ILLIXR: Enabling End-to-End Extended Reality Research

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
University of Illinois at Urbana-Champaign
illixr@cs.illinois.edu

***Abstract*—An increasing number of edge systems have large computational demands, stringent resource constraints, and end-to-end quality-driven goodness metrics. Architects have embraced domain-specific accelerators to meet the demands of such systems. We make the case for research that shifts emphasis from domain-specific accelerators to domain-specific *systems*, with a consequent shift from evaluations using benchmarks that are collections of independent applications to those using testbeds that are full integrated systems. We describe extended reality (XR) as an exciting domain motivating such domain-specific systems research, but hampered by the lack of an end-to-end evaluation testbed.**

We present ILLIXR (Illinois Extended Reality testbed), the first fully open source XR system and research testbed. ILLIXR enables system innovations with end-to-end co-designed hardware, compiler, OS, and algorithm, and driven by end-user perceived quality-of-experience (QoE) metrics. Using ILLIXR, we perform the first comprehensive quantitative analysis of performance, power, and QoE for a complete XR system and its individual components. We describe several implications of our results that propel new directions in architecture, systems, and algorithm research for domain-specific systems in general, and XR in particular, all enabled by ILLIXR.

I. Introduction

Recent years have seen the convergence of multiple disruptive trends to fundamentally change computer systems: (1) With the end of Dennard scaling and Moore's law, application-driven specialization has emerged as a key architectural technique to meet the requirements of emerging applications, (2) computing and data availability have reached an inflection point that is enabling a number of new application domains, and (3) these applications are increasingly deployed on resource-constrained edge devices, where they interface directly with the end-user and the physical world.

In response to these trends, our research conferences have seen an explosion of papers on efficient accelerators, many focused on machine learning. To truly achieve the promise of efficient domain-specific edge computing, however, will require architects to broaden their portfolio from specialization for individual accelerators for individual applications to specialization for domain-specific *systems*. Such systems may consist of multiple interacting sub-domains (or applications), requiring multiple accelerators that interact with each other in myriad ways to collectively meet end-user demands. Programming languages and OS researchers must also grapple with the heterogeneity of such domain-specific systems, developing scalable methods for compilation, scheduling, and resource management. Furthermore, it is likely that meeting the end-user quality demands of such systems will require co-designing the hardware, compiler, and OS along with the application [1], [2].

Case for Extended Reality as a driving domain: The following observations make the case that virtual, augmented, and mixed reality, collectively referred to as extended reality (XR),[1] is a rich domain that can propel research needed for this era of end-to-end quality-driven, resource-constrained, co-designed domain-specific edge systems:

(1) Pervasive: XR will transform many aspects of our lives, including teaching, science, medicine, entertainment, and more. Indeed, XR is envisioned to be the next interface for most of computing [3]–[5].

(2) Challenging demands: While XR systems exist today, they are far from providing a tetherless experience approaching perceptual abilities of humans. There is a gap of several orders of magnitude between what is needed and achievable in performance, power, and usability, giving systems researchers a potentially rich space to innovate. Table I summarizes various system-level quality related metrics for state-of-the-art XR devices and the aspiration for ideal future devices.

(3) Multiple and diverse components: XR involves several diverse sub-domains — computer vision, robotics, graphics, machine learning, optics, audio, and video — making it challenging to design a system that executes each one well while respecting the resource constraints.

(4) Full-stack implications: The combination of real-time constraints, complex interacting pipelines, and ever-changing algorithms creates a need for full stack optimizations involving the hardware, compiler, operating system, and algorithm [2].

(5) Flexible accuracy for end-to-end user experience: The end user being a human with limited perception enables a rich space of accuracy-aware resource trade-offs, but requires the ability to quantify impact on end-to-end experience.

Case for an XR system testbed: A key obstacle to architecture and systems research for XR is that there are no open source benchmarks providing the entire XR workflow to drive such research. While there exist open source codes for some individual components of the XR workflow (typically developed by domain researchers), there is no integrated suite that enables researching an XR *system*. As we move from the era of general-purpose, homogeneous cores on chip to domain-specific, heterogeneous *system*-on-chip architectures, benchmarks need to follow the same trajectory. While previous benchmarks comprising of suites of independent applications (e.g., Parsec [6], Rodinia [7], SPEC [8], [9], SPLASH [10]–[12], and more) sufficed to evaluate general-purpose single- and multicore architectures, there is now a need for a *full-system-benchmark* methodology, better viewed as a *full system testbed*, to design and evaluate system-on-chip architectures. Such a methodology must bring together the diversity of components that will interact with each other in the domain-specific system and also be flexible and extensible to accept future new components.

An XR full-system-benchmark or testbed will continue to enable traditional research for accelerating a given XR component with conventional metrics such as power, performance, and area for that component, but will additionally allow evaluations for the end-to-end impact on the system. More importantly, the integrated system will enable new research that co-designs acceleration of its multiple, diverse, and demanding components across the full stack, driven by end-to-end user experience.

Challenges and contributions: This paper presents ILLIXR (Illinois Extended Reality testbed), the first fully open-source XR system and testbed; the first detailed quantitative characterization of performance, power, and quality-of-experience (QoE) metrics for a complete XR system on desktop and embedded class machines; and several resulting future directions for architecture and systems research that are enabled by ILLIXR.

There were two broad challenges in the development of ILLIXR.

[1] Virtual reality (VR) immerses the user in a completely digital environment. Augmented Reality (AR) enhances the user's real world with overlaid digital content. Mixed reality (MR) goes beyond AR in enabling the user to interact with virtual objects in their real world.

First, ILLIXR required expertise in a large number of sub-domains (e.g., computer vision, robotics, graphics, optics, and audio). We consulted with many academic and industry experts in these sub-domains and XR systems (Section VIII). Through these discussions, we identified a representative XR workflow with state-of-the-art algorithms and open source codes for its constituent components, which we integrated into the ILLIXR testbed. ILLIXR has now been vetted through presentations to several XR industry and academic groups and events (e.g., [13]) and is the basis of an industry-backed consortium to enable XR research and standardize XR systems benchmarking [14].

Second, until recently, commercial XR devices had proprietary interfaces. For example, the interface between an Oculus head mounted device (HMD) runtime and the Unity or Unreal game engines that run on the HMD has been closed as are the interfaces between the different components within the HMD runtime. OpenXR [15], an open standard, was released in July 2019 to partly address this problem by providing a standard for XR device runtimes to interface with the applications that run on them. ILLIXR leverages Monado [16], an open implementation of OpenXR, to provide an OpenXR compliant XR system. However, OpenXR is still evolving, it does not address the problem of interfaces between the different components within an XR system, and there were limited OpenXR compliant applications (e.g., game engines and games) during the development of ILLIXR. Nevertheless, we are able to report results with sophisticated applications running on ILLIXR, and the rapidly growing popularity of OpenXR opens up a rich space of applications that ILLIXR can support to drive future research.

Specifically, this work makes the following contributions.

(1) We develop ILLIXR, the first fully open-source XR system and research testbed, consisting of a representative XR workflow (i) with state-of-the-art components, (ii) integrated with a modular and extensible multithreaded runtime that easily allows new components (or new implementations of existing components) to be added, (iii) providing an OpenXR compliant interface to XR applications (e.g., game engines), and (iv) with the ability to report (and trade-off) end-to-end quality of experience (QoE) metrics. The ILLIXR components represent the sub-domains of robotics (odometry or SLAM), computer vision (scene reconstruction), machine learning (eye tracking), image processing (camera), graphics (reprojection), optics (lens distortion, chromatic abberation correction), audio (3D audio encoding and decoding), and displays (holographic displays).

(2) We present the first detailed quantitative characterization of performance, power, and QoE metrics for a complete XR system. We conduct our analysis on desktop and embedded class machines with CPUs and GPUs, driven by a game engine running representative VR and AR applications. Overall, we find that current systems are far from the needs of future devices, making a case for efficiency through techniques such as specialization, codesign, and approximation.

(3) Our system-level analysis shows a wide variation in the resource utilization of different components. Although components with high resource utilization should clearly be targeted for optimization, components with relatively low utilization are also critical due to their impact on QoE.

(4) Power breakdowns show that CPUs, GPUs, and memories are only part of the power consumption. The rest of the SoC and system logic, including data movement for displays and sensors is a major component as well, motivating technologies such as on-sensor computing.

(5) XR components are diverse in their use of CPU, GPU compute, and GPU graphics, and exhibit a range of IPC and system bottlenecks. Analyzing their compute and memory characteristics, we find a variety of patterns and subtasks, with none dominating. The number and diversity of these patterns pose a research question for the granularity at which accelerators should be designed and whether and how they should be shared among different components. These observations motivate research in automated tools to identify acceleratable primitives, architectures for communication between accelerators, and accelerator software interfaces and programming models.

TABLE I: Ideal requirements of VR and AR vs. state-of-the-art devices, Varjo VR-3 for VR and Microsoft HoloLens 2 for AR. We identified the values of the various aspirational metrics through an extensive survey of the literature [3], [17], [18]. VR devices are typically larger and so afford more power and thermal headroom. The Varjo VR-3 offloads most of its work to an attached server, so its power and area values are not meaningful. The ideal case requirements for power, area, and weight are based on current devices which are considered close to ideal in these (but not other) respects – Snapdragon 835 in the Oculus Quest VR headset and APQ8009w in the North Focals small AR glasses.

Metric	Varjo VR-3 [19]	Ideal VR [17], [20]	Microsoft HoloLens 2	Ideal AR [17], [20], [21]
Resolution (MPixels)	15.7	200	4.4 [22]	200
Field-of-view (Degrees)	115	Full: 165×175 Stereo: 120×135	52 diagonal [23], [24]	Full: 165×175 Stereo: 120×135
Refresh rate (Hz)	90	90 – 144	120 [25]	90 – 144
Motion-to-photon latency (ms)	< 20	< 20	< 9 [26]	< 5
Power (W)	N/A	1 – 2	> 7 [27]–[29]	0.1 – 0.2
Silicon area (mm^2)	N/A	100 – 200	> 173 [27], [30]	< 100
Weight (grams)	944	100 – 200	566 [22]	10s

(6) Most components exhibit significant variability in per-frame execution time due to input-dependence or resource contention, thereby making it challenging to schedule and allocate shared resources. Further, a large number of system parameters need to be tuned for an optimal XR experience with different resource usage trade-offs. The current process is mostly ad hoc and exacerbated by the above variability. These observations motivate research on QoE-driven resource management, scheduling, and approximation.

Overall, this work provides the architecture, systems, and XR algorithms research community with a one-of-a-kind infrastructure and foundational analyses to enable new research within each layer of the system stack or co-designed across the stack, impacting a variety of domain-specific design methodologies in general and for XR in particular.

ILLIXR is fully open source and is designed to be extensible so that the broader community can easily contribute new components and features and new implementations of current ones. The current ILLIXR code repository [14] consists of over 100K lines of code for the main components, another 80K for the OpenXR interface implementation [16], several XR applications and an open source game engine, and continues to grow with new components and algorithms. The appendix provides information about the artifacts used in this paper (evaluated and accepted for the availability criterion). We have formed the ILLIXR consortium to manage the ongoing development of ILLIXR, including community contributions and further standardization of XR benchmarking [14].

II. The ILLIXR System

Figure 1 presents the ILLIXR system. ILLIXR captures state-of-the-art components that belong to an XR runtime such as Oculus VR (OVR) and are shipped with an XR headset. Applications, often built using a game engine, are separate from the XR system or runtime, and interface with it using the runtime's API. For example, a game developed on the Unity game engine for an Oculus headset runs on top of OVR, querying information from OVR and submitting rendered frames to it using the OVR API (recently replaced by the OpenXR API). Analogously, applications interface with ILLIXR using the OpenXR API [15]. Consequently, ILLIXR does not include components from the application, but captures the performance impact of the application running on ILLIXR. We collectively refer to all application-level tasks (e.g., button-input handling, scene simulation, physics, rendering, etc.) as "application" in the rest of this paper.

