sinclair
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
finnnorth@gmail.com

William Sentosa
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
sentosa2@illinois.edu

Brighten Godfrey
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
pbg@illinois.edu

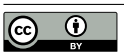
Sarita Adve
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
sadve@illinois.edu

Abstract

Extended reality (XR) devices must render high-quality 3D graphics at low latency to deliver truly immersive experiences. However, XR devices are severely power- and resource-constrained, limiting the quality of on-device (local) rendering. Offloading rendering to a powerful remote machine can enhance graphics quality, but network latency can degrade the overall experience. To mask latency, XR systems reproject the rendered frame to compensate for user motion since the rendered pose. Traditional reprojection, known as TimeWarp, uses a lightweight mechanism to compensate for latency in rotational motion, but not translational motion. Compensating for translational motion is more expensive, but is increasingly important at higher latencies.

We present XRgo, the first open-source XR runtime that supports application-transparent offloading of rendering to a remote server with client-side six-degrees-of-freedom (6-DoF) reprojection. XRgo uses OpenWarp, the first open-source and OpenXR-compatible lightweight 6-DoF mesh-based reprojection technique. Compared to on-device rendering, XRgo consistently renders at the target frame rate while consuming less power. User studies show that on average, OpenWarp provides a superior XR experience than TimeWarp. Further, OpenWarp only consumes $< 2\%$ more power than TimeWarp. Our findings reveal that the quality of experience depends on both network average latency and variability, highlighting the need for end-to-end evaluations under live network conditions.

*The first two authors contributed equally to the paper.



MMSys'25, March 31–April 4, 2025, Stellenbosch, South Africa

ACM ISBN 979-8-4007-1467-2/25/03
https://doi.org/10.1145/3712676.3714444

CCS Concepts

• **Computing methodologies** → **Virtual reality**; **Perception**; Image-based rendering; • **Computer systems organization** → *Real-time system architecture*; • **Human-centered computing** → *Ubiquitous and mobile computing design and evaluation methods.*

Keywords

Extended reality, remote rendering, asynchronous reprojection, offloading, low power, wireless networks

ACM Reference Format:
Steven Gao, Jeffrey Liu, Qinjun Jiang, Finn Sinclair, William Sentosa, Brighten Godfrey, and Sarita Adve. 2025. XRgo: Design and Evaluation of Rendering Offload for Low-Power Extended Reality Devices. In *The 16th ACM Multimedia Systems Conference (MMSys'25), March 31–April 4, 2025, Stellenbosch, South Africa.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3712676.3714444

1 Introduction

Extended reality (XR) devices enable the possibility of highly immersive experiences. However, truly immersive experiences require an XR headset to render high-quality 3D graphics with low latency while consuming low power. These devices must also have a lightweight form factor, thus introducing further constraints on computational power. Recently, offloading computation to a more powerful machine has introduced the potential for significant power savings. Remotely rendering scenes allows for increased geometric and shading complexity, as mobile-level devices are still largely limited to low polygon counts with forward shading. Remote machines can also benefit from techniques such as hardware-accelerated ray tracing [26] and the compute capacity to render recent graphics primitives (e.g., neural models) [25, 37] in real-time. However, the naïve solution of video streaming without compensation mechanisms suffers

from the inherent latency caused by network conditions to the point where the experience may be intolerable.

To mask this additional latency, the first line of compensation introduces post-rendering rotational reprojection, also known as TimeWarp (TW) [53], which reprojects (or warps) the rendered frame according to the user's current pose. However, TW can only account for differences in head rotation and cannot properly mask translational differences. The main challenge of reprojecting for translational motion is that new information can be introduced, e.g., object disocclusion. When this information is not available from the most recent frame, missing pixels must be filled in. The problem is that effective hole-filling techniques are usually expensive and cannot run at target frame rates. Any supplemental data used by these algorithms would also occupy more network bandwidth.

Recent advancements have introduced candidate solutions to the above challenges, but they are either proprietary, target mobile/desktop displays, simulate constant network latencies, and/or lack power measurements [7, 16, 29, 44]. It is unclear if these techniques and their evaluations directly translate to XR head-mounted displays (HMDs) under live network conditions. Furthermore, these algorithms also require modifications to the renderer, making it difficult to integrate them with existing applications immediately.

This paper presents the design and evaluation of XRgo, the first open-source end-to-end XR runtime that addresses the above concerns. Specifically, we make the following contributions:

- We present XRgo: e**x**tended **r**eality with **g**raphics **o**ffloaded, the first end-to-end open-source[1] XR runtime that supports application-transparent and OpenXR-compliant[2] offloading of rendering to a remote server with client-side six-degrees-of-freedom (6-DoF) asynchronous reprojection.
- XRgo uses OpenWarp, the first open-source[3] and OpenXR-compatible lightweight mesh-based 6-DoF reprojection algorithm. OW remains lightweight by using fixed-function rasterization hardware for hole-filling, leveraging depth data provided by the OpenXR API.
- We conduct the first user study paired with power measurements to compare quality and power between on-device rendering, offloading with TW, and offloading with OW.
- We demonstrate that, on average, OW provides better user experiences than TW, and only consumes $< 2\%$ more power.
- We show that the quality of experience depends on network latency and variability, differing considerably between live networks and constant latency simulations.

Our findings underscore the need for end-to-end systems and evaluations with network variability to study XR offloading.

2 Background and Related Work

2.1 Reprojection

Users are generally more sensitive to rendering latency when viewing through a headset than a traditional flat display. A commonly used metric is motion-to-photon (MTP) latency [54], the time from the user's motion with a new pose to the time that the displayed frame changes to reflect the new pose. Studies estimate that MTP latency should be less than 20 ms for virtual reality (VR) and 5 ms for augmented reality (AR) [12, 23]. XR runtimes attempt to mask higher end-to-end latencies through a combination of pose prediction and asynchronous reprojection (or warping). The application first renders a frame with the pose predicted for the display time. Then, right before display time, the runtime executes a reprojection pass using an updated pose [54]. If the application renderer takes too long and misses the display time, the runtime reprojects the previously rendered frame using the new pose. To minimize latency, it is important for the reprojection pass to run as close to the display time as possible. Asynchronous reprojection thus enables display responsiveness at the target frame rate even if the application renderer itself is running at intolerably high latencies.

Rotational reprojection [42, 53], mostly commonly TimeWarp (TW) [53], is an inexpensive reprojection technique that has seen widespread adoption. However, rotation-only TW only accounts for changes in the user's head rotation, and cannot compensate for differences in translational motion, which become noticeable at higher latencies [6]. To combat this, XR runtimes such as the Oculus Quest 2 have introduced translational reprojection techniques [7]. These runtimes are proprietary and the impact of translational reprojection on end-to-end XR performance or power has not been studied in the literature.

In the broader graphics literature, reusing information from previous frames has a long history of research [46]. Early work used forward reprojection techniques [35, 55], which suffered from holes from missing information. Nehab et al. presented a reverse reprojection cache [38] that could be used for reverse evaluation, but required additional computation to fill in holes. Yang et al. proposed a bidirectional, or hybrid, approach for reprojection into intermediate frames [58], but such an algorithm requires frames from the *future*, making it unsuitable for our purpose.

Some geometry-based warping algorithms [16, 34, 45, 47, 50] are similar to OW in that they render with proxy geometry, but generally require multiple reference frames to generate and composite the final frame. Many are also significantly more expensive and cannot run at full frame rate on a mobile-level device. Other image-based rendering algorithms use depth peeling [27, 44] to supply additional hole-filling information. In contrast, OW only requires the previous frame as input (minimizing bandwidth) and does not use an additional composition pass (minimizing compute), making it more suitable for an offloaded scenario. Our user studies from Section 6.1 demonstrate that these simplifications are acceptable.

Outatime is a cloud gaming system that compensates for network latency by speculatively pre-rendering frames [29]. Though not designed for XR, Outatime's error correction uses a coarse-mesh-based reprojection and hole-filling method similar to OW. However, as detailed in Section 3.3, OW uses a different overscanning mechanism to remain OpenXR-compatible. Our method has lower bandwidth requirements, with user studies indicating that it is acceptable.

