XARP: A Human-First and Agent-Ready Extended Reality Toolkit in Python

ARTHUR CAETANO, University of California, Santa Barbara, USA
RADHA KUMARAN, Saarland University, Saarland Informatics Campus, Germany
KELVIN JOU, University of California, Santa Barbara, USA
TOBIAS HÖLLERER, University of California, Santa Barbara, USA
MISHA SRA, University of California, Santa Barbara, USA

Building XR-AI research prototypes requires navigating two largely separate ecosystems. Mainstream XR development relies on C#/C++ and game engines, while AI development is centered on Python. This toolchain fragmentation slows down contributions to human-AI spatial interaction research. To broaden access to XR development in the Python ecosystem, we present XARP (XR Agent-ready Remote Procedures), a toolkit for rapid XR-AI prototyping in Python. XARP application logic runs on a Python server and controls a Unity client through WebSocket messages. This architecture enables compatibility with multiple client platforms and live reloading of application code without client redeployment. XARP is available to humans as a library and to AI agents as callable tools and through Model Context Protocol. We designed XARP through formative case studies and refined it through an early acceptance evaluation with 24 XR and AI developers and a six-week longitudinal study with two developers building an independent research project. Potential users expected the toolkit to improve their performance and facilitate development. Sustained use confirmed faster iteration and easier setup compared to conventional XR workflows, with asset-intensive and performance-critical projects emerging as the clearest limitations. Technical benchmarks show that hand and head tracking data streaming was close to the device refresh rate of 72 FPS, and that AI agents using XARP consumed 19% fewer tokens than those writing equivalent C# Unity code. Beyond broadening access to XR development, XARP reduces engineering friction in spatial computing research and opens new pathways for AI agents to participate in XR application development. XARP is open source and available at https://github.com/hal-ucsb/xarp.

CCS Concepts: • **Human-centered computing** → **Mixed / augmented reality**; **User interface toolkits**; • **Computing methodologies** → *Intelligent agents*.

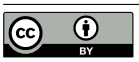
Additional Key Words and Phrases: Toolkit, XR, Python, AI, Agents, Model Context Protocol

ACM Reference Format:
Arthur Caetano, Radha Kumaran, Kelvin Jou, Tobias Höllerer, and Misha Sra. 2026. XARP: A Human-First and Agent-Ready Extended Reality Toolkit in Python. *Proc. ACM Hum.-Comput. Interact.* 10, 4, Article EICS010 (June 2026), 33 pages. https://doi.org/10.1145/3816762

1 Introduction

Human-AI spatial interaction has a long research trajectory, from early work supporting aircraft piloting [2] and machinery maintenance [15] to recent advances in context-aware guidance [35, 56,

Authors' Contact Information: Arthur Caetano, Computer Science, University of California, Santa Barbara, Santa Barbara, California, USA, caetano@ucsb.edu; Radha Kumaran, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany, kumaran@cs.uni-saarland.de; Kelvin Jou, Computer Science, University of California, Santa Barbara, Santa Barbara, California, USA, kelvinjou@ucsb.edu; Tobias Höllerer, Computer Science, University of California, Santa Barbara, Santa Barbara, California, USA, holl@cs.ucsb.edu; Misha Sra, Computer Science, University of California, Santa Barbara, Santa Barbara, California, USA, sra@ucsb.edu.



ACM 2573-0142/2026/6-ARTEICS010
https://doi.org/10.1145/3816762

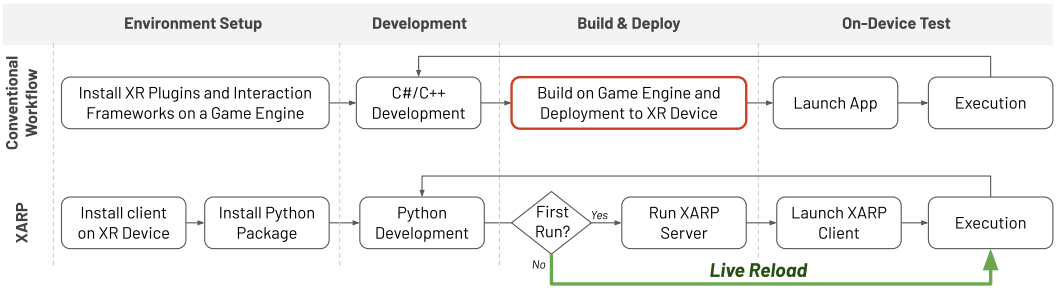


Fig. 1. Setting up XARP requires only installing a Python package and the client binary on the XR device, without requiring game engines. Development leverages tools and practices already familiar to Python developers. XARP offers live reloading, which allows app modifications to take effect on XR devices within seconds without redeployment.