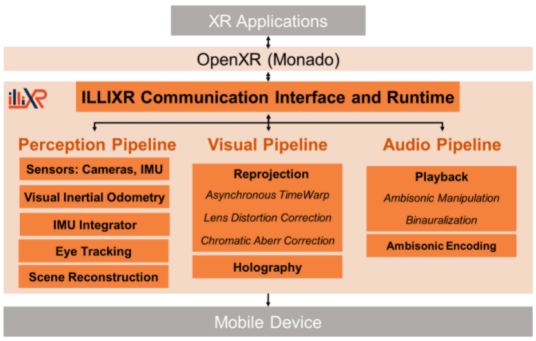


Fig. 1: The ILLIXR system and its relation to XR applications, the OpenXR interface, and mobile platforms.

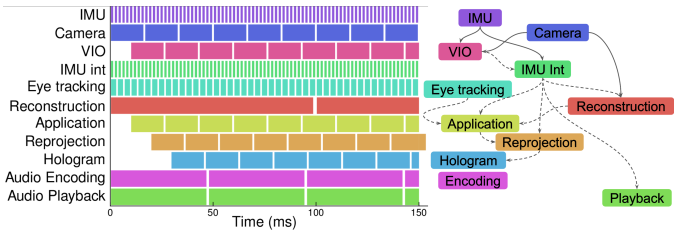


Fig. 2: Interactions between ILLIXR components. The left part shows an ideal ILLIXR execution schedule and the right part shows inter-component dependencies that the ILLIXR scheduler must maintain (§II-B). Solid arrows are synchronous and dashed are asynchronous dependencies.

As shown in Figure 1, ILLIXR consists of three pipelines – perception, visual, and audio – each containing several components and interacting with each other through the ILLIXR communication interface and runtime. Figure 2 illustrates these interactions – the left side presents a timeline for an ideal schedule for the different components and the right side shows the dependencies between the components (enforced by the ILLIXR runtime). We distilled this workflow from multiple sources that describe different parts of a typical VR, AR, or MR pipeline, including conversations with several experts from academia and industry (§I).

ILLIXR represents a state-of-the-art system capable of running sophisticated XR applications and providing an end-to-end XR user experience. Nevertheless, XR is an emerging and evolving domain and no XR experimental testbed or commercial device can be construed as complete in the traditional sense. Compared to specific commercial headsets today, ILLIXR may miss a component (e.g., Oculus Quest 2 provides hand tracking) and/or it may have additional or more advanced components (e.g., except for HoloLens 2, most current XR systems do not have scene reconstruction). Further, while ILLIXR supports state-of-the-art algorithms for its components, new algorithms are continuously evolving. ILLIXR supports a modular and extensible design that makes it relatively easy to swap and add new components.

The system presented here assumes all computation is done on the XR device (i.e., no offloading to other edge devices or to the cloud) and we assume a single user (i.e., no communication with multiple XR devices). This is similar to current standalone headsets such as Oculus Quest 2.[2]

§II-A next describes the three ILLIXR pipelines and their components, §II-B describes the modular runtime architecture that integrates these components, and §II-C describes the metrics and telemetry support in ILLIXR. Table II summarizes the algorithm and implementation information for each component in the three pipelines. ILLIXR already supports multiple, easily interchangeable alternatives for some components; for lack of space, we pick one alternative, indicated by a * in the table, for detailed results in this paper.

[2]We are working on supporting networking, edge and cloud work partitioning, and multiparty XR within ILLIXR. Since component interfaces are well-specified and modular, a local component can be easily swapped with a remote one without modifying the rest of the system. We have already implemented offloading some components and plan a generalized offloading module that any component can use.

TABLE II: ILLIXR component algorithms and implementations. GLSL stands for OpenGL Shading Language. We used the default CPU-GPU partitioning in the reference implementation of each component. * represents the implementation alternative for which we provide detailed results.

Component	Algorithm	Implementation
Perception Pipeline		
Camera	ZED SDK* [31]	C++
Camera	Intel RealSense SDK [32]	C++
IMU	ZED SDK* [31]	C++
IMU	Intel RealSense SDK [32]	C++
VIO	OpenVINS* [33]	C++
VIO	Kimera-VIO [34]	C++
IMU Integrator	RK4* [33]	C++
IMU Integrator	GTSAM [35]	C++
Eye Tracking	RITnet [36]	Python,CUDA
Scene Reconstruction	ElasticFusion* [37]	C++,CUDA,GLSL
Scene Reconstruction	KinectFusion [38]	C++, CUDA
Visual Pipeline		
Reprojection	VP-matrix reprojection with pose [39]	C++, GLSL
Lens Distortion	Mesh-based radial distortion [39]	C++, GLSL
Chromatic Aberration	Mesh-based radial distortion [39]	C++, GLSL
Adaptive display	Weighted Gerchberg–Saxton [40]	CUDA
Audio Pipeline		
Audio Encoding	Ambisonic encoding [41]	C++
Audio Playback	Ambisonic manipulation, binauralization [41]	C++

A. Pipelines

The perception pipeline translates the user's physical motion into information understandable to the rest of the system so it can render and play the new scene and sound for the user's new position. The input to this part comes from sensors such as cameras and an inertial measurement unit (IMU) to provide the user's acceleration and angular velocity. The processing components include **camera and IMU processing**, head tracking or **VIO (Visual Inertial Odometry)** for obtaining low frequency but precise estimates of the user's pose (the position and orientation of their head), **IMU integration** for obtaining high frequency pose estimates, **eye tracking** for determining the gaze of the user, and **scene reconstruction** for generating a 3D model of the user's surroundings.

The visual pipeline takes information about the user's new pose from the perception pipeline and the submitted frame from the application, and produces the final display using two components. **Asynchronous reprojection** corrects the rendered image submitted by the application for optical distortions and compensates for latency from the application's rendering process by reprojecting the frame from the perspective of the freshly read/predicted pose [39].[3] The version of ILLIXR used here implements rotational reprojection or TimeWarp [39]) (we have since also implemented translational reprojection). ILLIXR supports computational **holography** [42] to present multiple focal planes to the user [43] (adaptive displays).[4]

The audio pipeline is responsible for generating spatial audio and is composed of **audio encoding**, which encodes monophonic sound sources into a virtual soundfield [45], and **audio playback**, which adds localization cues to the sound [46].

[3]We provide the ability to predict the pose when the frame will actually be displayed and reproject based on the predicted pose.

[4]Although we can generate holograms, we do not yet have a holographic display setup with SLMs and other optical elements, and there are no general purpose pre-assembled off-the-shelf holographic displays available. We currently therefore display the generated frames on a standard LCD monitor or a North Star AR headset [44].

B. Runtime and Communication Framework

Figure 2 shows an ideal execution timeline for the different XR components and their temporal dependencies. On the left, each colored rectangle boundary represents the period of the respective component — ideally, the component would finish execution before its next invocation. The right side shows a static representation of the dependencies among the components, illustrating the interaction between the different pipelines.[5] We say a consumer component exhibits a *synchronous* dependence on a producer component if the former has to wait for the last invocation of the latter to complete (solid arrow). An *asynchronous* dependence is softer, where the consumer can start execution with data from a previously complete invocation of the producer component (dashed arrow).

An XR system is unlikely to follow the idealized schedule due to shared and constrained resources and variable running times. Thus, an explicit runtime is needed for effective resource management and scheduling while maintaining the inter-component dependencies, resource constraints, and quality of experience. The ILLIXR runtime currently runs on Linux – it schedules resources while enforcing dependencies among the components, in part deferring to the Linux kernel and GPU driver.

To achieve extensibility and modularity, ILLIXR provides a well-defined communication framework, with components implemented as plugins supported by a plugin-loader.

The communication framework is structured around *event streams*. Event streams support writes, asynchronous reads, and synchronous reads. Synchronous reads allow a consumer to see every value produced by the producer, while asynchronous reads allow a consumer to ask for the latest value.

Plugins are distributed as shared-object files for the runtime to load. The runtime gives the plugins access to other plugins but in a limited sense; they can only interact through event streams. This architecture allows for modularity. Each component in Table II is implemented in its own plugin. Each plugin is interchangeable with another as long as it complies with the event-stream interface. Future researchers can test alternative implementations of a single plugin without needing to reinvent the rest of the system. Development can iterate quickly, because plugins are compiled independently.

ILLIXR is supported as a device driver in the Monado OpenXR runtime [16]. This allows ILLIXR to run OpenXR applications, including those developed using game engines such as Unity, Unreal, and Godot (currently only Godot and Unreal have OpenXR support on Linux).

Finally, although ILLIXR supports live sensor inputs (e.g., through cameras and IMUs) and an HMD display, it does not *require* such external hardware. To enable universal use and ease experimentation, the only requirement is a computer (desktop/laptop/embedded board). Offline, pre-recorded datasets can be fed to all parts of ILLIXR due to its well-defined and modular communication interfaces. As an example, ILLIXR's offline camera+IMU component reads from a pre-recorded dataset and publishes to the same output stream as a live camera+IMU component, appearing indistinguishable from a real camera/IMU to the rest of the system. Similarly, ILLIXR does not require an HMD and can display the final stereoscopic images to a regular monitor.

C. Metrics

ILLIXR provides several metrics to evaluate the goodness of the system. In addition to reporting conventional performance metrics such as per-component frame rate, per-component execution time, and power-related metrics, ILLIXR reports several QoE metrics: 1) motion-to-photon latency [47], a standard measure of the lag between user motion and image updates; 2) Structural Similarity Index Measure (SSIM) [48], one of the most commonly used image quality metrics in XR studies [49]–[51]; and 3) FLIP [52], a recently introduced image quality metric that addresses shortcomings of SSIM.

While ILLIXR currently implements SSIM and FLIP, its pose and image collection infrastructure is generic and extensible, enabling evaluation of other (evolving) metrics for image or video quality. This is important as such metrics for XR are still an active area of research. Notably, both SSIM and FLIP are image metrics, whereas the final output of the visual pipeline is a video, requiring consideration of aspects such as temporal coherence and smoothness (jitter) as well. For instance, VMAF [53] and Video ATLAS [54] have made important steps in this direction.

We do not yet compute a quality metric for audio beyond bitrate, but plan to add the recently developed AMBIQUAL [55].

III. Experimental Methodology

The goals of our experiments are to quantitatively show that (1) XR presents a rich opportunity for computer architecture and systems research in domain-specific edge systems; (2) fully exploiting this opportunity requires an end-to-end system that models the complex, interacting pipelines in an XR workflow; and (3) ILLIXR is a unique testbed that provides such an end-to-end system, enabling new research directions in domain-specific edge systems architecture in general and for XR in particular.

Towards the above goals, we perform a comprehensive characterization of a live end-to-end ILLIXR system on multiple hardware platforms with representative XR workloads. We also characterize standalone components of ILLIXR using appropriate component-specific off-the-shelf datasets. This section details our experimental setup.

A. Experimental Setup

There are no existing systems that meet all the aspirational performance, power, and quality criteria for a fully mobile XR experience as summarized in Table I. For our characterization, we chose to run ILLIXR on two hardware platforms, with three total configurations, representing a broad spectrum of power-performance tradeoffs and current XR devices.

A high-end desktop platform: We used a state-of-the-art desktop system with an Intel Xeon E-2236 CPU (6C12T) and a discrete NVIDIA RTX 2080 GPU. This platform's thermal design power (TDP) rating is far above what is deemed acceptable for a mobile XR system, but it is representative of the platforms on which current tethered systems run (e.g., Varjo VR-3 as in Table I) and it can be viewed as an approximate upper bound for performance of CPU+GPU based near-future embedded (mobile) systems.

An embedded platform with two configurations: We used an NVIDIA Jetson AGX Xavier development board [56] consisting of an Arm CPU (8C8T) and an NVIDIA Volta GPU. All experiments were run with the Jetson in 10 Watt mode, the lowest possible preset. We used two different configurations – a high performance one (*Jetson-HP*) and a low power one (*Jetson-LP*). We used the maximum available clock frequencies for Jetson-HP and half those for Jetson-LP. These configurations approximate the hardware and/or the TDP rating of several commercial mobile XR devices. For example, Magic Leap One [57] used a Jetson TX2 [58], which is similar in design and TDP to our Xavier configuration. HoloLens 2 [27]–[29] and the Qualcomm Snapdragon 835 [59] used in Oculus Quest [60] have TDPs in the same range as our two Jetson configurations.