More recent work also includes the use of motion vectors and/or optical flow during frame extrapolation [5, 14, 39, 57, 59]. These

[1] XRgo is implemented within ILLIXR and the code is available at https://github.com/ILLIXR/ILLIXR.
[2] OpenXR is a popular runtime-agnostic standard, permitting application portability across OpenXR runtime implementations.
[3] OpenWarp was developed as one of the authors' B.S. thesis [13] with the original standalone implementation at https://github.com/Zee2/openwarp.

techniques further use this information for the purpose of supersampling to a higher resolution, and may also assist with reprojecting dynamic objects. Although OW does not incorporate such information, the OpenXR API supports feeding motion vectors to the runtime; our offloading system makes it easy to implement and evaluate such algorithms on XR headsets.

2.2 Offloaded Rendering for XR

There is much recent research on XR systems with offloaded rendering [2, 4, 22, 24, 28, 30–32, 36, 43], proposing a variety of techniques to address network latency and bandwidth as discussed below. Our work is the first to implement 6-DoF reprojection in a complete XR system and study its impact through a user study paired with power measurements.

Like our work, several previous studies have explored offloading all application rendering to a server. The open-source ALVR [2] and ElectricMaple [43] XR systems implement application-transparent offloaded rendering with client-side asynchronous TW. Liu et al. [30] propose to minimize offloading latency by parallelizing frame encoding, pipelining the streaming process, and coordinating server-side rendering with client-side display times. The CollabVR system from Ke et al. [24] aims to minimize encoding latency and bandwidth by exploiting frame redundancy in stereo images to only transmit the left-eye image. The FovOptix system [4] studies foveated rendering and video encoding under live network conditions with VR headsets. The design and evaluation of XRgo and OW are orthogonal and reciprocally complement these works.

To handle responsiveness in mobile VR systems, many offloading systems choose to split rendering between the client and server. Some systems [22, 28, 31, 36] render the background scene as a panoramic view on the server, while rendering the foreground or other interactive objects on the client. The RenderFusion system introduced by Lu et al. [32] further introduces a partitioning scheme to dynamically balance local and remote rendering, optimizing quality according to network conditions and on-device rendering capacity. OW is complementary to these techniques and can enable more foreground objects to be rendered on the server by better handling parallax effects, reducing the power and computations on the client. XRgo may also benefit from the above techniques for locally rendering interactive or dynamic objects, as OW is better suited for static scenes (however, they are not application-transparent and break OpenXR compatibility).

2.3 ILLIXR Runtime

ILLIXR [1, 18, 19] is an open-source XR runtime and research testbed. It provides various state-of-the-art XR components (as *plugins*) and facilitates their scheduling and communication. Communication is handled via the Switchboard framework, following a publish-subscribe model with shared memory. Each plugin can publish or subscribe to event streams, referred to as *topics*. Plugins can either execute a callback whenever new data is published, or asynchronously read the most recent data from a given topic.

Plugins are compiled as dynamic libraries that are composed at runtime. This allows for a modular system that enables swap-in implementations as long as all data flow dependencies are respected. This framework is also naturally suitable for offloading because the actual flow of data is abstracted away from individual plugins. For example, as long as the reprojection plugin has access to some rendered image and its associated pose, it does not need to know whether this image was rendered locally or decoded over the network. ILLIXR supports native applications as well as OpenXR applications through integration with Monado [11].

Similar to the RemoteVIO system proposed by Jiang et al. [21], we build on the ILLIXR runtime to transmit topics over a network. However, we target the graphics pipeline, while RemoteVIO targets the head tracking pipeline.

3 XRgo System Design

This section first describes the high-level modifications to an XR runtime to enable offloading rendering. We then provide a theoretical formulation of our reprojection algorithm, and discuss implications to consider when reprojecting for offloaded XR rendering.

3.1 System Overview

Figure 1a illustrates the key rendering-related components of a traditional on-device XR runtime. Figure 1b illustrates XRgo, with the runtime split between the client device and the server.

XRgo augments the traditional on-device XR runtime by adding a network driver to transmit poses and images between the server and client. The computationally expensive application rendering is moved to the server, which requires poses streamed from the client. When a new frame is rendered, the server encodes the frame and transmits it back to the client along with the frame's associated pose. For reprojection techniques such as OW, XRgo also supports transmitting the rendered depth associated with the stereoscopic images. The challenge of rendering remotely is that MTP latency accumulates from pose transmission to the server, rendering, encoding, frame transmission to the client, and decoding. Naïvely, this latency is intolerable without any compensation mechanisms.

The client machine now maintains a lighter runtime, running pose estimation and streaming the head pose to the server. Upon receiving a new encoded frame from the server, the client decodes the frame and pushes it to a buffer pool. Before each display period, the asynchronous reprojection pass uses the most recently decoded frame to display a reprojected image to the user.

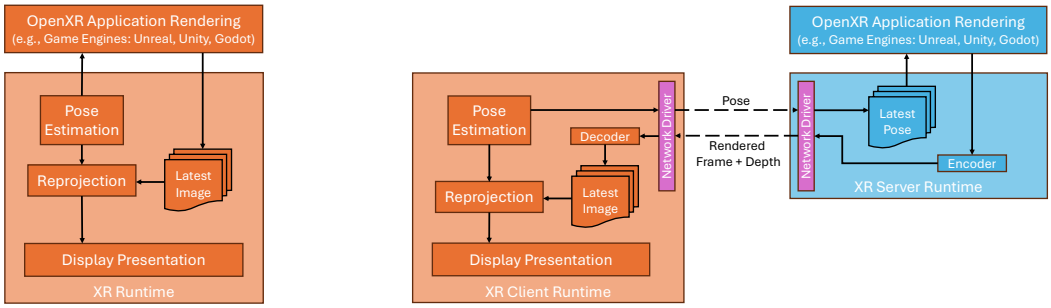
Compared to on-device rendering, the XRgo client-side decoder may be considered as the application renderer where higher latency is induced by the network and video codec. By reprojecting at the target frame rate using the latest pose, the client can effectively maintain a satisfactory MTP latency (with perceptual quality depending on the reprojection algorithm).

3.2 OpenWarp - 6-DoF Reprojection

We next describe OpenWarp, a lightweight, mesh-based 6-DoF reprojection algorithm.

Theoretical formulation. We start by introducing the basic warp equation that aims to reproject a source image from view 1 to view 2. For a given pixel at (u_1, v_1) with depth d_1, we can express its new image coordinates (u_2, v_2) with depth d_2 using Equation 1.

$$\begin{bmatrix} u_2 & v_2 & d_2 & 1 \end{bmatrix}^T = \mathbf{P}\mathbf{V}_2\mathbf{V}_1^{-1}\mathbf{P}^{-1} \begin{bmatrix} u_1 & v_1 & d_1 & 1 \end{bmatrix}^T . \quad (1)$$



(a) Original on-device runtime (b) XRgo runtime at the client and server

Figure 1: (a) Overview of rendering-related components in the base XR runtime with on-device application rendering. (b) Overview of the XRgo runtime, highlighting the key architectural differences from the on-device runtime. Application rendering is shifted from the client device to a remote server. The client and server instances of the runtime interface through network backends that transmit poses (from the client) and encoded frames and depth data (from the server).

Here, $\mathbf{P}$ is the projection matrix and $\mathbf{V}_1, \mathbf{V}_2$ are the view matrices corresponding to view 1 and view 2 respectively. However, a naïve forward evaluation of this equation between a source and destination image is neither injective nor surjective.

The challenge with a non-injective mapping is that multiple pixels from the source image may fit into a single pixel on the destination image, thus requiring a post-warp merging process based on each of the computed depth values. On the other hand, the challenge with non-surjectivity is that no pixels from the source image may map onto another pixel on the destination image, thus requiring an additional pass to fill in these holes.

OpenWarp algorithm. We use a *reprojection mesh*, which computes the forward warp equation for each vertex of a planar mesh instead of each pixel. This allows us to distort the reprojection mesh into a continuous polygonal approximation of the rendered scene. Once we have this mesh approximation of the scene, we rasterize the mesh from a novel view. An example is visualized in Figure 2c.

(a) Base mesh (b) Depth image (c) Reprojection mesh

Figure 2: OW transforms a uniform grid of vertices (a) using the depth image (b) back into world space as a reprojection mesh (c), thus serving as a continuous scene approximation for rendering at different views.

Using a sparse reprojection mesh not only requires less computation than the per-pixel computation of a naïve per-pixel forward evaluation, but also takes advantage of two key stages of the graphics pipeline to directly address the non-injectivity and non-surjectivity of the warp equation. Firstly, even if multiple source pixels may map to the same destination pixel, merging is automatically handled by depth testing. On the other hand, even if some destination pixels have no direct mapping, the continuous representation of the reprojection mesh induces an automatic hole-filling during the fragment shader due to vertex attribute interpolation. Both of these advantages come directly from rasterization's hardware-accelerated fixed-function pipeline.