65, 68] and scalable authoring [12, 22, 69]. Despite this long history and renewed interest, building XR-AI research prototypes remains a practical challenge. The XR development ecosystem relies on C#/C++ frameworks and game engines [7, 8, 18] and is largely separated from the Python ecosystem that dominates AI development [44, 57]. While XR developers can integrate AI via web APIs or on-device inference[1], AI developers using Python have limited options for developing XR interfaces [5, 9, 31]. This asymmetrical gap hinders Python developers, the majority of AI researchers [44], from contributing to research in human-AI spatial interaction, delaying the benefits these applications may provide to society.

To close this gap, we present XARP (**X**R **A**gent-ready **R**emote **P**rocedures), a toolkit for rapid XR-AI prototyping in Python. With XARP, Python developers can quickly build XR interfaces in their familiar ecosystem, enabling a workflow similar to what Streamlit[2] and Gradio[3] offer for web interfaces. Our toolkit provides high-level abstractions for common XR operations, including displaying 3D meshes and hand tracking. Internally, XARP implements a client-server architecture in which application logic executes on a Python server that controls a Unity client through commands encoded in MessagePack, an efficient binary alternative to JSON, transmitted over a WebSocket connection.

This design has several benefits. First, during development, changes to application logic on the server take effect on the XR device without redeploying the client. This live reloading feature bypasses the lengthy build-deploy cycles that slow down iteration and break creative flow in XR development. Second, the client abstracts device-specific details from XARP applications, making them portable to any XR platform with a compatible client. A single port of the Unity client to a new platform is sufficient to make all XARP applications compatible with it. Third, AI agents can use XARP as a set of callable tools or through a built-in Model Context Protocol (MCP) server [3], enabling them to control XR environments at the same abstraction level as human developers.

Our design and evaluation methodology consisted of four stages. In a formative phase, we conducted two case studies with novice XR developers to elicit design requirements from their experiences and develop an initial version of the toolkit. Next, we conducted an early acceptance evaluation through an online video walkthrough and survey with 24 XR and AI developers. In a longitudinal study, two XR-AI developers used XARP over six weeks to develop a prototype for an

[1]Unity Inference Engine: https://docs.unity3d.com/Packages/com.unity.sentis@2.1/manual/index.html, Accessed March 27, 2026.
[2]Streamlit: https://streamlit.io. Accessed April 3, 2026.
[3]Gradio: https://www.gradio.app. Accessed April 3, 2026.

```python
from xarp.server import run,
    show_qrcode_link

def hello_world(xarp, params):
    # Get connection parameters
    hi = f'Hello, {params["name"]}!'
    xarp.write(hi, title="XARP")

if __name__ == '__main__':
    # QR code with connection parameters
    show_qrcode_link(name="world")

    # run the application entrypoint
    run(hello_world)
```

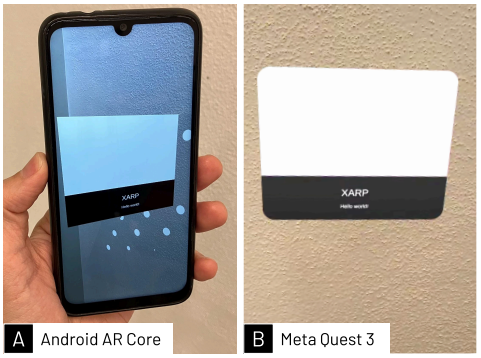


Fig. 2. XARP allows multiplatform XR development in Python. The same "hello world" application running on (a) Android ARCore and (b) Meta Quest 3, showcasing XARP's multiplatform capabilities enabled by client-server decoupling.

independent research project. Finally, we performed technical evaluations comparing streaming performance across different configurations and AI agent token consumption in code generation tasks using XARP versus Unity C#.

We demonstrate that XARP effectively enables XR interface development in Python through "how-to" scenarios with example applications [32], showcasing live reloading and multiplatform capabilities. Our human evaluations show that XR novices familiar with Python benefit from the lower entry *threshold* [50], while experienced XR developers benefit from fast prototyping. They further delineate the toolkit's *ceiling* [50], distinguishing future development roadmap items (e.g., trackable physical elements) from structural limitations (e.g., fine-grained graphics control). Technical evaluation shows that agents using XARP consume fewer tokens at the median than their C#/Unity counterparts. Hand and head streaming performance approached the device refresh rate of 72 FPS under ideal conditions.