I/O setup: For the live end-to-end ILLIXR system experiments, our I/O setup is as follows. For the perception pipeline, we connected a ZED Mini camera [61] to the above platforms via a USB-C cable. A user walked in our lab with the camera, providing live camera and IMU input (Table II) to ILLIXR. For the visual pipeline, we run representative VR and AR applications on a game engine on ILLIXR (§III-C) – these applications interact with the perception pipeline to provide the visual pipeline with the image frames to display.

[5] An additional constraint is that reprojection should be scheduled as late as possible (before the next vsync) so it has the freshest pose to generate the reprojected frame to be finally displayed.

TABLE III: Key ILLIXR parameters that required manual system-level tuning. Several other parameters were tuned at the component level.

Component	Parameter Range	Tuned	Deadline
Camera (VIO)	Frame rate = 15 – 100 Hz	15 Hz	66.7 ms
	Resolution = VGA – 2K	VGA	–
	Exposure = 0.2 – 20 ms	1 ms	–
IMU (Integrator)	Frame rate = $\leq$800 Hz	500 Hz	2 ms
Display (Visual pipeline, Application)	Frame rate = 30 – 144 Hz	120	8.33 ms
	Resolution = $\leq$2K	2K	–
	Field-of-view = $\leq$ 180	90	–
Audio (Encoding, Playback)	Frame rate = 48 – 96 Hz	48 Hz	20.8 ms
	Block size = 256 – 2048	1024	–

ILLIXR can display the (corrected and reprojected) images on both a desktop LCD monitor and a North Star AR headset [44] connected to the above hardware platforms. In this paper, we use a desktop monitor due to its ability to provide multiple resolution levels and refresh rates for experimentation. Although this means that the user does not see the display while walking with the camera, we practiced a trajectory that provides a reasonable response to the displayed images and used that (live) trajectory to collect results. Finally, for the audio pipeline, we use pre-recorded input. We run our experiments for approximately 30 seconds.

B. Integrated ILLIXR System Configuration

The end-to-end integrated ILLIXR configuration in our experiments uses the components as described in Table II except for scene reconstruction, eye tracking, and hologram. The OpenXR standard currently does not support an interface for an application to use the results of scene reconstruction and only recently added an interface for eye tracking. We, therefore, do not have any applications available to use these components in an integrated setting. Although we can generate holograms, we do not yet have a holographic display (§II-A). We do report results in standalone mode for these components using off-the-shelf component-specific datasets.

Configuring an XR system requires tuning multiple parameters of the different components to provide the best end-to-end user experience on the deployed hardware. This is a complex process involving the simultaneous optimization of many QoE metrics and system parameters. Currently tuning such parameters is a manual, mostly ad hoc process. Table III summarizes the key parameters for ILLIXR that required system-level tuning, the range available in our system for these parameters, and the final value we chose at the end of our manual tuning. We made initial guesses based on our intuition, and then chose final values based on both our perception of the smoothness of the system and profiling results. We expect the availability of ILLIXR will enable new research in more systematic techniques for performing such end-to-end system optimization.

C. Applications

To evaluate ILLIXR, we used four different XR applications: Sponza [62], Materials [63], Platformer [64], and a custom AR demo application with sparse graphics.

Sponza places the user inside the atrium of the famous Sponza Palace in Dubrovnik, Croatia. The objective of the application is to showcase rendering of high polygon count meshes and global illumination. Materials presents the user with several sphere like objects that are composed of different complex materials, and showcases a variety of physically based rendering (PBR) techniques, such as displacement mapping, subsurface scattering, and anisotropic reflections. Platformer contains a maze with crab-like "enemies" that the user can shoot, and focuses on demonstrating physics and collisions. The AR application contains a single light source, a few virtual objects, and an animated ball, and showcases the overlaying of both stationary and moving virtual objects on the physical world. We had to develop our own AR application because existing applications in the XR ecosystem all predominantly target VR (outside of Microsoft HoloLens).

The applications were chosen for diversity of rendering complexity, with Sponza being the most graphics-intensive and AR demo being the least. We did not evaluate an MR application as OpenXR does not yet contain a scene reconstruction interface.

Since Unity and Unreal did not have OpenXR support on Linux when this work was done, all four applications use the Godot game engine [65], a popular open-source alternative with Linux OpenXR support.

D. Experiments with Standalone Components

For more detailed insight into the individual components of ILLIXR, we also ran each component by itself on our desktop and embedded platforms using off-the-shelf standalone datasets: Vicon Room 1 Medium [66] for VIO, dyson_lab [37] for scene reconstruction, OpenEDS [67] for eye tracking, 2560×1440 pixel frames from VR Museum of Fine Art [68] for reprojection and hologram, and 48 KHz audio clips [69], [70] from Freesound [71] for audio encoding and playback.

E. Metrics

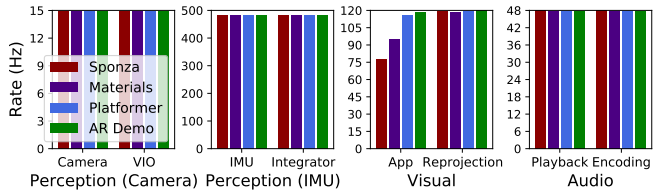
Execution time: We used NSight Systems [72] to obtain the overall execution timeline of the components and ILLIXR, including serial CPU, parallel CPU, GPU compute, and GPU graphics phases. VTune [73] (desktop only) and `perf` [74] provided CPU hotspot analysis and hardware performance counter information. NSight Compute [75] and Graphics [76] provided detailed information on CUDA kernels and GLSL shaders respectively. We also developed a logging framework that allows ILLIXR to easily collect the wall clock time and CPU time of each of its components with negligible overhead. We determined the contribution of a given component towards the CPU time by computing the total CPU cycles consumed by that component as a fraction of the cycles used by all components.

Power and energy: For the desktop, we measured CPU power using `perf` and GPU power using `nvidia-smi` [77] (a standard method to measure power on NVIDIA GPUs [78]–[80]). On the embedded platform, we collected power and energy using a custom profiler, similar to [81], that monitored different power rails present on the board [82] to calculate both the average power and average energy for different components of the system: CPU, GPU, DDR, SoC (on-chip microcontrollers; excludes CPU and GPU power), and Sys (display, storage, I/O) [83].

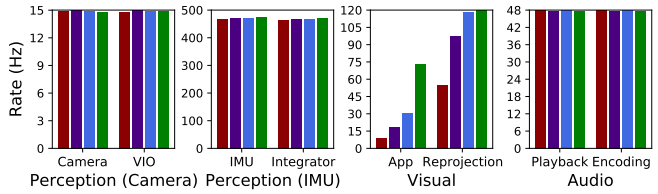
Motion-to-photon latency (MTP): We compute motion-to-photon latency as the age of the reprojected image's pose when the frame is submitted for display. This latency is the sum of how old the IMU sample used for pose calculation was,[6] the time taken by reprojection itself, and the wait time until the frame buffer is accepted for display. Mathematically, this can be formulated as: $latency = t_{imu\_age} + t_{reprojection} + t_{swap}$. We do not include $t_{display}$, the time taken to display the frame on screen, in this calculation. This calculation is performed and logged by the reprojection component every time it runs. If reprojection misses vsync, the additional latency is captured in t_{swap}.

Image Quality: To compute image quality, we compare the outputs of the actual XR system being studied to those of an idealized configuration that directly receives ground truth poses from a data set (we use Vicon Room 1 Medium [66]). A key implementation challenge was that collecting the post-reprojection images from the GPU for comparison incurs too much overhead and perturbs the run. This is mitigated by collecting images produced by the application renderer (which are cheaper to collect) and the poses generated (ground truth for the idealized system), and then applying reprojection offline (using the above poses) to get the final image that would have been displayed. We compare the reprojected images of the actual and idealized systems to compute both SSIM and FLIP. We report 1-FLIP to be consistent with SSIM (0 being no similarity and 1 being identical images).

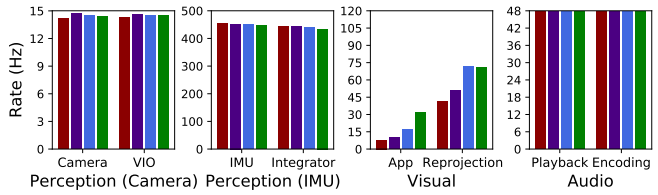
[6] This does not account for pose prediction as is common practice. Pose prediction potentially reduces MTP but it is hard to account for mispredictions.



(a) Desktop



(b) Jetson-HP



(c) Jetson-LP

Fig. 3: Average frame rate for each component in the different pipelines on each application and hardware platform. The y-axis is capped at the target frame rate of the pipeline.

IV. Results

We present a comprehensive characterization of the integrated ILLIXR system (§IV-A) and its components in isolation (§IV-B). These results constitute the first detailed characterization of an entire XR system for architecture and systems researchers. §V describes the implications of these results.

A. Integrated ILLIXR System

Sections IV-A1–IV-A3 present performance, power, and QoE for the integrated ILLIXR system, all of which contribute to the goodness of the user's XR experience.

1) Performance: We present ILLIXR system performance in terms of achieved frame rate for each component, and compare it against the target frame rate (Table III). Further, to understand the importance of each component to execution resources, we present the contribution of each component to the total CPU execution cycles.

Component frame rates. Figures 3(a)-(c) show each component's average frame rate for each application, for a given hardware configuration. Components with the same target frame rate are presented in the same graph (the maximum value on the y-axis is the target). Thus, we show separate graphs for components in the perception, visual, and audio pipelines, with the perception pipeline further divided into two graphs representing the camera and IMU driven components.

Focusing on the desktop, Figure 3a shows that virtually all components meet, or almost meet, their target frame rates (the application component for Sponza and Materials are the only exceptions). The IMU components are slightly lower than the target frame rate only because of scheduling non-determinism at their required 2 ms period. This high performance, however, comes with a significant power cost.

Moving to the lower power Jetson, we find more components missing their target frame rates. With Jetson-LP, only the audio pipeline is able to meet its target. The visual pipeline components are both severely degraded in all cases for Jetson-LP and most cases for Jetson-HP.

Although we assume modern display resolutions and refresh rates, future systems will support larger and faster displays with larger field-of-view and will integrate more components, further stressing the entire system.

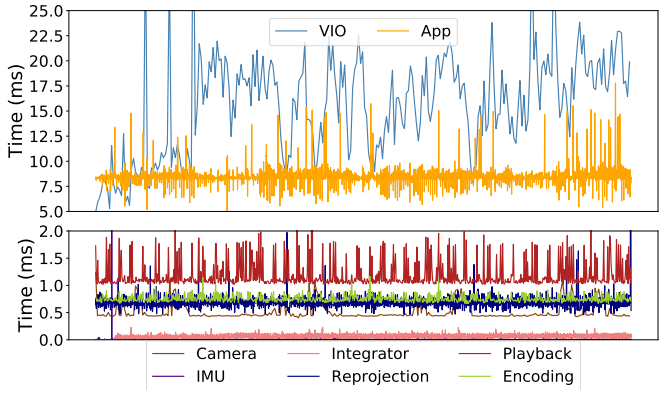


Fig. 4: Per-frame execution times for Platformer on the desktop. The top graph shows VIO and the application. The bottom graph shows the remaining components. Note the different scales of the y-axes.

Execution time per frame. While achieved frame rate cannot exceed the target frame rate because it is controlled by the runtime, execution time per frame can be arbitrarily low or high. The mean execution times follow trends similar to those for frame rates; however, the standard deviations for execution time are surprisingly significant in many cases (we omit graphs for space). As a result, although the mean frame rate for some components is comfortably within the target deadline, several frames do miss their deadlines; e.g., VIO on Jetson-LP. Whether this affects the user's QoE depends on how well the rest of the system compensates for these missed deadlines. For example, even if the VIO runs a little behind the camera, the IMU part of the perception pipeline may be able to compensate. Similarly, if the application misses some deadlines, reprojection may be able to compensate. §IV-A3 provides results for system-level QoE metrics that address these questions.

For a more detailed view, Figure 4 shows the execution time of each frame of each ILLIXR component for Platformer on the desktop (other timelines are omitted for space). The components are split across two graphs for clarity. We expect variability in VIO (blue) and application (yellow) since these computations are known to be input-dependent. However, we see significant variability in the other components as well. This variability is due to scheduling and resource contention, and contributes to the high standard deviation of execution time.