Depth adjustment. Since the reprojection mesh is a continuous representation, large discontinuities between foreground and background objects introduce an edge case that isn't well-handled by attribute interpolation. In these cases, if one vertex lies on the foreground while another lies in the background, interpolation may be obvious due to significant color differences (Figure 3a).

(a) Default depth (b) Adjusted depth

Figure 3: Without any depth adjustments in (a), the color interpolation between foreground and background is obvious. Correcting for this in (b) leads to small bump (overhang) artifacts. For backgrounds without complex patterns, these artifacts remain unobtrusive.

We handle this by checking depths within a small neighborhood of pixels. If any neighboring pixel's depth is significantly closer than the current pixel's depth (by some threshold M), the current pixel is identified as a boundary between the background and foreground. The boundary pixel's corresponding vertex depth is set to that of the closer pixel, ensuring that large stretches in the reprojection mesh mostly interpolate background colors alone. Figure 4 visualizes this depth adjustment. Since background vertices are shifted to the foreground, there is a small visible "overhang" around the edges of foreground objects, which may or may not be obtrusive depending on the scene (Figure 3b).

In most cases, overhang artifacts are more tolerable than the noticeable interpolation between contrasting background and foreground colors. Overhang artifacts also become less noticeable with finer reprojection mesh resolutions. Our results in Section 6.1 show that they are usually acceptable with our selected configurations.

3.3 Quality/Power/Bandwidth Tradeoffs

Our system design and 6-DoF reprojection algorithm introduce a set of tradeoffs to be considered when offloading rendering.

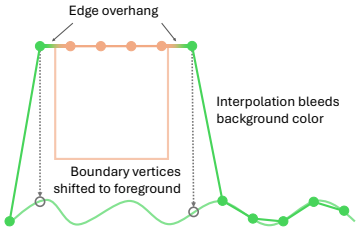


Figure 4: A 2D example scene and overlaid reprojection mesh. Given the orange box in the foreground and green waves in the background, the reprojection mesh forms an approximate scene. To avoid over-stretching between the orange and green vertices, the boundary background vertex is adjusted to match the foreground depth. This creates a slight overhang, where some background colors are closer than they should be, but reduces foreground-background stretching.

Video compression. The choice to offload rendering requires color and depth compression. When compressing at a given bitrate, two main encoding parameters affect image quality:

- *Frame rate*. The encoding frame rate is decoupled from the server-side rendering frame rate, and the most recently rendered frame is encoded when desired. At higher encoding frame rates, individual frames must be compressed more heavily, resulting in more compression artifacts and lower visual quality. At lower encoding frame rates, the client-side reprojection algorithm may need to compensate for longer latencies due to additional gaps in decoded frames.
- *Resolution*. Video encoding usually encodes at equal or lower resolution than the source. Increasing the rendered frame resolution increases the amount of information that must be compressed into a fixed number of bits, potentially reducing the image quality by introducing compressed artifacts.

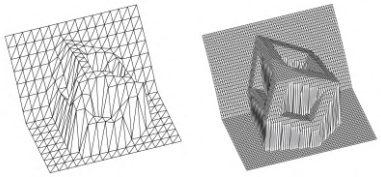
Network streaming. Although offloading removes the task of rendering from the client, it also introduces the overheads of transmitting poses, receiving frames, decoding the frames, and potentially more complex reprojection. All of these overheads increase device power consumption, so offloading should only be considered if these overheads incur less power than local rendering.

Depth compression. TW only requires color, while OW requires both color and depth. At a fixed bandwidth, TW can encode color at a higher bitrate because OW must transmit the depth as well. Thus, when there is little translational motion to compensate for, TW may have clearer visual quality compared to OW.

Overscanned field-of-view (FoV) and effective resolution. Since OW only uses the server-rendered frame to generate the reprojection mesh for re-rendering, the server-rendered frame's parameters can be decoupled from those of the final image.

For example, the server may render at an *overscanned* FoV for the client to generate a larger-than-current-frame reprojection mesh. The advantage of this approach is that more information at the borders is available during reprojection, which may potentially compensate for more latency. However, this also means that only a portion of each encoded frame is used during client-side reprojection. To maintain the same effective resolution as before (without overscanning), the encoded frame resolution may need to increase according to the rendered image's increased angular area. However, this incurs the video compression tradeoffs discussed earlier. In our selected configurations, we only overscan by 10% and find that increasing the frame resolution is not strictly necessary.

Grid Resolution. Due to the algorithm's use of vertex attribute interpolation, a finer grid resolution introduces less noticeable interpolation artifacts between vertices. An example is shown in Figure 5. However, this costs more client-side compute power.



(a) 16x16 mesh **(b) 64x64 mesh**

Figure 5: Resolution of the reprojection mesh affects the quality of the scene approximation. The (a) 16x16 mesh does not capture as many details as the (b) 64x64 mesh, but the former requires less compute (and power) for reprojection.

4 System Implementation

XRgo is implemented on top of the ILLIXR runtime (running on the server and the client) and supports OpenXR applications. The OpenXR application is simply run on the server and is not aware of any offloading scheme. The server-side runtime compresses the application frames and transmits them to the client for decoding (Section 4.1). Communication between the server and client is facilitated through a network backend in ILLIXR (Section 4.2).

Server: The server-side runtime composites and compresses the rendered frame submitted by the application. We modify the compositor to pass rendered frame depth, when available, through the entire composition pipeline. Our implementation then injects a post-composition ILLIXR plugin to compress and transmit (rather than display) the frame to the client.

Client: The client-side runtime implements an additional plugin to push poses to the server, and receive transmitted frame packets to decode. Decoded frames are pushed to a topic where a separate plugin, running at the target display rate, reprojects the most recently decoded frame according to the current pose.

4.1 Video Compression

We modify FFmpeg's [51] `hevc_nvenc` to avoid any redundant GPU-CPU memory copies during compression, such that all encoding/decoding operations remain on the GPU. Any unnecessary copies significantly increase the encoding/decoding pipeline latency. We encode both color and depth images via H.264 4:2:0 in BT.709 colorspace with extended luma range [9]. We cast the 32-bit depth to the same 8 bits across three RGB channels to conform to the codec's bit-depth. Although this is generally undesirable for encoding depth, we show in Section 6.1 that OW remains robust.

On our NVIDIA Jetson platform (discussed in Section 5.1), we implement a pipeline to decode the frames using the Jetson Multimedia API [40] and convert the decoded frames from multiplanar format to a linear RGBA format. The rest of the graphics pipeline uses the converted image for reprojection.

4.2 Network Backend

To transmit poses and encoded color and depth images, we extend ILLIXR's Switchboard *topics* [19] to support serialization and transmission over the network, which we call *networked topics*. Networked topics behave just like regular topics, except that they are initialized with hints that enable optimizations in the networking stack such as priority, ordering, packetization, etc. Since our encoding pipeline requires frames to be decoded in order of encoding, we implement a TCP network backend in Switchboard to guarantee an ordered stream. Although our case only transmits poses to the server and frames to the client, this enables Switchboard to flexibly transmit multiple topics as desired and could be used to enable the offloading of other computation in the future.

5 Evaluation Methodology

This section first presents the platforms we use and the baseline we compare against. We then describe the applications and metrics we use to evaluate our system.

5.1 Experimental Platform, Baseline, and Evaluated System Configurations

Headset display. We use the Valve Index [52] as our target display, powered by our system in various setups described below. The headset has two 1440×1600 displays with a refresh rate of up to 144 Hz. However, the 144 Hz mode is only available as an experimental option in the SteamVR runtime. Since most untethered VR headsets on the market run at 90 Hz, we also select 90 Hz as the target frame rate for both local and offloaded evaluations.

System under test. Since commercially available headsets are proprietary, we cannot run the XRGo runtime directly on them. Instead, we run the XRgo client-side runtime on an NVIDIA Jetson AGX Orin 64 GB Developer Kit [41], which is an embedded class system that approximates headset hardware. We use the 30 W power profile for the Jetson to emulate the thermal design power (TDP) of current headsets. To get a complete XR user experience, we connect the Jetson to the Valve Index headset, using the headset purely as a display. We run the XRgo server-side runtime on a high-end workstation equipped with an Intel Core i9-13900KF CPU and a discrete NVIDIA GeForce RTX 4090 GPU.