Beyond enabling XR development for new audiences, XARP advances human-AI spatial interaction research by removing *accidental complexity* [17, 29], allowing researchers to concentrate on *essential* aspects. A shared abstraction for XR accelerates hypothesis testing and improves reproducibility through controlled cross-platform studies and systematic comparison of design alternatives. More broadly, such abstractions become increasingly critical as AI agents take on more active roles in software development and academic research [27, 28]. This article is organized in two parts. Sections 2 to 6 offer hands-on demonstrations and a toolkit description; section 7 onwards presents related work, methodology, evaluations, and findings.

2 Developing with the XARP Library

This section presents a how-to scenario followed by an example application [32], demonstrating that XARP supports a complete XR development workflow within the Python ecosystem. The how-to scenario covers environment setup, app development, live reloading, and deployment to two platforms, demonstrating the toolkit's threshold [50]. The 3D object reconstruction example application demonstrates that XARP can support common XR-AI research needs.

2.1 How-To Scenario - Main Workflow

2.1.1 Environment Setup. To get started, developers install the XARP Python package directly from the open-source repository using a package manager. Next, they download and install a

XARP client compatible with their XR device, also available in the open-source repository. This step requires no game engine and relies only on tools provided by the device vendor[4]. Prebuilt client binaries are available for Meta Quest and Android ARCore, with the Unity client source also available for developers who wish to extend or modify it. Porting XARP to additional platforms is part of the future roadmap.

2.1.2 XR Development in Python. Once setup is complete, developers write their application as an entry point function and run it by calling `xarp.server.run` as seen in the code snippet in fig. 2. This function generates a QR code encoding the server's local address and any developer-defined parameters. The client scans it to connect automatically to the server and launch the app.

2.1.3 Live Reloading. Developers can modify the code and restart the server to see the changes take effect directly on a running client. This is possible because the client briefly disconnects when the server restarts and then automatically reconnects either to the last connected address or to a new QR code. Once the client reconnects, the new application code is executed. This live-reloading mechanism takes less than 30 seconds, compared to conventional build-and-deploy processes that commonly exceed 10 minutes on first-time deploys. Tethered execution on game engines[5] also bypasses the build step but requires a Windows machine and renders via the workstation's GPU, meaning results do not reflect real on-device performance. XARP's live reloading runs directly on the target device without tethering, preserving both iteration speed and hardware fidelity. By shortening the time from modification to test on the target platform, XARP contributes to faster iterations and allows developers to stay in a creative flow. This process is depicted in fig. 1.

2.1.4 Multiplatform Execution. Another benefit of using XARP is that the same application can run on multiple XR platforms. The only necessary step is to install the appropriate client and scan the server's QR code; the application itself remains unchanged. This is possible because clients work as runtime abstractions that map commands received from the server into platform-specific functions. This approach leverages the OpenXR multiplatform capabilities offered through Unity while removing the burden of porting each application from end developers. Figure 2 shows the same app running on the Meta Quest 3 and on an Android device.

2.2 Example Application - 3D Object Reconstruction

In-situ 3D object reconstruction is an active research area in human-AI spatial interaction. Systems in this category often use image-to-3D pipelines for fabrication [62], collection [37], and remote collaboration [26]. This example demonstrates gesture-based object selection and interactive visualization that can serve as a scaffold for similar systems. Full implementation of the code snippet shown in fig. 3 can be found in the supplemental material.

2.2.1 Physical Object Selection. The example application streams hand poses and checks for a pinch gesture to trigger an RGB image capture. A vignette mask highlights the center of the image, and an image-to-3D pipeline [59] generates a 3D object reconstruction in GLB format from the masked region. Gesture detection uses a deterministic joint angle test, as shown in line 6 of the code snippet in fig. 3(a). Beyond hand tracking, XARP's streaming mechanism supports additional sensing modalities, such as depth images and head positions.

2.2.2 Interactive Visualization. The reconstructed object is displayed as a 3D element and manipulated using hand tracking and pinch gestures. XARP supports multiple element types, including

[4]Meta Quest Developer Hub: https://developers.meta.com/horizon/documentation/unity/ts-mqdh. Accessed March 27, 2026. Android Debug Bridge: https://developer.android.com/tools/adb. Accessed March 27, 2026.
[5]Unity Link: https://developers.meta.com/horizon/documentation/unity/unity-link/. Accessed April 30, 2026.

text labels, image panels, 3D primitives, GLB objects, video, and audio. fig. 3(b) shows 3D object reconstruction controlled via the right-hand palm (lines 28–30).