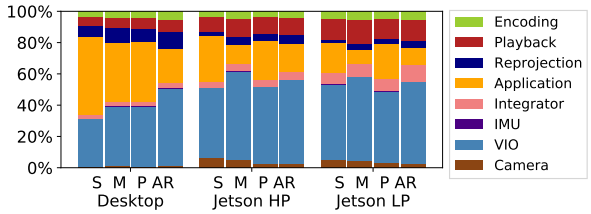
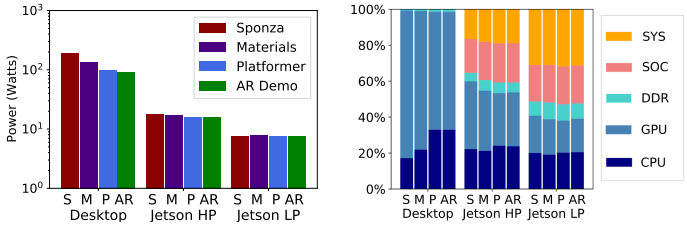


Fig. 5: Contributions of ILLIXR components to CPU time for different applications and hardware. S = Sponza, M = Materials, P = Platformer, AR = AR Demo.

Distribution of cycles. Figure 5 shows the relative attribution of the total cycles consumed in the CPU to the different ILLIXR components for the different applications and hardware platforms. Focusing first on the desktop, Figure 5 shows that VIO and the application are the largest

contributors to CPU cycles, with one or the other dominating, depending on the application. Reprojection and audio playback follow next, becoming larger relative contributors as the application complexity reduces. Although reprojection never exceeds 10% of the total cycles, it is a dominant contributor to motion-to-photon latency and, as discussed below, cannot be neglected for optimization.

Jetson-HP and Jetson-LP show similar trends except that we find that the application's and reprojection's relative contribution decreases while other components such as the IMU integrator become more significant relative to the desktop. This phenomenon occurs because with the more resource-contrained Jetson configurations, the application and reprojection often miss their deadline and are forced to skip the next frame. Thus, the overall work performed by these components reduces, but shows up as poorer end-to-end QoE metrics discussed later (§IV-A3).



(a) Total Power (W) (b) Power Breakdown

Fig. 6: (a) Total power (note log scale) and (b) relative contribution to power by different hardware units for ILLIXR running each application on each hardware platform. S = Sponza, M = Materials, P = Platformer, AR = AR Demo.

2) Power: **Total Power.** Figure 6a shows the total power consumed by ILLIXR running each application on each hardware platform. The power gap from the ideal (Table I) is severe on all three platforms. Jetson-LP, the lowest power platform, is about two orders of magnitude off in terms of the ideal power while the desktop is off by three. As with performance, larger resolutions, frame rates, and field-of-views, and more components would further widen this gap.

Contribution to power from different hardware components. Figure 6b shows the relative contribution of different hardware units to the total power, broken down as CPU, GPU, DDR (DRAM), SoC, and Sys (§III-E). Although the GPU dominates power on the desktop, all the above hardware units contribute substantially on the Jetson. SoC and Sys power are often ignored, but they consume more than 50% of total power on Jetson-LP. These results highlight the need for studying all aspects of the system – reducing the power gap for XR requires optimizing system-level hardware components as well.

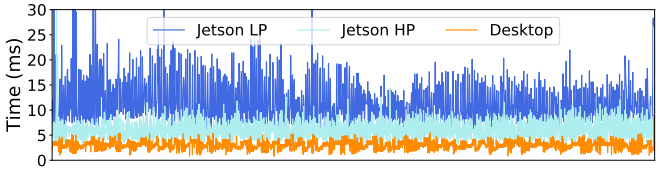


Fig. 7: Motion-to-photon latency per frame of Platformer.

3) Quality of Experience: Quality of an XR experience is often determined through user studies [84]–[86]; however, these can be expensive, time consuming, and subjective. ILLIXR, therefore, provides several quantitative metrics to measure QoE. We report both results of visual examination and quantitative metrics below.

Visual examination. A detailed user study is outside the scope of this work, but there were several artifacts in the displayed image that were clearly visible. As indicated by the performance metrics, the desktop displayed smooth images for all four applications. Jetson-HP showed perceptibly increased judder for Sponza. Jetson-LP showed dramatic pose drift and clearly unacceptable images for Sponza and Materials, though it was somewhat acceptable for the less intense Platformer and AR Demo. As discussed in §IV-A1, the average VIO frame rate for Jetson-LP stayed high, but the variability in the per-frame execution time resulted in many missed deadlines, which could not be fully compensated by the IMU or reprojection. These effects are quantified with the metrics below.

TABLE IV: Motion-to-photon latency in milliseconds (mean±std dev), without $t_{display}$ (a few ms). Target MTP is 20 ms for VR and 5 ms for AR (Table I).

Platform	Sponza	Materials	Platformer	AR Demo
Desktop	3.1±1.1	3.1±1.0	3.0±0.9	3.0±0.9
Jetson-HP	13.5±10.7	7.7±2.7	6.0±1.9	5.6±1.4
Jetson-LP	19.3±14.5	16.4±4.9	11.3±4.7	12.0±3.4

TABLE V: Image quality metrics (mean±std dev) for Sponza.

	Desktop	Jetson-HP	Jetson-LP
SSIM	0.83±0.04	0.80±0.05	0.68±0.09
1-FLIP	0.86±0.05	0.85±0.05	0.65±0.17

Motion-to-photon latency. Table IV shows the mean and standard deviation of MTP for all cases. Figure 7 shows MTP for each frame over the execution of Platformer on all hardware (we omit the other applications for space).

Table IV and our detailed per-frame data shows that the desktop can achieve the target VR MTP of 20ms (Table I) for virtually all frames. Jetson-HP is able to make the target VR MTP for the average frame for all applications and for most frames for all except Sponza (even after adding $t_{display}$). Jetson-LP shows a significant MTP degradation – on average, it still meets the target VR MTP, but both the mean and variability increase with increasing complexity of the application until Sponza is practically unusable. For AR, most cases appear to meet the target (5ms) on the desktop, but adding $t_{display}$ significantly exceeds the target. Both Jetsons cannot make the target AR MTP for an average frame. Thus, the MTP data collected by ILLIXR is consistent with the visual observations reported above.

Offline metrics for image quality. Table V shows the mean and standard deviation for SSIM and 1-FLIP for Sponza on all hardware configurations. (We omit other applications for space.) Recall that these metrics require offline analysis and use a different offline trajectory dataset for VIO (that comes with ground truth) from the live camera trajectory reported in the rest of this section (§III-E). Nevertheless, the visual experience of the applications is similar. We find that the metrics degrade as the hardware platform becomes more constrained, although the SSIM and FLIP values seem deceptively high for the Jetsons (specifically the Jetson-LP where VIO shows a dramatic drift). Quantifying image quality for XR is known to be challenging [87], [88]. While some work has proposed the use of more conventional graphics-inspired metrics such as SSIM and FLIP, this is still a topic of ongoing research [89], [90]. ILLIXR is able to capture the proposed metrics, but our work also motivates and enables research on better image quality metrics for XR experiences.

B. Analyzing Components in Isolation

We next examine ILLIXR components in isolation. Tables VI and VII summarize the algorithmic tasks within each component, including the key computation and memory patterns, as well as the time spent in each task, analyzed on the high-end desktop platform. We do not show the sensor related ILLIXR components since they are quite simple, and we do not present a table for eye tracking as neural networks are well understood in the architecture community. We define tasks as distinct high level algorithmic steps. Figure 8 shows the CPU IPC (Instructions-Per-Cycle) and cycle breakdown for each component in aggregate. We do not provide a similar breakdown on the GPU as GPU profiling tools do not provide one.

TABLE VI: Task breakdown of VIO and scene reconstruction.

Task	Time	Computation	Memory Pattern
		VIO	
Feature detection Detects new features in the new camera images	15%	KLT; FAST	Globally mixed dense and sparse image accesses; locally dense image stencil
Feature matching Matches features across images	13%	KLT; GEMM; linear algebra	Globally mixed dense and sparse image accesses; locally dense image stencil; mixed dense and random feature map accesses
Feature initialization Adds new features to state	14%	SVD; Gauss-Newton; Jacobian; nullspace projection; GEMM	Dense feature map accesses; mixed dense and sparse state matrix accesses
MSCKF update Updates state using MSCKF algorithm	23%	SVD; Gauss-Newton; Cholesky; QR; Jacobian; nullspace projection; chi^2 check; GEMM	Dense feature map accesses; mixed dense and sparse state matrix accesses
SLAM update Updates state using EKF-SLAM algorithm	20%	Identical to **MSCKF update**	Similar, but not identical, to **MSCKF update**
Marginalization Removes features from state	5%	Cholesky; matrix arithmetic	Dense feature map and state matrix accesses
Other Miscellaneous tasks	10%	Gaussian filter; histogram	Globally dense image stencil
		Scene Reconstruction	
Camera Processing Processes incoming camera depth image	5%	Bilateral filter; invalid depth rejection	Locally dense image stencil
Image Processing Pre-processes RGB-D image for tracking and mapping	18%	Generation of vertex map, normal map, and image intensity; image undistortion; pose transformation of old map	Globally dense image accesses; locally dense image stencil; layout change from RGB_RGB $\rightarrow$ RR_GG_BB
Pose Estimation Estimates 6DOF pose	28%	Iterative closest point; photometric error; geometric error; reduction	Globally mixed dense and sparse image accesses; locally dense image accesses
Surfel Prediction Calculates active surfels in current frame	34%	Gauss-Newton; Cholesky; Jacobian; image sampling; fern encoding and matching; matrix transformations	Globally dense accesses to deformation graph; globally sparse image accesses; fern database lookup
Map Fusion Updates map with new surfel information	15%	Binary search; nearest neighbor search; matrix transformations	Globally sparse accesses to deformation graph; locally dense accesses to surfel list

TABLE VII: Task breakdown of visual and audio pipeline components.

Task	Time	Computation	Memory Pattern
		Reprojection	
FBO FBO state management	24%	Framebuffer bind and clear	Driver calls; CPU-GPU synchronization
OpenGL State Update Sets up OpenGL state	54%	OpenGL state updates; one drawcall per eye	Driver calls; CPU-GPU synchronization
Reprojection Applies reprojection transformation to image	22%	6 matrix-vector MULs/vertex	Dense accesses to uniform, vertex, and fragment buffers; 3 sparse texture accesses/fragment
		Hologram	
Hologram-to-depth Propagates pixel phase to depth plane	57%	Transcendentals; FMADDs; tree reduction	Globally dense accesses to hologram phases
Sum Sums phase differences from hologram-to-depth	$<$ 0.1%	Tree reduction	Globally dense accesses to partial sums
Depth-to-hologram Propagates depth plane phase to pixel	43%	Transcendentals; FMADDs; thread-local reduction	Globally dense accesses to depth phases
		Audio Encoding	
Normalization INT16 to FP32	7%	Element-wise FP32 division	Globally dense accesses to audio samples
Encoding Sample to soundfield mapping	81%	$Y[j][i] = D \times X[j]$	Globally dense column-major accesses to soundfield
Summation HOA soundfield summation	12%	$Y[i][j] += X_k[i][j]\ \forall k$	Globally dense row-major accesses to soundfield
		Audio Playback	
Psychoacoustic filter Applies optimization filter	29%	FFT; frequency domain convolution; IFFT	Butterfly pattern; Globally dense accesses to FFT output
Rotation Rotates soundfield using pose	6%	Transcendentals; FMADDs	Globally dense accesses to soundfield
Zoom Zooms soundfield using pose	5%	FMADDs	Globally dense column-major accesses to soundfield
Binauralization Applies HRTFs	60%	Identical to **psychoacoustic filter**	Identical to **psychoacoustic filter**

1) Common Observations: **Task Dominance** No component is composed of just one task. The most homogeneous is audio encoding where ambisonic encoding is 81% of the total – but accelerating just that would limit the overall speedup to 5.3× by Amdahl's law, which may not be sufficient given the power and performance gap shown in §IV-A. VIO is the most diverse, with seven major tasks and many more sub-tasks.