The server and client are connected with various network configurations described below. To evaluate the benefit of OW, we run the offloaded client-side configurations with both OpenWarp (referred to as OW) and TimeWarp (referred to as TW). To evaluate the benefit of offloading, we also test ILLIXR with TW running entirely on the Jetson with no offloading (referred to as Local).

Ideal baseline. We evaluate user experience with our various Jetson configurations by comparing against an ideal baseline. This baseline represents the best experience achievable with the headset tethered to a high-end workstation. We use the same workstation as the server in the XRGo configurations above. We run ILLIXR on this configuration and ensure that it can run at the target frame rate, while also using TW to reproject right before display (to minimize latency between render and display time). This setup represents the ideal quality of experience (QoE) to strive for.

Network configurations. In offloaded scenarios, we consider our client under the following set of live[4] and simulated (using tcconfig [17] with constant latency and no bandwidth constraints) network conditions (the server is always connected via Ethernet):

- Wi-Fi available in our lab, which provides approximately 500 Mbps for both upload and download bandwidth respectively, with an average 3.66 ms round-trip latency. However, there are occasional spikes of up to low-mid double digits due to varying network traffic. We compare this against a fixed constant network round-trip time of 20 ms.
- Trace-driven 5G emulation[5] following the approach from [48]. We collect time-varying latency and bandwidth traces using a Verizon 5G phone (Google Pixel 5) inside our lab that sends data to our workstation. The latency and bandwidth traces are collected separately. The traces last for 200 seconds (repeated indefinitely) and have 46.37 ± 9.87 ms (mean ± std) round-trip latency, 66.11 ± 0.79 Mbps upload bandwidth, and 225.23 ± 8.72 Mbps download bandwidth. We compare this against a fixed constant network round-trip time of 50 ms.

In total, each scene (application) we evaluate (described below) has 9 configurations: 1 for the local case and 8 across all network and reprojection permutations.

Other parameters: We match the headset display by rendering and encoding each eye buffer at 1440×1600 resolution. We encode at a total bitrate of 190 Mbps, a safe lower bound of the 5G trace (and WiFi) bandwidth. For TW, this entire bitrate can be allocated to encoding color. For OW, we partition this bitrate into 180 Mbps and 10 Mbps to encode color and depth respectively — we find that OW is relatively robust against heavily compressed depth.

For OW, we use a grid size of 360×400. We find that a neighborhood radius of 0.005 (with texture coordinates being normalized in $[0, 1]$) is appropriate for this grid size. We also overscan by 10% (Section 3.3) but use the same image resolution. This theoretically decreases the effective resolution of the transmitted image, but our user studies confirm that this is acceptable (Section 6.1).

5.2 Applications and Scenes

All scenes are exported from the Godot 4 Game Engine [3] with its Forward+ renderer. We target two different situations aiming to include diverse scene characteristics:

- **Bistro** [33], shown in Figure 6a, is an outdoor scene with primarily direct lighting, representing an architectural visualization with moderate graphics.
- **Spaceship** [20], shown in Figure 6b, is a semi-open scene with complex indirect lighting and geometry, representing a game environment with high-end graphics.

Importantly, neither application could meet the target frame rate on the Jetson in Local mode. We note that Unreal Engine 5 (UE5) can offer additional rendering features, but at the time of writing,

[4]Live refers to network conditions with time-varying latency and bandwidth, as in real networks. We used live Wi-Fi or 5G traces collected from actual 5G networks.
[5]We opted for trace-driven emulation instead of live 5G to reduce variability and ensure consistency across user tests, as the performance of live 5G networks can vary significantly at different times of the day (e.g., daytime vs. nighttime).

(a) Bistro scene **(b) Spaceship scene**

Figure 6: (a) The Bistro scene and (b) the Spaceship scene captured from a camera in the Godot Game Engine preview.

the Jetson Orin 64 GB does not support UE5 projects.[6] Using such applications could make it even more expensive to render on-device.

5.3 User Study on Quality of Experience

Since computed image metrics are not always representative of a user's actual quality-of-experience [49], we evaluate OW's perceptual quality with a user study. This study protocol is derived from prior work [21] and has Institutional Review Board approval.

5.3.1 Participant Recruitment and Demographic. We recruit 29 participants via email lists and direct contact. Participants are compensated with $10 for every 30 minutes, including any necessary breaks. 20 participants self-identify as male and 9 participants as female. Ages are 19-55 years old, with an average age of 26.38 ± 9.03 (mean ± std) years old. All participants have normal or corrected-to-normal vision. 8 participants report never interacting with XR, 16 report rarely interacting with XR (a few days a year), 4 report occasionally interacting with XR (a few days most months), and 1 report frequently interacting with XR (several days most weeks).

5.3.2 Experimental Protocol. Before any experiments, participants may freely use the baseline configuration to adjust the headset and familiarize themselves with the system. Here, they are placed in a warm-up scene, different from the two described in Section 5.2.

Once experiments begin, participants first explore the test scene presented through the baseline configuration. They are told that this is the ideal configuration that they will be asked to compare all test configurations against. They may request to revisit the baseline whenever they would like to refresh their point of reference (otherwise, it is not repeated to prevent fatigue). They are encouraged to make both rotational and translational movements, and are free to walk around the test space (a $\sim 4.5 \times 2.4$ meter rectangle).

After experiencing the baseline, the user begins trials of the test configurations described in Section 5.1, which are kept unknown to them. For each trial, the participant is asked to compare the test experience against the first baseline. To avoid bias, configurations are roughly counter-balanced by randomizing the scene order per user, the network condition per scene, and the reprojection algorithm per network condition. Participants may take breaks for as long as desired between trials to minimize fatigue, and may also choose to re-evaluate specific configurations if necessary after re-trying the baseline. Including breaks, each user spent around $60 - 90$ minutes to complete all 18 trials. The participant evaluates each trial with the following questionnaire [21], where Experience 1 is always the baseline while Experience 2 is the test configuration:

(1) Did you find any difference in the two XR experiences?
   (a) Yes, I can tell that they are different.
   (b) No, they are indistinguishable to me.
(2) Which experience did you like better?
   (a) Experience 1.
   (b) Experience 2.
(3) For the experience you ranked worse, how significant is the degradation in comparison to the better experience?
   (a) Only slightly worse or almost the same.
   (b) Worse but the experience is acceptable to me.
   (c) Much worse and not acceptable to me
(4) What contributed to the degradation you perceived (select one or more)? [7]
   (a) More jitters.
   (b) More lag.
   (c) More jerkiness.
   (d) More drift.
   (e) Lost tracking.
   (f) Other (please specify).
(5) Please share any additional comments or suggestions you may have regarding your experience.

Since we are interested in overall QoE, we only use responses to Questions 1, 2, and 3 to quantify ratings. Experience 2 delivering an indistinguishable experience or better (determined from Questions 1 and 2) receives a score of 4; if Experience 2 felt worse, the answer to Question 3 corresponds to a score of (a) $\rightarrow$ 3, (b) $\rightarrow$ 2, (c) $\rightarrow$ 1.

We do not factor Questions 4 or 5 into the rating value. Instead, we use Question 4 to potentially identify what may have affected user experience, but leave a rigorous evaluation of specific degradation causes to future work. We use Question 5 to understand unanticipated issues in our protocol, discussed in Section 6.1.4.

5.3.3 Statistical Analysis. We run a set of statistical tests to analyze the results of our user study. First, we apply the two-sample t-test and Wilcoxon Signed-Rank Test [56] to evaluate the average treatment effect. Then, we run Ordinal Logistic Regression (OLR) [15] to evaluate the effect size of each independent variable (reprojection algorithm, average network latency, and network variability).