Task Diversity As shown in the component tables, there is a remarkable diversity of algorithmic tasks. These tasks are differently amenable for execution on the CPU (e.g., audio playback), GPU-compute (e.g., hologram), or GPU-graphics (e.g., reprojection). The compute patterns span stencils (KLT), GEMM, Gauss-Newton, and FFT, among others, and are often shared by components (e.g., Cholesky in VIO and scene reconstruction). The memory patterns are equally diverse, spanning dense and sparse, local and global, and row-major and column-major accesses to various types of data structures. Moreover, the working set sizes of the components range from tens of KB to hundreds of MB, which can change the characteristics of a given memory pattern. Salient examples include locally dense stencils becoming memory bandwidth bound when the working set size is large, and globally sparse accesses *not* becoming memory bound when the working set size is small relative to the on-chip storage capacity.

Microarchitectural Diversity Figure 8 shows significant microarchitectural diversity in addition to the algorithmic diversity discussed above. The IPC ranges from 0.3 (reprojection) to 3.5 (audio playback), with a variety of bottlenecks on the frontend and backend of the CPU pipeline.

Input-Dependence Although we saw significant per-frame execution time variability in all components in the integrated system, the only components that exhibit such variability standalone are VIO and scene reconstruction. VIO shows a range of execution time throughout its execution, with a coefficient of variation from 17% to 26% across the

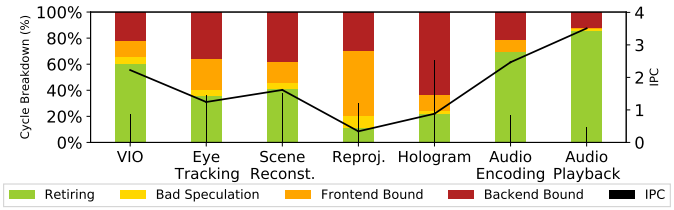


Fig. 8: Cycle breakdown and IPC of ILLIXR components.

studied datasets [66]. Scene reconstruction's execution time keeps steadily increasing due to the increasing size of its map. Loop closure attempts result in execution time spikes of 100's of ms, an order of magnitude more than its average per-frame execution time. We omit detailed graphs due to space.

2) Architectural Deep Dive of Components: We performed a detailed microarchitectural characterization of each component task (detailed data omitted for space). Below we highlight salient characteristics of each component.

VIO is a complex CPU workload with wide microarchitectural task diversity. The average IPC is 2.2, but there are several computationally intensive tasks with good vectorization that present higher average IPC; e.g., KLT and GEMM have an IPC of 3.2+. On the memory side, the working set size of most tasks is several hundred KBs, which is larger than the L1 and L2, but is small enough to fit in the LLC – the LLC shows only 0.1 MPKI (misses per kilo-instruction) while the L2 shows 7.9 MPKI. Our results include the effect of demand prefetchers in the CPU, which are quite effective – although the L2 shows 7.9 MPKI overall, it shows 0.6 MPKI for demand loads.

Eye tracking is a typical deep neural network that spends 74% of its total time executing convolution operations and activation functions, 19% performing batch copies, and the remaining 7% in miscellaneous tasks. Notably, the weights only occupy 0.98 MB but a total of 1922 MB is accessed during a forward pass, still making it memory bandwidth bound. However, the batch size is only two (one image per eye), resulting in low overall utilization of the GPU.

Scene reconstruction is a hybrid CPU-GPU workload that spends most of its time on GPU tasks. It is a memory bandwidth bound workload with many tasks consuming 200 GB/s of memory bandwidth and a few even surpassing 400 GB/s due to the large number of accesses per pixel (depth, normal, vertex, etc.). There is significant variation in data reuse, with some kernels possessing no reuse because of streaming accesses and some kernels possessing excellent reuse due to stencil operations. Moreover, conversions between CUDA (RR_GG_BB) and OpenGL (RGB_RGB) data layouts are common, as the tasks are split between GPU compute and GPU graphics. Reductions are a commonly found compute primitive in many tasks.

Reprojection is a hybrid CPU-GPU workload that spends a significant amount of time setting up framebuffer and OpenGL state prior to execution of the reprojection shaders. Framebuffer and OpenGL state updates are both performed via the GPU driver. Consequently, reprojection only has an IPC of 0.3, with most of the CPU cycles spent in frontend stalls due to the large instruction footprint of the GPU driver. The shaders themselves have very little compute and are memory bandwidth bound due to the size of the framebuffer. Moreover, the irregular texture cache accesses due to chromatic aberration correction result in a cache hit rate of 75%, which is low compared to the 90% achieved by non-correcting shaders.

Hologram executes all its tasks on the GPU as CUDA kernels, all of which are compute-bound. Almost all the executed instructions are FP fused multiply-add (FFMA), integer multiply-add (IMAD), and FP add (FADD). There is significant utilization of the FP64 hardware pipe – 75% in hologram-to-depth – as the transcendental calculations necessitate high precision. Memory traffic is limited to accessing each pixel's phase value only once in each task (reads in hologram-to-depth and writes in depth-to-hologram); however, this is significant data given the size of the framebuffer, and consumes 75-140 GB/s of memory bandwidth. The depth plane's phase values are accessed more often, but are only few tens of bytes in size, and are stored in the scratchpad for reuse.

Audio encoding is a compute-bound workload that is able to leverage vectorization and dense data structure accesses to achieve an IPC of 2.5, with 69% of cycles being spent retiring instructions. However, the component's IPC is limited due to backend stalls caused by division and modulo operations being bottlenecked by the lone hardware divider.

Audio playback is also a compute-bound workload, but has no division operations, resulting in even higher IPC and hardware utilization. The FFT and FMADD compute is vectorized, and the densely accessed 64 KB HOA soundfield comfortably fits in the L2 cache of the CPU, resulting in an average load latency of 7 cycles. Consequently, 86% of the cycles are spent retiring instructions, and a fairly high IPC of 3.5 is achieved.

V. Implications for Architecture and Systems Research

Architects have embraced specialization but most research focuses on accelerators for single programs. ILLIXR is motivated by research for specializing an entire domain-specific system, specifically edge systems with constrained resources, high computation demands, and goodness metrics based on end-to-end domain-specific quality of output; e.g., AR glasses, robots/drones, autonomous vehicles, etc. Our results characterize end-to-end performance, power, and QoE of an XR device, exposing new systems research opportunities and demonstrating ILLIXR as a unique testbed to enable exploration of these domain-specific systems, as follows.

A. Performance, Power, and QoE Gaps

Our results in §IV-A quantitatively show that collectively there is a several orders of magnitude performance, power, and QoE gap between current representative desktop and embedded class systems and the goals in Table I. The gap will be further exacerbated with higher fidelity displays and more components for a more feature-rich XR experience (e.g., scene reconstruction, eye tracking, hand tracking, and holography).

While the presence of these gaps itself is not a surprise, we provide the first such quantification and analysis. This provides insights for directions for architecture and systems research (below) as well as demonstrates ILLIXR as a one-of-a-kind testbed that can enable such research.

B. Component and Task Diversity

Our more detailed results in §IV-A1 and §IV-B show that no one component dominates all the metrics of interest. Thus, to close the aforementioned gaps, *all* components have to be considered together, even those that may appear relatively inexpensive at first glance. Moreover, there is a diversity of tasks within and across components, and no single task dominates. It is likely impractical to build a unique accelerator for every task given the large number of tasks and the severe power and area constraints for XR devices: leakage power will be additive across accelerators, and interfaces between these accelerators and other peripheral logic will further add to this power and area (we identified 27 tasks across all components, and expect more tasks with new components).

At the same time, our results show that a number of common primitives exist across components; e.g., Cholesky in VIO and scene reconstruction, making the case for shared hardware across components. While determining whether components share compute or memory primitives is possible by analyzing the components in isolation, whether we should instantiate only one Cholesky block or whether we should duplicate or pipeline it to meet the requirements of both components can only be answered by taking run-time interactions into account, which ILLIXR enables.

The choice of shared hardware not only impacts architecture (and the software stack such as compilation and scheduling), but also microarchitecture. Analyzing VIO in isolation, the communication between GEMM and Cholesky as part of the chi^2 check can be hardcoded to improve efficiency.

In a full system, the shared nature of Cholesky motivates a different microarchitecture with support for efficient communication with multiple producers and consumers instead of just GEMM. The above observations motivate new design space exploration tools research for system-wide codesign.

C. Full-system Power Contributions

§IV-A2 shows that addressing the power gap requires considering *system-level* hardware components, such as display and other I/O, including numerous sensors. While we do not measure individual sensor power, it is included in the Sys power on the Jetson. This motivates research in unconventional architecture paradigms such as on-sensor computing to save I/O power; e.g., the image processing tasks of VIO can be moved to the sensor so that only detected features and not entire camera frames are sent from the camera to the SoC. To reduce sensor power, sensor parameters can be tuned; e.g., reducing camera exposure can save power at the cost of a darker image. However, sensors are typically shared among components [91], and thus decisions regarding which tasks to move on-sensor and how to dynamically alter exposure must consider the entire system and not just one component. ILLIXR enables this.

D. Variability

Our results in §IV-A1 show large variability in per-frame processing times in many cases, either due to inherent input-dependent nature of the component (e.g., VIO) or due to resource contention from other components. This variability poses challenges to, and motivates research directions in, scheduling and resource partitioning and allocation of shared hardware resources. As discussed in §IV-B2, the components exhibit a variety of memory access patterns, which further complicates the design of shared resources. For instance, scene reconstruction is memory bandwidth bound, which motivates a processing-in-memory based design for it when considered in isolation. However, PIM-based designs steal memory bandwidth away from external memory accesses performed by other components in the system, which still need to be serviced to meet end-to-end QoE constraints. Thus, the design of a particular component is influenced by the rest of the system, which ILLIXR allows us to study.

E. End-to-end Quality Metrics

The end-to-end nature of ILLIXR allows it to provide end-to-end quality metrics such as MTP and see the impact of optimizations on the final displayed images. §IV-A3 shows that per-component metrics (e.g., VIO frame rate) are insufficient to determine the impact on the final user experience. At the same time, there is a continuum of acceptable experiences. Techniques such as approximate computing take advantage of such applications but often focus on subsystems that make it hard to assess end user impact. For example, VIO provides several parameters such as number of tracked points, SLAM features, etc. that can be tuned to trade off accuracy and performance. We experimented with two sets of parameters of VIO, and found that the average trajectory error could be reduced from 8.1 cm to 4.9 cm at the cost of a 1.5× increase in average per-frame execution time. Looking at pose error by itself (a subsystem metric [33]), it was unclear whether the performance cost justified the extra accuracy. However, when we ran the full system, we found that the lower accuracy setting was sufficient for good tracking, preventing us from allocating unnecessary resources for VIO. Similarly, at a system level, there are a multitude of parameters that need to be tuned for optimal performance of the system (Table III). Our results from §IV-A3 also motivate work on defining more perceptive quantitative metrics for image quality, which can be developed using ILLIXR.

F. Other Implications

ILLIXR also enables research in designing common compiler intermediate representations (IRs) to enable code generation for a variety of accelerators; device–edge server–cloud server work partitioning; content streaming and multiparty XR; and security and privacy.

G. Evaluation Tools

Architects use a variety of tools to evaluate their research. Design for future domain-specific systems is no different and ILLIXR lays the foundation for other tools as well. ILLIXR can be immediately used for architecture research described above using existing HLS tools to map components/SoC designs to RTL, FPGAs, and ASICs. Given the era of specialization, architecture researchers are increasingly using such methods. We can also use ILLIXR with simulation. We describe three ideas below, and expect that further research into this topic will yield even more solutions. 1) As with other benchmarks, we can scale down ILLIXR inputs and run on simulators such as gem5 [92], with the graphics components on GLTraceSim [93]. 2) We can collect input/output traces of each component via the ILLIXR runtime on a real machine, and organize them like a rosbag [94] to drive simulations of components of interest. 3) We can run all ILLIXR components on a common dilated clock – the runtime slows down components running on real hardware to match the real-time speed of simulated components. This models a hybrid real+simulated system. While building such simulators is possible, it is outside the scope of our paper.

VI. Related Work

There is an increasing interest in XR in the architecture community and several works have shown promising specialization results for individual XR components: Processing-in-Memory architectures for graphics [95], [96], video accelerators for VR [81], [97]–[100], 3D point cloud acceleration [101], [102], stereo acceleration [103], [104], computer vision accelerators [105], [106], specialized sensors [107], scene reconstruction hardware [108], and SLAM chips [109]–[111]. However, these works have been for single components in isolation or for a subset of the entire XR system. Significant work has also been done on remote rendering [49], [112]–[117] and 360 video streaming [118]–[121]. While these works do consider full end-to-end systems, they only focus on the rendering aspect of the system and do not consider the interplay between the various components of the system. Our work instead provides a full-system-benchmark, consisting of a set of XR components representing a full XR workflow, along with insights into their characteristics. Our goal is to propel architecture and systems research in the direction of full domain-specific *system* design.

There have been other works that have developed benchmark suites for XR. Raytrace in SPLASH-2 [11], VRMark [122], FCAT [123], and Unigine [124] are examples of graphics rendering benchmarks, but do not contain the perception pipeline nor adaptive display components. The SLAMBench [125]–[127] series of benchmarks aid the characterization and analysis of available SLAM algorithms, but contain neither the visual pipeline nor the audio pipeline. Unlike our work, none of these benchmarks suites look at multiple components of the XR pipeline and instead just focus on one aspect.

Project Esky [128] is a partially open-source XR runtime for Project North Star [44]. It uses closed head tracking software from Intel T26x cameras [129] and its communication and runtime framework has limited modularity – it supports a pre-defined set of plugins and does not expose interfaces to enable easy plug-n-play for new components and implementations. It also does not support OpenXR, the growing open XR application standard. Chen et al. [130] characterized AR applications on a commodity smartphone, but neither analyzed individual components nor considered futuristic components due to the closed-source nature of the platform. Yi et al. [131] performed scheduling of parallel inference and rendering tasks in smartphone-based AR applications, but did not consider other XR system components due to their closed-source nature.

VII. Conclusion

This paper develops ILLIXR, the first open source full system XR testbed for driving future end-to-end QoE-driven, co-designed architecture, systems, and algorithm research. ILLIXR-enabled analysis shows several interesting implications for system design – demanding performance, power, and quality requirements; a large diversity of critical tasks; significant computation variability that challenges scheduling and specialization; and a diversity in bottlenecks throughout the system, from I/O and compute requirements to power and resource contention. ILLIXR and our analysis have the potential to propel many new directions in architecture, systems, and algorithms research. Although ILLIXR already incorporates a representative workflow, research is already exposing new frontiers, such as the integration of ML and low-latency client-cloud applications. Through the ILLIXR consortium, we envision ILLIXR will become a community resource that continues to evolve, providing an increasingly comprehensive resource to solve the most difficult challenges of QoE-driven domain-specific system design.

VIII. Acknowledgments

We developed ILLIXR with many consultations with researchers and practitioners in many domains: audio, graphics, optics, robotics, signal processing, vision, and XR systems. We are deeply grateful for all of these discussions and specifically to the following: Ameen Akel, Wei Cui, Aleksandra Faust, Liang Gao, Rod Hooker, Matt Horsnell, Amit Jindal, Steve LaValle, Steve Lovegrove, David Luebke, Andrew Maimone, Vegard Øye, Maurizio Paganini, Martin Persson, Archontis Politis, Eric Shaffer, and Paris Smaragdis. The development of ILLIXR was supported by the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA, the Center for Future Architectures Research (C-FAR), one of the six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, the National Science Foundation under grant CCF 16-19245, and by a Google Faculty Research Award. The development of ILLIXR was also aided by generous hardware and software donations from Arm and NVIDIA.

References

[1] S. Adve, R. Bodik, and L. Ceze, "I-USHER: Interfaces to Unlock the Specialized HardwarE Revolution," *DARPA Information Science and Technology (ISAT) study*, Apr. 2019. [Online]. Available: https://rsim.cs.illinois.edu/Talks/I-USHER.pdf

[2] S. Stoller, M. Carbin, S. V. Adve, K. Agrawal, G. Blelloch, D. Stanzione, K. Yelick, and M. Zaharia, "Future directions for parallel and distributed computing," 2019, Report of an NSF workshop to influence the successor to the Scalable Parallelism in the Extreme (SPX) program. [Online]. Available: rsim.cs.illinois.edu/Pubs/SPX_2019_Workshop_Report.pdf

[3] M. McGuire, "Exclusive: How nvidia research is reinventing the display pipeline for the future of vr, part 1," 2017. [Online]. Available: https://www.roadtovr.com/exclusive-how-nvidia-research-is-reinventing-the-display-pipeline-for-the-future-of-vr-part-1/

[4] R. Aggarwal, T. P. Grantcharov, J. R. Eriksen, D. Blirup, V. B. Kristiansen, P. Funch-Jensen, and A. Darzi, "An evidence-based virtual reality training program for novice laparoscopic surgeons," *Annals of surgery*, vol. 244, no. 2, pp. 310–314, Aug 2006. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/16858196

[5] S. Krohn, J. Tromp, E. M. Quinque, J. Belger, F. Klotzsche, S. Rekers, P. Chojecki, J. de Mooij, M. Akbal, C. McCall, A. Villringer, M. Gaebler, C. Finke, and A. Thöne-Otto, "Multidimensional evaluation of virtual reality paradigms in clinical neuropsychology: Application of the vr-check framework," *J Med Internet Res*, vol. 22, no. 4, p. e16724, Apr 2020. [Online]. Available: https://doi.org/10.2196/16724

[6] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 72–81. [Online]. Available: https://doi.org/10.1145/1454115.1454128

[7] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE Intl. Symp. on Workload Characterization (IISWC)*, 2009, pp. 44–54.

[8] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, p. 1–17, Sep. 2006. [Online]. Available: https://doi.org/10.1145/1186736.1186737

[9] J. Bucek, K.-D. Lange, and J. v. Kistowski, "Spec cpu2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 41–42. [Online]. Available: https://doi.org/10.1145/3185768.3185771

[10] J. P. Singh, W.-D. Weber, and A. Gupta, "Splash: Stanford parallel applications for shared-memory," *SIGARCH Comput. Archit. News*, vol. 20, no. 1, p. 5–44, Mar. 1992. [Online]. Available: https://doi.org/10.1145/130823.130824

[11] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, ser. ISCA '95. New York, NY, USA: Association for Computing Machinery, 1995, p. 24–36. [Online]. Available: https://doi.org/10.1145/223982.223990

[12] C. Sakalis, C. Leonardsson, S. Kaxiras, and A. Ros, "Splash-3: A properly synchronized benchmark suite for contemporary research," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2016, pp. 101–111.

[13] S. V. Adve and M. Huzaifa, "An open-source testbed to democratize extended reality research, development, and benchmarking," NVIDIA GPU Technology Conference (GTC), April 2021. [Online]. Available: https://www.youtube.com/watch?v=ZY98lWksnpM

[14] "ILLIXR Consortium," https://illixr.org, 2021.

[15] The Khronos Group Inc., "The openxr specification," Available at https://www.khronos.org/registry/OpenXR/specs/1.0/html/xrspec.html (accessed April 5, 2020), Mar. 2020, version 1.0.8.

[16] "Monado - open source XR platform." [Online]. Available: https://monado.dev/

[17] D. Kanter, "Graphics processing requirements for enabling immersive vr," *Whitepaper*, 2015.

[18] M. McGuire, "Exclusive: How nvidia research is reinventing the display pipeline for the future of vr, part 2," 2017. [Online]. Available: https://www.roadtovr.com/exclusive-nvidia-research-reinventing-display-pipeline-future-vr-part-2/

[19] Varjo, "Varjo VR-3," 2020. [Online]. Available: https://varjo.com/products/vr-3/

[20] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, "Toward low-latency and ultra-reliable virtual reality," *IEEE Network*, vol. 32, no. 2, pp. 78–84, 2018.

[21] D. Takahashi, "Oculus chief scientist mike abrash still sees the rosy future through ar/vr glasses," https://venturebeat.com/2018/09/26/oculus-chief-scientist-mike-abrash-still-sees-the-rosy-future-through-ar-vr-glasses/, September 2018.

[22] Microsoft, "Microsoft hololens 2," 2019. [Online]. Available: https://www.microsoft.com/en-us/hololens/hardware

[23] Wikipedia contributors, "Hololens 2 — Wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/HoloLens_2, 2019.

[24] L. Goode, "The hololens 2 puts a full-fledged computer on your face," https://www.wired.com/story/microsoft-hololens-2-headset/, Feb 2019.

[25] Skarredghost, "All you need to know on hololens 2," https://skarredghost.com/2019/02/24/all-you-need-know-hololens-2/, May 2019.

[26] Rebecca Pool, "Ar/vr/mr 2020: The future now arriving," 2017. [Online]. Available: http://microvision.blogspot.com/2020/02/arvrmr-2020-future-now-arriving.html

[27] Ian Cutress, "Hot chips 31 live blogs: Microsoft hololens 2.0 silicon," https://www.anandtech.com/show/14775/hot-chips-31-live-blogs-microsoft-hololens-20-silicon, 2019.

[28] R. Smith and A. Frumusanu, "The Snapdragon 845 Performance Preview: Setting the Stage for Flagship Android 2018," https://www.anandtech.com/show/12420/snapdragon-845-performance-preview/4, 2018.

[29] K. Hinum, "Qualcomm Snapdragon 855 SoC - Benchmarks and Specs," 2019. [Online]. Available: https://www.notebookcheck.net/Qualcomm-Snapdragon-855-SoC-Benchmarks-and-Specs.375436.0.html

[30] I. Cutress, "Spotted: Qualcomm Snapdragon 8cx wafer on 7nm," https://www.anandtech.com/show/13687/qualcomm-snapdragon-8cx-wafer-on-7nm, 2018.

[31] Stereolabs, "ZED Software Development Kit," 2020. [Online]. Available: https://www.stereolabs.com/developers/release/

[32] Intel, "Intelrealsense," https://github.com/IntelRealSense/librealsense, 2015.

[33] "OpenVINS repository," https://github.com/rpng/open_vins, 2019.

[34] "KimeraVIO repository," https://github.com/MIT-SPARK/Kimera-VIO, 2017.

[35] "GTSAM repository," 2020. [Online]. Available: https://github.com/borglab/gtsam

[36] A. K. Chaudhary, R. Kothari, M. Acharya, S. Dangi, N. Nair, R. Bailey, C. Kanan, G. Diaz, and J. B. Pelz, "Ritnet: Real-time semantic segmentation of the eye for gaze tracking," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 3698–3702.

[37] "ElasticFusion repository," http://github.com/mp3guy/ElasticFusion/, 2015.

[38] "KinectFusionApp repository," https://github.com/chrdiller/KinectFusionApp, 2018.

[39] J. van Waveren, "The asynchronous time warp for virtual reality on consumer hardware," in *Proc. 22nd Conf. on Virtual Reality Software and Technology*, 2016, pp. 37–46.

[40] M. Persson, D. Engström, and M. Goksör, "Real-time generation of fully optimized holograms for optical trapping applications," in *Optical Trapping and Optical Micromanipulation VIII*, K. Dholakia and G. C. Spalding, Eds., vol. 8097. International Society for Optics and Photonics, 2011, pp. 291–299.

[41] "libspatialaudio repository," https://github.com/videolabs/libspatialaudio, 2019.

[42] R. D. Leonardo, F. Ianni, and G. Ruocco, "Computer generation of optimal holograms for optical trap arrays," *Opt. Express*, vol. 15, no. 4, pp. 1913–1922, Feb 2007. [Online]. Available: http://www.opticsexpress.org/abstract.cfm?URI=oe-15-4-1913

[43] J.-H. R. Chang, B. V. K. V. Kumar, and A. C. Sankaranarayanan, "Towards multifocal displays with dense focal stacks," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 198:1–198:13, Dec. 2018. [Online]. Available: http://doi.acm.org/10.1145/3272127.3275015

[44] Leap Motion, "Project North Star," 2020. [Online]. Available: https://github.com/leapmotion/ProjectNorthStar

[45] F. Hollerweger, "An introduction to higher order ambisonic," 2008.