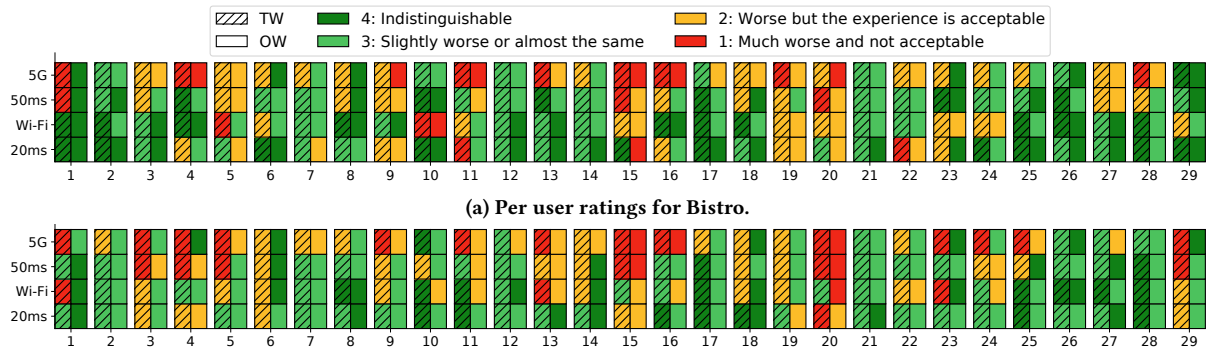
The two-sample t-test serves as a widely recognized reference for determining statistical significance. However, due to the ordinal nature of our dependent variable (user ratings), the t-test is not well-suited because 1) user rankings may not be normally distributed, 2) the population might not have equal variances, and 3) the real distance between two rankings may not be equal. The Wilcoxon test relaxes these assumptions and suits the ordinal user rankings.

We further employ OLR to explore the effect size of individual factors. OLR is suitable for modeling relationships where the dependent variable is ordinal and the intervals between them may not be equal. We employ the Brant test [8] to assess the proportional odds assumption, a prerequisite for OLR,[8] which assumes that the relationship between the predictors and the ordinal response variable is consistent across all outcome categories. The model estimates the log odds of the dependent variable being at or below a certain level

[6] https://forums.developer.nvidia.com/t/jetson-agx-orin-unreal-engin-vulkan-problem/277494

[7] Each artifact is explained in the footnotes of the questionnaire. Answers to this question are not factored into the quantitative rating.

[8] In our results, the p-values of the Brant test are all less than 0.05, confirming that the proportional odds assumption is met.



(a) Per user ratings for Bistro.

(b) Per user ratings for Spaceship.

Figure 7: Raw user study results with quantified and color-coded rating values from Section 5.3. Each user (x-axis) rated their experience with offloaded rendering on each network condition (y-axis) using both reprojection algorithms (paired bars, left is TW and right is OW). Parts (a) and (b) show results for Bistro and Spaceship respectively. Higher rating is better, with 4 (dark green) indicating that the test configuration is indistinguishable or better compared to the baseline, and 1 (red) indicating that the test configuration is unacceptable. For the local rendering configurations (not shown here), all users rated them as unacceptable, with the exception of user 28 who rated Bistro as worse but acceptable.

given the independent variables. We model average latency as a continuous variable and other independent variables as categorical. Network variability and reprojection are binary-encoded where 1 denotes live network and using OW for reprojection.

5.4 Power Measurements

We measure the total power consumption from all configurations described in Section 5.1, except for the simulated network latencies.

We use the Jetson Developer SDK's power measurement tool, `tegrastats`, to collect instantaneous voltage and current across power rails every 10 ms. The `GPU_SOC` rail measures GPU and SOC power, which includes various fixed-function hardware such as the video decoder; `CPU_CV` measures CPU power consumption as the CV cores are not utilized for our system; `I/O` measures the power supplied to display port, USB, and various other I/O functions; `DDR` measures the DDR memory power consumption.

To reduce variance in rendering power, we use a dataset to provide the trajectory rather than live experiments. We adopt the EuRoC MAV V1_02 trajectory [10]. This trajectory is replayed throughout all power trials, so the same poses are always rendered.

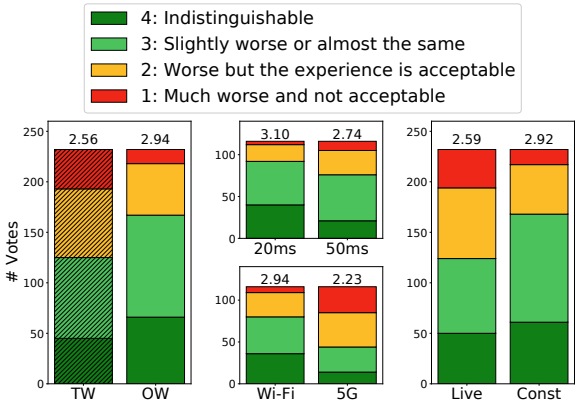
6 Results

This section presents the results of our user study (Section 6.1) and power measurements (Section 6.2).

6.1 User Study

Figure 7 presents the raw data from our user study, showing how each user (x-axis) rated their experience for each network condition (y-axis), each reprojection algorithm (left bar for TW and right bar for OW), and each scene (7a for Bistro and 7b for Spaceship). For better visualization, we use dark green, light green, yellow, and red colors to represent the ratings from best (4 for indistinguishable) to worst (1 for unacceptable).

We derive the following data from Figure 7 to inform our analysis and conclusions below. Figure 8 aggregates the number of votes for each rating to ascertain the overall impact of the reprojection



(a) Reprojection **(b) Latency** **(c) Variability**

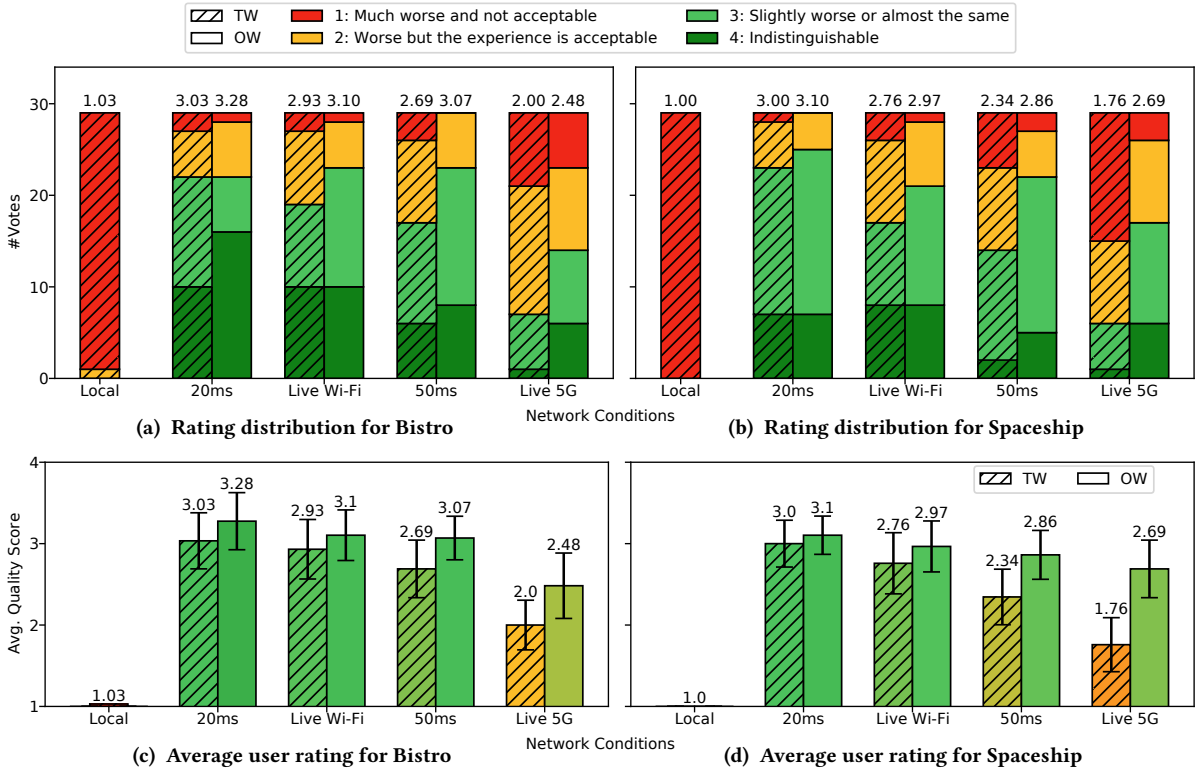
Figure 8: Aggregated votes by (a) reprojection, (b) network latencies, grouped by constant latencies on the top and live network on the bottom, and (c) network variability.

algorithm (Figure 8a), network latency (Figure 8b), and network variability (Figure 8c) on the user experience. Figures 9a and 9b aggregate the rating votes for each combination of reprojection algorithm, network latency, and network variability for the two scenes. Figures 9c and 9d show the corresponding average ratings with a 95% confidence interval. Table 1 shows results for the three statistical tests described in Section 5.3.3 to corroborate the statistical significance of our conclusions.

6.1.1 *Reprojection Algorithm.*

OW outperforms TW on average. Figure 8a shows the average user ratings for TW and OW as 2.56 and 2.94 respectively, aggregated across all experimental conditions. Overall, OW delivers an experience more similar to the baseline than TW. Table 1 shows almost-zero p-values for the t- and Wilcoxon tests, and a high OLR coefficient for OW, confirming statistical significance of our results.

Figure 8a also shows that, in our experiments, OW delivers more high-quality experiences and fewer unacceptable experiences compared to TW. For example, the dark/light green (high-quality)



(a) Rating distribution for Bistro

(b) Rating distribution for Spaceship

(c) Average user rating for Bistro

(d) Average user rating for Spaceship

Figure 9: Finer-grained rating distributions (a, b) and average ratings (c, d). 95% confidence intervals are plotted on the average ratings. For paired bars: left is TW, right is OW. The left column is Bistro and the right column is Spaceship. Higher is better.

	t-test p-value	Wilcoxon p-value	OLR coef	OLR p-value
Analysis for Aggregated Results in Figure 8				
OW vs. TW	0.000	0.000	0.7682	0.000
Avg. lat.			-0.0312	0.000
20ms vs. 50ms	0.001	0.000		
Wi-Fi vs. 5G	0.000	0.000		
Live vs. Const	0.000	0.000	-1.0135	0.000
Analysis for OW vs. TW at High Latency and Variability				
Lat. (5G + 50ms)	0.000	0.000		
Var. (Wi-Fi + 5G)	0.001	0.000		

Table 1: Statistical tests from Section 5.3.3. For the t- and Wilcoxon test, we only compare average latency under the same network variability condition (live or const). For OLR, we parameterize average latency as a continuous variable; bolded categorical variables are binary-encoded as 1. All p-values labeled 0.000 are <0.0005. For analysis of OW vs. TW at high latency and variability, data is aggregated by comparable latency (5G + 50ms) and variability (Wi-Fi + 5G).

portions constitute 72% of OW experiences vs. 54% for TW, while the red (unacceptable) part is only 6.0% for OW vs. 16.8% for TW.

6.1.2 *Latency.*

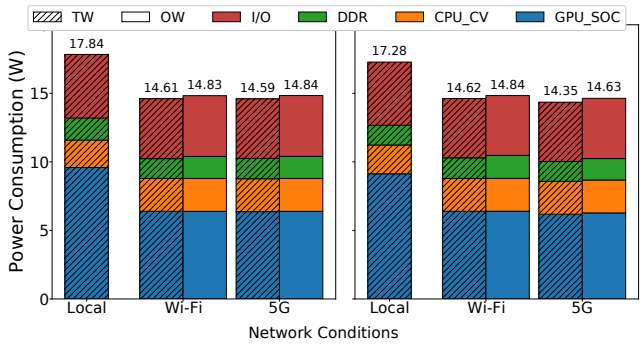
Higher latencies degrade user experience. Figure 8b examines the impact of network latency on user experience. The top compares constant latency networks at 20ms and 50ms latency, while the bottom compares live networks with an average latency of 3.66 ms for Wi-Fi and 46.37 ms for 5G. In both cases, higher latency shows a worse average rating — 2.74 vs. 3.10 for constant 50 ms vs. 20 ms and 2.23 vs. 2.94 for live 5G vs. Wi-Fi. Table 1 shows that these differences are statistically significant via the t- and Wilcoxon tests, with an OLR coefficient of −0.0311 that predicts, with high statistical significance, a negative relationship between QoE and latency.

OW outperforms TW even at higher latencies. Figure 9 shows that even though higher latencies negatively impact experience, OW outperforms TW at these latencies for both constant latency and live networks, providing a better-than-acceptable experience on average. At a constant 50ms latency, OW has average ratings of 3.07 and 2.86 for Bistro and Spaceship respectively as opposed to 2.69 and 2.34 for TW. For 5G, OW has average ratings of 2.48 and 2.69 for the two scenes compared to 2.00 and 1.76 for TW. Table 1 supports the statistical significance when comparing OW to TW under high latencies (5G + 50 ms).

6.1.3 *Live Network Conditions.*

Network variability degrades user experience. Figure 8c shows that on average, the live networks have a lower rating than the constant latency networks (2.59 vs. 2.92). Table 1 shows this difference is statistically significant, and the OLR coefficient confirms a strong negative relationship between QoE and network variability.[9]

[9]A variability comparison between live Wi-Fi and 20 ms networks is nuanced due to their different average latencies. Although Wi-Fi average latency is only 3.66 ms, Figure 9 shows that it still has slightly worse QoE than 20 ms constant latency. We intuit that network variability degrades the QoE, outweighing the lower latency. 5G and 50 ms networks, however, can be compared directly and Figure 9 shows that 5G has a consistently lower rating. A t-test and Wilcoxon test on 5G vs. 50 ms aggregated



(a) Power for Bistro **(b) Power for Spaceship**

Figure 10: Measured power on (a) Bistro and (b) Spaceship with local rendering, 5G offloading, and Wi-Fi offloading. For paired bars: left is TW, right is OW. Lower is better.

OW outperforms TW at high network variability. Figure 9 shows that OW has higher average ratings than TW under 5G and Wi-Fi for both scenes. For example, 5G ratings are 2.48 for OW vs. 2.00 for TW for Bistro. Corresponding values for Spaceship are 2.69 vs. 1.76. Wi-Fi also demonstrates higher ratings for OW over TW, but the difference is less pronounced. The tests in Table 1 establish the statistical significance that OW outperforms TW under network variability (Wi-Fi + 5G). When latency spikes, OW reveals more obvious stretching artifacts while TW users feel more abrupt movements due to high lag. Our results suggest that users, on average, prefer the OW artifacts over TW's staggered movement.

6.1.4 Constraints of Our Study. This section discusses the constraints of our study and potential future work. As detailed in Section 5.3, we asked the users to perform multiple trials with different configurations. Although the stimuli presentation schedule may introduce dependencies between trials, we aimed to mitigate this by randomizing presentation order per user and instructing them to only compare against the baseline. Due to the number of trials per user (16 excluding baseline), we did not schedule additional hidden references to prevent user fatigue. We also compensated participants based on time to encourage more careful evaluations.

We included users with diverse prior XR expertise but did not distinguish them by this demographic. Furthermore, due to the diversity of elements in the scenes, we also did not focus on scene dependency. We leave the study of these factors to future work.

Nonetheless, to the best of our knowledge, our study is the first published end-to-end user study that extensively (with 522 total trials) investigates reprojection algorithms for XR offloading.

6.2 Power Consumption

We report the end-to-end power measurements for both scenes in Figure 10. The Jetson measurements between 5G and Wi-Fi are roughly equal, so we discuss the measurements reported from the 5G experiments and note that similar observations apply to Wi-Fi.

In a locally rendered system with a 30 W TDP profile on the Jetson, both scenes fail to meet the target frame rate. Bistro averaged 20 application frames per second (FPS) and 45 reprojection FPS, and Spaceship averaged 8 FPS and 32 FPS respectively. All but one user rated this unacceptable (Figure 9). Furthermore, as Figure 10 shows, this unacceptable experience consumed around 3W more power than the offloading client running at full 90FPS.

We also benchmarked the application at 60 W TDP, the maximum on an Nvidia AGX Orin. While Bistro was able to make target frame rate most of the time, Spaceship failed to meet the target with 32 application FPS and 65 reprojection FPS on average. At 60 W TDP, Spaceship consumed 44.41 W and Bistro 44.16 W, which is around 30 W more than the thin client for rendering offload. Even with the higher energy budget of 60 W, from the authors' experience, the QoE of Spaceship running locally is still lower than that of offloading due to low application and reprojection frame rates.

Although with offloading, the client must communicate with the server, decode transmitted frames, and apply a potentially more complex reprojection at the target frame rate, we find this overhead is much smaller than the power required to render the scene locally at an acceptable frame rate. Figure 10 shows the power for the client with rendering offloaded is about 15W. Further, the power overhead of OW decoding additional depth images and rendering a reprojection mesh is small; when offloading on 5G, OW only consumes 1.81% more power than TW.

7 Conclusions

This paper presents XRgo, an end-to-end open-source XR runtime that enables application-transparent and OpenXR-compliant offloaded rendering with OpenWarp, a lightweight client-side 6-DoF reprojection algorithm. XRgo enables us to evaluate both the power and quality tradeoffs of OW against TW in an offloaded environment. Our user study demonstrates that OW's translational latency compensation significantly improves the experience compared to TW in an offloaded setting. Thus, the combination of offloading rendering with asynchronous OW delivers high-quality 3D graphics at low power and low effective latency.