[46] M. Frank, F. Zotter, and A. Sontacchi, "Producing 3d audio in ambisonics," in *Audio Engineering Society Conference: 57th International Conference: The Future of Audio Entertainment Technology – Cinema, Television and the Internet*, Mar 2015. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=17605

[47] D. Wagner, "Motion to Photon Latency in Mobile AR and VR," 2018. [Online]. Available: https://medium.com/@DAQRI/motion-to-photon-latency-in-mobile-ar-and-vr-99f82c480926

[48] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[49] J. Meng, S. Paul, and Y. C. Hu, "Coterie: Exploiting frame similarity to enable high-quality multiplayer vr on commodity mobile devices," in *Proc. of the 25th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20, 2020, p. 923–937.

[50] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, N. Dai, and H.-S. Lee, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," *IEEE Transactions on Mobile Computing*, 2019.

[51] A. Schollmeyer, S. Schneegans, S. Beck, A. Steed, and B. Froehlich, "Efficient hybrid image warping for high frame-rate stereoscopic rendering," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 4, pp. 1332–1341, 2017.

[52] P. Andersson, T. Akenine-Möller, J. Nilsson, K. Åström, M. Oskarsson, and M. Fairchild, "Flip: A difference evaluator for alternating images," *Proceedings of the ACM in Computer Graphics and Interactive Techniques*, vol. 3, no. 2, 2020.

[53] Z. Li, C. Bampis, J. Novak, A. Aaron, K. Swanson, A. Moorthy, and J. De Cock, "Vmaf: The journey continues," https://netflixtechblog.com/vmaf-the-journey-continues-44b51ee9ed12, October 2018.

[54] C. G. Bampis and A. C. Bovik, "Learning to predict streaming video qoe: Distortions, rebuffering and memory," *ArXiv*, vol. abs/1703.00633, 2017.

[55] M. Narbutt, J. Skoglund, A. Allen, M. Chinen, D. Barry, and A. Hines, "Ambiqual: Towards a quality metric for headphone rendered compressed ambisonic spatial audio," *Applied Sciences*, vol. 10, no. 9, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/9/3188

[56] NVIDIA, "NVIDIA AGX Xavier," 2017. [Online]. Available: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit

[57] M. Leap, "Magic Leap 1," 2019. [Online]. Available: https://www.magicleap.com/en-us/magic-leap-1

[58] NVIDIA, "NVIDIA TX2," 2017. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/

[59] F. Abazovic, "Snapdragon 835 is a 10nm slap in Intel's face ," 2018. [Online]. Available: https://www.fudzilla.com/news/mobile/42154-snapdragon-835-in-10nm-is-slap-in-intel-s-face

[60] Oculus, "Oculus Quest: All-in-One VR Headset," 2019. [Online]. Available: https://www.oculus.com/quest/?locale=en_US

[61] Stereolabs, "ZED Mini - Mixed-Reality Camera," 2018. [Online]. Available: https://www.stereolabs.com/zed-mini/

[62] Monado, "Sponza scene in Godot with OpenXR addon," https://gitlab.freedesktop.org/monado/demos/godot-sponza-openxr, 2019.

[63] Godot, "Material testers," https://github.com/godotengine/godot-demo-projects/tree/master/3d/material_testers, 2020.

[64] ——, "Platformer 3D," https://github.com/godotengine/godot-demo-projects/tree/master/3d/platformer, 2020.

[65] J. Linietsky, A. Manzur, and contributors, "Godot," https://godotengine.org/, 2020.

[66] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[67] S. J. Garbin, Y. Shen, I. Schuetz, R. Cavin, G. Hughes, and S. S. Talathi, "OpenEDS: Open Eye Dataset," 2019.

[68] F. Sinclair, "The vr museum of fine art," 2016. [Online]. Available: https://store.steampowered.com/app/515020/The_VR_Museum_of_Fine_Art/

[69] "Science Teacher Lecturing," May 2013. [Online]. Available: https://freesound.org/people/SpliceSound/sounds/188214/

[70] "Radio Recording," https://freesound.org/people/waveplay./sounds/397000/, July 2017.

[71] "Freesound," https://freesound.org/.

[72] NVIDIA, "Nsight systems," 2020. [Online]. Available: https://developer.nvidia.com/nsight-systems

[73] Intel, "Intel vtune profiler," 2020. [Online]. Available: https://software.intel.com/content/www/us/en/develop/tools/vtune-profiler.html

[74] N/A, "perf," 2020. [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page

[75] NVIDIA, "Nsight compute," 2020. [Online]. Available: https://developer.nvidia.com/nsight-compute

[76] ——, "Nsight graphics," 2020. [Online]. Available: https://developer.nvidia.com/nsight-graphics

[77] ——, "Nvidia system management interface," 2020. [Online]. Available: https://developer.nvidia.com/nvidia-system-management-interface

[78] G. Ali, S. Bhalachandra, N. Wright, A. Sill, and Y. Chen, "Evaluation of power controls and counters on general-purpose graphics processing units (gpus)," 2020.

[79] C.-H. Hsu, S.-H. Chang, J.-H. Liang, H.-P. Chou, C.-H. Liu, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, and D.-C. Juan, "Monas: Multi-objective neural architecture search using reinforcement learning," *arXiv preprint arXiv:1806.10332*, 2018.

[80] J. Lew, D. A. Shah, S. Pati, S. Cattell, M. Zhang, A. Sandhupatla, C. Ng, N. Goli, M. D. Sinclair, T. G. Rogers *et al.*, "Analyzing machine learning workloads using a detailed gpu simulator," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 151–152.

[81] Y. Leng, C.-C. Chen, Q. Sun, J. Huang, and Y. Zhu, "Energy-efficient video processing for virtual reality," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: ACM, 2019, pp. 91–103. [Online]. Available: http://doi.acm.org/10.1145/3307650.3322264

[82] NVIDIA, "Software-based power consumption modeling." [Online]. Available: https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/power_management_jetson_xavier.html#wwpID0E0AG0HA

[83] ——, "Xavier series soc technical reference manual." [Online]. Available: https://developer.nvidia.com/embedded/dlc/xavier-series-soc-technical-reference-manual

[84] B. Bauman and P. Seeling, "Towards predictions of the image quality of experience for augmented reality scenarios," *arXiv preprint arXiv:1705.01123*, 2017.

[85] Y. Arifin, T. G. Sastria, and E. Barlian, "User experience metric for augmented reality application: a review," *Procedia Computer Science*, vol. 135, pp. 648–656, 2018.

[86] Y. Zhu, X. Min, D. Zhu, K. Gu, J. Zhou, G. Zhai, X. Yang, and W. Zhang, "Toward better understanding of saliency prediction in augmented 360 degree videos," 2020.

[87] B. Zhang, J. Zhao, S. Yang, Y. Zhang, J. Wang, and Z. Fei, "Subjective and objective quality assessment of panoramic videos in virtual reality environments," in *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE, 2017, pp. 163–168.

[88] H. G. Kim, H.-T. Lim, and Y. M. Ro, "Deep virtual reality image quality assessment with human perception guider for omnidirectional image," *IEEE*

Transactions on Circuits and Systems for Video Technology, vol. 30, no. 4, pp. 917–928, 2019.

[89] J. Yang, T. Liu, B. Jiang, H. Song, and W. Lu, "3d panoramic virtual reality video quality assessment based on 3d convolutional neural networks," *IEEE Access*, vol. 6, pp. 38 669–38 682, 2018.

[90] H.-T. Lim, H. G. Kim, and Y. M. Ra, "Vr iqa net: Deep virtual reality image quality assessment using adversarial learning," in *2018 IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 6737–6741.

[91] S. Han, B. Liu, R. Cabezas, C. Twigg, P. Zhang, J. Petkau, T.-H. Yu, C.-J. Tai, M. Akbay, Z. Wang, A. Nitzan, G. Dong, Y. Ye, L. Tao, C. Wan, and R. Wang, "Megatrack: monochrome egocentric articulated hand-tracking for virtual reality," *ACM Transactions on Graphics*, vol. 39, 07 2020.

[92] N. Binkert, B. Beckmann, G. Black, S. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib Bin Altaf, N. Vaish, M. Hill, and D. Wood, "The gem5 simulator," *SIGARCH Computer Architecture News*, vol. 39, pp. 1–7, 08 2011.

[93] A. Sembrant, T. E. Carlson, E. Hagersten, and D. Black-Schaffer, "A graphics tracing framework for exploring cpu+gpu memory systems," in *2017 IEEE Intl. Symposium on Workload Characterization (IISWC)*, 2017, pp. 54–65.

[94] T. Field, J. Leibs, J. Bowman, and D. Thomas, "rosbag," 2021. [Online]. Available: http://wiki.ros.org/rosbag

[95] C. Xie, S. L. Song, J. Wang, W. Zhang, and X. Fu, "Processing-in-memory enabled graphics processors for 3d rendering," in *2017 IEEE Intl. Symp. on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 637–648.

[96] C. Xie, X. Zhang, A. Li, X. Fu, and S. Song, "Pim-vr: Erasing motion anomalies in highly-interactive virtual reality world with customized memory cube," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2019, pp. 609–622.

[97] Y. Leng, C.-C. Chen, Q. Sun, J. Huang, and Y. Zhu, "Semantic-aware virtual reality video streaming," in *Proc. of the 9th Asia-Pacific Workshop on Systems*, ser. APSys '18, 2018, pp. 21:1–21:7.

[98] A. Mazumdar, A. Alaghi, J. T. Barron, D. Gallup, L. Ceze, M. Oskin, and S. M. Seitz, "A hardware-friendly bilateral solver for real-time virtual reality video," in *Proc. of High Performance Graphics*, 2017, pp. 13:1–13:10.

[99] A. Mazumdar, T. Moreau, S. Kim, M. Cowan, A. Alaghi, L. Ceze, M. Oskin, and V. Sathe, "Exploring computation-communication tradeoffs in camera systems," in *2017 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2017, pp. 177–186.

[100] P. Ranganathan, D. Stodolsky, J. Calow, J. Dorfman, M. Guevara, C. W. Smullen IV, A. Kuusela, R. Balasubramanian, S. Bhatia, P. Chauhan *et al.*, "Warehouse-scale video acceleration: co-design and deployment in the wild," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 600–615.

[101] T. Xu, B. Tian, and Y. Zhu, "Tigris: Architecture and algorithms for 3D perception in point clouds," in *Proc. 52nd Annual IEEE/ACM Intl. Symposium on Microarchitecture*, ser. MICRO '52, 2019, pp. 629–642.

[102] Y. Feng, B. Tian, T. Xu, P. Whatmough, and Y. Zhu, "Mesorasi: Architecture support for point cloud analytics via delayed-aggregation," *2020 53rd Annual IEEE/ACM Intl. Symp. on Microarchitecture (MICRO)*, Oct 2020.

[103] J. Choi, E. P. Kim, R. A. Rutenbar, and N. R. Shanbhag, "Error resilient mrf message passing architecture for stereo matching," in *SiPS 2013 Proceedings*, 2013, pp. 348–353.

[104] Y. Feng, P. Whatmough, and Y. Zhu, "Asv: Accelerated stereo vision system," in *Proc. 52nd Ann. IEEE/ACM Intl. Symp. on Microarchitecture*, ser. MICRO '52, 2019, pp. 643–656.

[105] Y. Zhu, A. Samajdar, M. Mattina, and P. Whatmough, "Euphrates: Algorithm-soc co-design for low-power mobile continuous vision," in *Proc. of the 45th Annual Intl. Symposium on Computer Architecture*, 2018, pp. 547–560.

[106] S. Triest, D. Nikolov, J. Rolland, and Y. Zhu, "Co-optimization of optics, architecture, and vision algorithms," in *Workshop on Approximate Computing Across the Stack (WAX)*, 2019.

[107] V. Kodukula, A. Shearer, V. Nguyen, S. Lingutla, Y. Liu, and R. LiKamWa, "Rhythmic pixel regions: multi-resolution visual sensing system towards high-precision visual computing at low power," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 573–586.

[108] Q. Gautier, A. Althoff, and R. Kastner, "Fpga architectures for real-time dense slam," in *2019 IEEE 30th Intl. Conf. on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160-052X, 2019, pp. 83–90.

[109] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, "Navion: A 2-mw fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 1106–1119, April 2019.

[110] D. K. Mandal, S. Jandhyala, O. J. Omer, G. S. Kalsi, B. George, G. Neela, S. K. Rethinagiri, S. Subramoney, L. Hacking, J. Radford, E. Jones, B. Kuttanna, and H. Wang, "Visual inertial odometry at the edge: A hardware-software co-design approach for ultra-low latency and power," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2019, pp. 960–963.

[111] Y. Gan, Y. Bo, B. Tian, L. Xu, W. Hu, S. Liu, Q. Liu, Y. Zhang, J. Tang, and Y. Zhu, "Eudoxus: Characterizing and accelerating localization in autonomous machines industry track paper," in *2021 IEEE Intl. Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 827–840.

[112] S. Zhao, H. Zhang, S. Bhuyan, C. S. Mishra, Z. Ying, M. T. Kandemir, A. Sivasubramaniam, and C. R. Das, "Deja View: Spatio-Temporal Compute Reuse for Energy-Efficient 360 VR Video Streaming," in *Proceedings of the 47th International Symposium on Computer Architecture*, ser. ISCA '20, 2020.

[113] T. Liu, S. He, S. Huang, D. Tsang, L. Tang, J. Mars, and W. Wang, "A benchmarking framework for interactive 3d applications in the cloud," *53rd Ann. IEEE/ACM Intl. Symp. on Microarchitecture (MICRO)*, Oct 2020.

[114] X. Liu, C. Vlachou, F. Qian, C. Wang, and K.-H. Kim, "Firefly: Untethered multi-user VR for commodity mobile devices," in *2020 USENIX Annual Technical Conference*, Jul. 2020, pp. 943–957.

[115] K. Boos, D. Chu, and E. Cuervo, "Flashback: Immersive virtual reality on mobile devices via rendering memoization," in *Proc. of the 14th Intl. Conference on Mobile Systems, Applications, and Services*, 2016, p. 291–304.

[116] Y. Li and W. Gao, "Muvr: Supporting multi-user mobile virtual reality with resource constrained edge cloud," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 1–16.

[117] C. Xie, X. Li, Y. Hu, H. Peng, M. Taylor, and S. L. Song, "Q-vr: system-level design for future mobile collaborative virtual reality," in *Proc. 26th ACM Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 587–599.

[118] Y. Bao, T. Zhang, A. Pande, H. Wu, and X. Liu, "Motion-prediction-based multicast for 360-degree video transmissions," in *2017 14th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2017, pp. 1–9.

[119] M. Dasari, A. Bhattacharya, S. Vargas, P. Sahu, A. Balasubramanian, and S. R. Das, "Streaming 360-degree videos using super-resolution," in *IEEE INFOCOM - IEEE Conf. on Computer Communications*, 2020, pp. 1977–1986.

[120] X. Hou, S. Dey, J. Zhang, and M. Budagavi, "Predictive view generation to enable mobile 360-degree and vr experiences," in *Proc. 2018 Morning Workshop on Virtual Reality and Augmented Reality Network*, ser. VR/AR Network '18, 2018, p. 20–26. [Online]. Available: https://doi.org/10.1145/3229625.3229629

[121] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan, "Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices," in *Proc. 24th Intl. Conf. on Mobile Computing and Networking*, 2018, p. 99–114.

[122] "VRMark," 2016. [Online]. Available: https://benchmarks.ul.com/vrmark

[123] "NVIDIA Frame Capture Analysis Tool," 2017. [Online]. Available: https://www.geforce.com/hardware/technology/fcat

[124] "Unigine Superposition Benchmark," 2017. [Online]. Available: https://benchmark.unigine.com/superposition

[125] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. J. Kelly, A. J. Davison, M. Luján, M. F. P. O'Boyle, G. Riley, N. Topham, and S. Furber, "Introducing slambench, a performance and accuracy benchmarking methodology for slam," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5783–5790.

[126] B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Lujan, S. Furber, A. J. Davison, P. H. J. Kelly, and M. F. P. O'Boyle, "Slambench2: Multi-objective head-to-head benchmarking for visual slam," in *2018 IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2018, pp. 3637–3644.

[127] M. Bujanca, P. Gafton, S. Saeedi, A. Nisbet, B. Bodin, M. F. P. O'Boyle, A. J. Davison, P. H. J. Kelly, G. Riley, B. Lennox, M. Luján, and S. Furber, "Slambench 3.0: Systematic automated reproducible evaluation of slam systems for robot vision challenges and scene understanding," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6351–6358.

[128] R. D. Constantine, D. F. Quiros, C. Rodda, B. C. Brown, N. B. Zerkin, and Á. Cassinelli, "Project esky: Enabling high fidelity augmented reality on an open source platform," *Companion Proceedings of the 2020 Conference on Interactive Surfaces and Spaces*, 2020.

[129] Intel, "Tracking Solutions," 2021. [Online]. Available: https://www.intelrealsense.com/tracking

[130] H. Chen, Y. Dai, H. Meng, Y. Chen, and T. Li, "Understanding the Characteristics of Mobile Augmented Reality Applications," in *Intl. Symp. on Performance Analysis of Systems and Software (ISPASS)*, April 2019, pp. 128–138.

[131] J. Yi and Y. Lee, "Heimdall: mobile gpu coordination platform for augmented reality applications," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.

Appendix: Artifact Information

A. Abstract

The artifacts consist of ILLIXR version 1, ILLIXR version 2, Monado, Godot, OpenXR test applications (AR Demo, Materials, Sponza, Platformer), analysis scripts, raw data, graphs, regression tests, and documentation.

Note that ILLIXR version 1 is the set of isolated components used for §IVB, while ILLIXR Version 2 is the same components in an integrated system used for §IVA. The artifact contains both. Where unspecified, we are referring to ILLIXR version 2.

We have made the effort to automate as much as possible of the installation process. For best results use a fresh install of Ubuntu 18.04 LTS or 20.04.

B. Artifact check-list (meta-information)

- **Program:** ILLIXR v1, ILLIXR v2, Monado, Godot, OpenXR test applications (AR Demo, Materials, Sponza, Platformer), analysis scripts.
- **Compilation:** Make 4.2, clang 10.0, CUDA 11.1, Python 3.8 (included in install scripts)
- **Data set:** See §IIIC (included in artifact) and §IIID (downloaded by program).
- **Run-time environment:** Ubuntu 18.04 + install scripts.
- **Hardware:** Any x86-64 system with an NVIDIA GPU can run, but one needs the hardware in §IIIA for exact repeatability.
- **Output:** `results/output/*` and `results/Graphs/*`.
- **Experiments:** For each hardware platform, for each app, run ILLIXR.
- **How much disk space required:** 5Gb, including downloaded datasets
- **How much time is needed to prepare workflow:** 1 hour
- **How much time is needed to complete experiments:** 20 minutes per hardware platform
- **Publicly available:** Yes, see archive URL.
- **Code licenses:** The system licensed under NCSA. Each component is licensed under one of: ElasticFusion License, NCSA, MIT, Simplified BSD, LGPL v3.0, LGPL v2.1, Boost Software License v1.0, GPL v3.0. See https://github.com/ILLIXR/ILLIXR/#licensing-structure
- **Data licenses:** Each dataset is licensed under one of: proprietary, ElasticFusion License, Creative Commons 0.
- **Archived (provide DOI):** https://doi.org/10.5281/zenodo.5523601

C. Description

1) How to access: One can find the version we used for this paper here (https://doi.org/10.5281/zenodo.5523601), and a rolling release here (https://github.com/ILLIXR/ILLIXR). We suggest using the rolling release, unless exact repeatability is desired.

2) Hardware dependencies: One can find the hardware we used for this paper in §IIIA. ILLIXR will still work on a generic x86-64 Ubuntu system with a GPU, but the results may not be exactly repeatable.

3) Software dependencies: Refer to NVIDIA's instructions for your GPU to install the proprietary NVIDIA driver and NVIDIA CUDA SDK.

```
./install_deps.sh
```

This script is *idempotent*, so there is no harm in interrupting it and running it twice. It installs basic build tools, profiling tools, Python, ROS, conda, OpenCV 3.4, Eigen, datasets, and several other resources.

See our online documentation for more details:
https://illixr.github.io/ILLIXR/getting_started

4) Data sets: The software pulls required datasets automatically when downloading software dependencies.

D. Installation

Not applicable.

E. Experiment workflow

First, we have to compile each application with Godot:

```
for app_path in OpenXR-Apps/*; do
  ./godot/bin/godot.x11.opt.tools.64
  # Import project (project.godot) (fig 1)
  # Export project (fig 2)
  # Select "Linux (Runnable)" (fig 3)
  # Select Custom Template="./godot/bin/godot.x11.opt.tools.64"
  # Select Export Path="./OpenXR-Apps/$app/bin"
  # Where app is replaced by $app shorname (e.g. "sponza")
done
```

Then, we run each application in ILLIXR V2:

```
hardware=""
# manually set to one of "jetsonlp", "jetsonhp", "desktop"
for app_path in OpenXR-Apps/*; do
  app=$(basename ${app_path})
  cmd="./ILLIXR/runner.sh ILLIXR/configs/${app}.yaml"
  ${cmd}
  nvidia-smi -q --display=UTILIZATION,POWER,TEMPERATURE \
    --loop-ms=200
  perf stat -e power/energy-cores/,power/energy-pkg/,power/energy-ram/ \
    -- ${cmd}
  mv ILLIXR/metrics results/metrics/metrics-${hardware}-${app}
done
```

To switch between high- and low-power mode on Jetson

```
sudo jetson_clocks --restore ${lp_or_hp}_mode.txt
```

F. Evaluation and expected results

Make sure to synchronize the `results/metrics/` directory from all hardware platforms onto the desktop before continuing. On the desktop, run

```
cd results/analysis
poetry run python3 main.py
```

The output from our run is available in `metrics-snapshot` and our graphs are available in `Graphs-snapshot`.

- Fig 3 is `results/Graphs/fps-jlp/jhp/desktop.pdf`
- Fig 4 is `results/Graphs/timeseries-platformer-desktop-1.pdf` and `results/Graphs/timeseries-platformer-desktop-2.pdf`
- Fig 5 is `results/Graphs/cpu-breakdown.pdf`
- Fig 6 is `results/Graphs/power-total.pdf` and `results/Graphs/power-breakdown.pdf`
- Fig 7 is `results/Graphs/mtp-platformer.pdf`
- Fig 8 is `results/Graphs/microarchitecture.pdf`

To replicate Table VI, run `./ILLIXRv1/all.sh` and see the reported statistics in stdout.

To replicate Table VII, run Visual Reprojection and Hologram in NVIDIA© Nsight™ Systems, and run Audio Encoding and Audio Decoding in Intel© VTune™ Hotspot Analysis. See `./ILLIXRv1/all.sh` for the commands to analyze.

To replicate Fig 8, run each non-trivial command of `ILLIXRv1/all.sh` in Intel© VTune™ Microarchitectural Exploration Analysis.

See https://illixr.github.io/ILLIXR/legacy/v1/ for more details on running the components of ILLIXR v1.

G. Experiment customization

Here are customizations we anticipate:

- Modify or add your own app to `ILLIXR/app/*`.
- Add your own analysis pass to `results/analysis/*.py`
- Add your own plugin to the ILLIXR system in `ILLIXR/*`. See our online documentation for details: https://illixr.github.io/ILLIXR/writing_your_plugin/

Fig. 9: Import a project in Godot

Fig. 10: Export a project in Godot

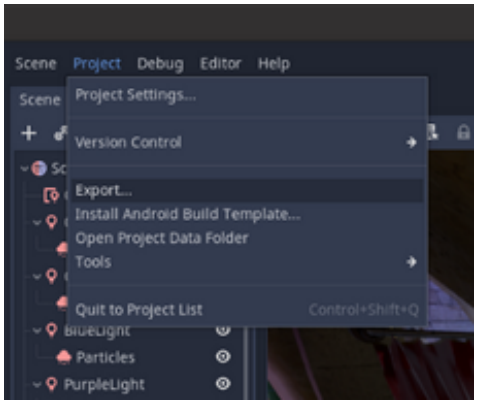


Fig. 11: Select export options in Godot