The modularity of our system enables future studies in asynchronous reprojection techniques, as well as video codecs, streaming, and network protocols, in an end-to-end system setting. Our evaluation has considered static scenes, and we expect OW to have limited effectiveness with moving objects. A promising future direction is to enhance XRgo with a combination of OW and other compensation techniques, specifically those for dynamic and interactive scenes, to enable additive benefits (Section 2.2). In the future, given multiple configurations and compensation techniques with different quality-power-performance tradeoffs, a policy could be employed to dynamically choose when to offload and which configuration to use based on given resource constraints.

8 Acknowledgments

We thank the anonymous reviewers and our shepherd, Hans-Jürgen Zepernick, for their constructive feedback. We also thank Professor Han Zhao for valuable discussions of statistical analysis for our user study. This work is supported in part by the National Science Foundation under grants 2120464 and 2217144, the IBM-Illinois Discovery Accelerator Institute (IIDAI), and gifts from Cisco and T-Mobile.

across the two scenes and reprojection algorithms (not shown in Table 1) provided substantial statistical significance with p-values < 0.0001.

References

[1] 2021. ILLIXR Illinois Extended Reality testbed. https://illixr.org.

[2] 2024. Air Light VR (ALVR). https://github.com/alvr-org/ALVR.

[3] 2024. Godot Engine. https://godotengine.org/.

[4] Ahmad Alhilal, Ze Wu, Yuk Hang Tsui, and Pan Hui. 2024. FovOptix: Human Vision-Compatible Video Encoding and Adaptive Streaming in VR Cloud Gaming. In *Proceedings of the 15th ACM Multimedia Systems Conference* (Bari, Italy) *(MMSys'24)*. Association for Computing Machinery, New York, NY, USA, 67–77. https://doi.org/10.1145/3625468.3647612

[5] AMD. 2023. AMD FidelityFX Super Resolution 3. https://gpuopen.com/fidelityfx-super-resolution-3/.

[6] Michael Antonov. 2015. Asynchronous Timewarp Examined. https://developers.meta.com/horizon/blog/asynchronous-timewarp-examined/.

[7] Dean Beeler, Ed Hutchins, and Paul Pedriana. 2016. Asychrononus Spacewarp. https://developer.oculus.com/blog/asynchronous-spacewarp/.

[8] Rollin Brant. 1990. Assessing Proportionality in the Proportional Odds Model for Ordinal Logistic Regression. *Biometrics* 46, 4 (1990), 1171–1178. http://www.jstor.org/stable/2532457

[9] ITU-R Recommendation BT. 2015. *Parameter values for the HDTV standards for production and international programme exchange*. Technical Report. International Telecommunication Union.

[10] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. 2016. The EuRoC Micro Aerial Vehicle Datasets. *The International Journal of Robotics Research* 35, 10 (2016), 1157–1163. https://doi.org/10.1177/0278364915620033 arXiv:https://doi.org/10.1177/0278364915620033

[11] Collabora. 2024. Monado - OpenXR Runtime. https://monado.dev/.

[12] Mohammed S. Elbamby, Cristina Perfecto, Mehdi Bennis, and Klaus Doppler. 2018. Toward Low-Latency and Ultra-Reliable Virtual Reality. *IEEE Network* 32, 2 (2018), 78–84. https://doi.org/10.1109/MNET.2018.1700268

[13] Sinclair Finn. 2020. *Spatio-temporal reprojection for virtual and augmented reality applications*. Bachelor's Thesis. University of Illinois at Urbana-Champaign. https://hdl.handle.net/2142/109178

[14] Jie Guo, Xihao Fu, Liqiang Lin, Hengjun Ma, Yanwen Guo, Shiqiu Liu, and Ling-Qi Yan. 2021. ExtraNet: Real-time Extrapolated Rendering for Low-latency Temporal Supersampling. *ACM Trans. Graph.* 40, 6, Article 278 (2021), 16 pages. https://doi.org/10.1145/3478513.3480531

[15] Frank E. Harrell. 2015. *Ordinal Logistic Regression*. Springer International Publishing, Cham, 311–325. https://doi.org/10.1007/978-3-319-19425-7_13

[16] Jozef Hladky, Michael Stengel, Nicholas Vining, Bernhard Kerbl, Hans-Peter Seidel, and Markus Steinberger. 2022. QuadStream: A Quad-Based Scene Streaming Architecture for Novel Viewpoint Reconstruction. *ACM Trans. Graph.* 41, 6, Article 233 (2022), 13 pages. https://doi.org/10.1145/3550454.3555524

[17] Tsuyoshi Hombashi. 2024. tcconfig. https://github.com/thombashi/tcconfig.

[18] Muhammad Huzaifa, Rishi Desai, Samuel Grayson, Xutao Jiang, Ying Jing, Jae Lee, Fang Lu, Yihan Pang, Joseph Ravichandran, Finn Sinclair, Boyuan Tian, Hengzhi Yuan, Jeffrey Zhang, and Sarita V. Adve. 2021. ILLIXR: Enabling End-to-End Extended Reality Research. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*. 24–38. https://doi.org/10.1109/IISWC53511.2021.00014

[19] Muhammad Huzaifa, Rishi Desai, Samuel Grayson, Xutao Jiang, Ying Jing, Jae Lee, Fang Lu, Yihan Pang, Joseph Ravichandran, Finn Sinclair, Boyuan Tian, Hengzhi Yuan, Jeffrey Zhang, and Sarita V. Adve. 2022. ILLIXR: An Open Testbed to Enable Extended Reality Systems Research. *IEEE Micro* 42, 4 (2022), 97–106. https://doi.org/10.1109/MM.2022.3161018

[20] Jaanus Jaggo. 2023. Abandoned Spaceship Demo. https://godotengine.org/asset-library/asset/1733.

[21] Qinjun Jiang, Yihan Pang, William Sentosa, Steven Gao, Muhammad Huzaifa, Jeffrey Zhang, Javier Perez-Ramirez, Dibakar Das, David Gonzalez-Aguirre, Brighten Godfrey, and Sarita Adve. 2025. RemoteVIO: Offloading Head Tracking in an End-to-End XR System. In *Proceedings of the 16th ACM Multimedia Systems Conference* (Stellenbosch, South Africa) *(MMSys'25)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3712676.3714442

[22] Teemu Kämäräinen, Matti Siekkinen, Jukka Eerikäinen, and Antti Ylä-Jääski. 2018. CloudVR: Cloud Accelerated Interactive Mobile Virtual Reality. In *Proceedings of the 26th ACM International Conference on Multimedia* (Seoul, Republic of Korea) *(MM'18)*. Association for Computing Machinery, New York, NY, USA, 1181–1189. https://doi.org/10.1145/3240508.3240620

[23] David Kanter. 2015. *Graphics Processing Requirements for Enabling Immersive VR*. Technical Report.

[24] Zhihui Ke, Xiaobo Zhou, Dadong Jiang, Hao Yan, and Tie Qiu. 2023. CollabVr: Reprojection-Based Edge-Client Collaborative Rendering for Real-Time High-Quality Mobile Virtual Reality. In *2023 IEEE Real-Time Systems Symposium (RTSS)*. 304–316. https://doi.org/10.1109/RTSS59052.2023.00034

[25] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (2023). https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[26] Emmett Kilgariff, Henry Moreton, Nick Stam, and Brandon Bell. 2018. NVIDIA Turing Architecture In-Depth. https://developer.nvidia.com/blog/nvidia-turing-architecture-in-depth/.

[27] Janghun Kim and Sungkil Lee. 2023. Potentially Visible Hidden-Volume Rendering for Multi-View Warping. *ACM Trans. Graph.* 42, 4, Article 86 (2023), 11 pages. https://doi.org/10.1145/3592108

[28] Zeqi Lai, Y. Charlie Hu, Yong Cui, Linhui Sun, and Ningwei Dai. 2017. Furion: Engineering High-Quality Immersive Virtual Reality on Today's Mobile Devices. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking* (Snowbird, Utah, USA) *(MobiCom'17)*. Association for Computing Machinery, New York, NY, USA, 409–421. https://doi.org/10.1145/3117811.3117815

[29] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. 2015. Outatime: Using Speculation to Enable Low-Latency Continuous Interaction for Mobile Cloud Gaming. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services* (Florence, Italy) *(MobiSys'15)*. Association for Computing Machinery, New York, NY, USA, 151–165. https://doi.org/10.1145/2742647.2742656

[30] Luyang Liu, Ruiguang Zhong, Wuyang Zhang, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Marco Gruteser. 2018. Cutting the Cord: Designing a High-quality Untethered VR System with Low Latency Remote Rendering. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services* (Munich, Germany) *(MobiSys'18)*. Association for Computing Machinery, New York, NY, USA, 68–80. https://doi.org/10.1145/3210240.3210313

[31] Xing Liu, Christina Vlachou, Feng Qian, Chendong Wang, and Kyu-Han Kim. 2020. Firefly: Untethered Multi-user VR for Commodity Mobile Devices. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 943–957. https://www.usenix.org/conference/atc20/presentation/liu-xing

[32] Edward Lu, Sagar Bharadwaj, Mallesham Dasari, Connor Smith, Srinivasan Seshan, and Anthony Rowe. 2023. RenderFusion: Balancing Local and Remote Rendering for Interactive 3D Scenes. In *2023 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 312–321. https://doi.org/10.1109/ISMAR59233.2023.00046

[33] Amazon Lumberyard. 2017. Amazon Lumberyard Bistro, Open Research Content Archive (ORCA). http://developer.nvidia.com/orca/amazon-lumberyard-bistro

[34] William R. Mark, Leonard McMillan, and Gary Bishop. 1997. Post-rendering 3D Warping. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics* (Providence, Rhode Island, USA) *(I3D'97)*. Association for Computing Machinery, New York, NY, USA, 7–ff. https://doi.org/10.1145/253284.253292

[35] Leonard McMillan and Gary Bishop. 1995. Head-tracked Stereoscopic Display Using Image Warping. In *Stereoscopic Displays and Virtual Reality Systems II*, Scott S. Fisher, John O. Merritt, and Mark T. Bolas (Eds.), Vol. 2409. International Society for Optics and Photonics, SPIE, 21 – 30. https://doi.org/10.1117/12.205865

[36] Jiayi Meng, Sibendu Paul, and Y. Charlie Hu. 2020. Coterie: Exploiting Frame Similarity to Enable High-Quality Multiplayer VR on Commodity Mobile Devices. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS'20)*. Association for Computing Machinery, New York, NY, USA, 923–937. https://doi.org/10.1145/3373376.3378516

[37] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2021. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *Commun. ACM* 65, 1 (2021), 99–106. https://doi.org/10.1145/3503250

[38] Diego Nehab, Pedro V. Sander, Jason Lawrence, Natalya Tatarchuk, and John R. Isidoro. 2007. Accelerating Real-Time Shading with Reverse Reprojection Caching. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware* (San Diego, California, USA) *(GH'07)*. Eurographics Association, Goslar, DEU, 25–35.

[39] NVIDIA. 2022. NVIDIA DLSS 3. https://www.nvidia.com/en-us/geforce/technologies/dlss/.

[40] NVIDIA. 2024. Jetson Linux API Reference. https://docs.nvidia.com/jetson/l4t-multimedia/.

[41] NVIDIA. 2024. NVIDIA Jetson Orin. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/.

[42] Edward Peek, Christof Lutteroth, and Burkhard Wünsche. 2013. More for Less: Fast Image Warping for Improving the Appearance of Head Tracking on HMDs. In *2013 28th International Conference on Image and Vision Computing New Zealand (IVCNZ 2013)*. 41–46. https://doi.org/10.1109/IVCNZ.2013.6726990

[43] PlutoVR and Collabora. 2024. ElectricMaple. https://github.com/PlutoVR/electric-maple.

[44] Bernhard Reinert, Johannes Kopf, Tobias Ritschel, Eduardo Cuervo, David Chu, and Hans-Peter Seidel. 2016. Proxy-guided Image-based Rendering for Mobile Devices. *Comput. Graph. Forum* 35, 7, 353–362.

[45] Gernot Schaufler. 1996. Exploiting Frame-to-frame Coherence in a Virtual Reality System. In *Proceedings of the IEEE 1996 Virtual Reality Annual International Symposium*. 95–102. https://doi.org/10.1109/VRAIS.1996.490516

[46] Daniel Scherzer, Lei Yang, Oliver Mattausch, Diego Nehab, Pedro V. Sander, Michael Wimmer, and Elmar Eisemann. 2012. Temporal Coherence Methods in Real-Time Rendering. *Computer Graphics Forum* 31, 8 (2012), 2378–2408.

https://doi.org/10.1111/j.1467-8659.2012.03075.x

[47] Andre Schollmeyer, Simon Schneegans, Stephan Beck, Anthony Steed, and Bernd Froehlich. 2017. Efficient Hybrid Image Warping for High Frame-Rate Stereoscopic Rendering. *IEEE Transactions on Visualization and Computer Graphics* 23, 4 (2017), 1332–1341. https://doi.org/10.1109/TVCG.2017.2657078

[48] William Sentosa, Balakrishnan Chandrasekaran, P. Brighten Godfrey, Haitham Hassanieh, and Bruce Maggs. 2023. DChannel: Accelerating Mobile Applications With Parallel High-bandwidth and Low-latency Channels. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 419–436. https://www.usenix.org/conference/nsdi23/presentation/sentosa

[49] Rahul Singh, Muhammad Huzaifa, Jeffrey Liu, Anjul Patney, Hashim Sharif, Yifan Zhao, and Sarita Adve. 2023. Power, Performance, and Image Quality Tradeoffs in Foveated Rendering. In *2023 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*. 205–214. https://doi.org/10.1109/VR55154.2023.00036

[50] Ferdi Smit, Robert van Liere, Stephan Beck, and Bernd Froehlich. 2009. An Image-Warping Architecture for VR: Low Latency versus Image Quality. In *2009 IEEE Virtual Reality Conference*. 27–34. https://doi.org/10.1109/VR.2009.4810995

[51] FFmpeg team. 2024. FFmpeg. https://ffmpeg.org/.

[52] Valve. 2024. Valve Index Headset. https://www.valvesoftware.com/en/index/headset.

[53] J. M. P. van Waveren. 2016. The Asynchronous Time Warp for Virtual Reality on Consumer Hardware. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology* (Munich, Germany) *(VRST'16)*. Association for Computing Machinery, New York, NY, USA, 37–46. https://doi.org/10.1145/2993369.2993375

[54] Daniel Wagner. 2018. Motion to Photon Latency in Mobile AR and VR. https://medium.com/@DAQRI/motion-to-photon-latency-in-mobile-ar-and-vr-99f82c480926.

[55] Bruce Walter, George Drettakis, and Steven Parker. 1999. Interactive Rendering using the Render Cache. In *Eurographics Workshop on Rendering*, Dani Lischinski and Greg Ward Larson (Eds.). The Eurographics Association. https://doi.org//10.2312/EGWR/EGWR99/019-030

[56] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83. http://www.jstor.org/stable/3001968

[57] Songyin Wu, Sungye Kim, Zheng Zeng, Deepak Vembar, Sangeeta Jha, Anton Kaplanyan, and Ling-Qi Yan. 2023. ExtraSS: A Framework for Joint Spatial Super Sampling and Frame Extrapolation. In *SIGGRAPH Asia 2023 Conference Papers* (Sydney, NSW, Australia) *(SA'23)*. Association for Computing Machinery, New York, NY, USA, Article 92, 11 pages. https://doi.org/10.1145/3610548.3618224

[58] Lei Yang, Yu-Chiu Tse, Pedro V. Sander, Jason Lawrence, Diego Nehab, Hugues Hoppe, and Clara L. Wilkins. 2011. Image-based Bidirectional Scene Reprojection. In *Proceedings of the 2011 SIGGRAPH Asia Conference* (Hong Kong, China) *(SA'11)*. Association for Computing Machinery, New York, NY, USA, Article 150, 10 pages. https://doi.org/10.1145/2024156.2024184

[59] Sipeng Yang, Qingchuan Zhu, Junhao Zhuge, Qiang Qiu, Chen Li, Yuzhong Yan, Huihui Xu, Ling-Qi Yan, and Xiaogang Jin. 2024. Mob-FGSR: Frame Generation and Super Resolution for Mobile Real-Time Rendering. In *ACM SIGGRAPH 2024 Conference Papers* (Denver, Colorado, USA) *(SIGGRAPH'24)*. Association for Computing Machinery, New York, NY, USA, Article 64, 11 pages. https://doi.org/10.1145/3641519.3657424