3 Developing with XARP Agents

Agents are autonomous systems that perceive and act in their environment on behalf of a user [41]. Agents can leverage vision-language models for visual understanding, natural language reasoning, and planning, and act through tool calling [61]. XARP serves as a set of callable tools that enable agents to perceive and act in XR environments, leveraging similar abstractions to those used by human developers.

3.1 How-To Scenario - XR Agents Workflow

3.1.1 Agent Injection. To accelerate integration with AI agents, XARP offers an out-of-the-box integration with Hugging Face smolagents, a model-agnostic agentic framework[6]. XARP is compatible with any agentic framework that can convert plain Python functions into callable tools. After following the same setup presented in section 2.1, developers include a new `agent` argument in their entry point function and execute it using the `xarp.agents.run_agent` function. XARP will automatically create an agent equipped with XR tools and inject it into the entry point function. At any point in the code, developers can delegate to agents, as seen in the code panel in fig. 4, lines 19 and 20.

[6]Hugging Face, *smolagents*: https://huggingface.co/docs/smolagents. Accessed March 27, 2026.

```
1  # start streaming hand poses
2  stream = xarp.sense(hands=True)
3  for frame in stream:
4      hands = frame["hands"]
5      # check for a right-hand pinch
6      if hands.right and pinch(hands.right):
7          # capture RGB image
8          img = xarp.image()
9          job_id = util.image_to_3d_job(img)
10         break
11 # save the 3D mesh on the device
12 mesh_asset = GLBAsset(
13     asset_key = "mesh_asset",
14     raw = util.download_glb(job_id))
15 xarp.save(mesh_asset)
16 # display the 3D mesh
17 mesh = Element(
18     key = "mesh_element",
19     asset = mesh_asset)
20 xarp.update(mesh)
21
22 # manipulate the mesh with a hand palm
23 for frame in stream:
24     hands = frame["hands"]
25     if hand := hands.right:
26         p = hand[PALM]
27         mesh.transform.position = p.position
28         mesh.transform.rotation = p.rotation
29         xarp.update(mesh)
```

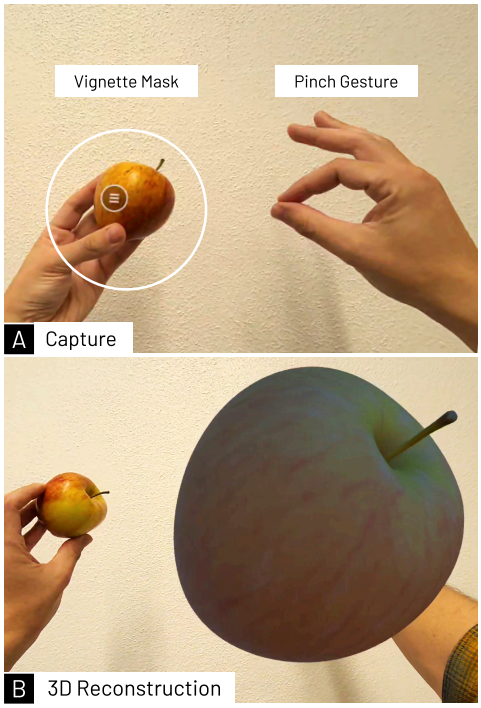


Fig. 3. 3D object reconstruction app. (a) A pinch gesture (line 6) triggers an image capture with vignette masking. (b) The reconstructed object is displayed in XR and controlled via hand tracking (lines 28-30).

```python
for frame in stream:
    hands = frame["hands"]
    if hands.left and victory(hands.left):
        if recording:
            break
        recording = True
        xarp.say("Tracking")

    # motion tracking
    if recording and hands.right:
        palm = hands.right[PALM]
        poses.append(palm)
        dumbbell.transform = Transform(
            position=palm.position,
            rotation=palm.rotation)
        xarp.update(dumbbell)

# xr agent feedback
agent.run("Analyze the palm poses during a
     standard dumbbell curl and find
    critical mistakes. Create a visual
    demonstration that combines labels,
    primitives, and colors to help users
    improve their technique. Make
    suggestions using speech.",
    additional_args=dict(poses=poses))
```

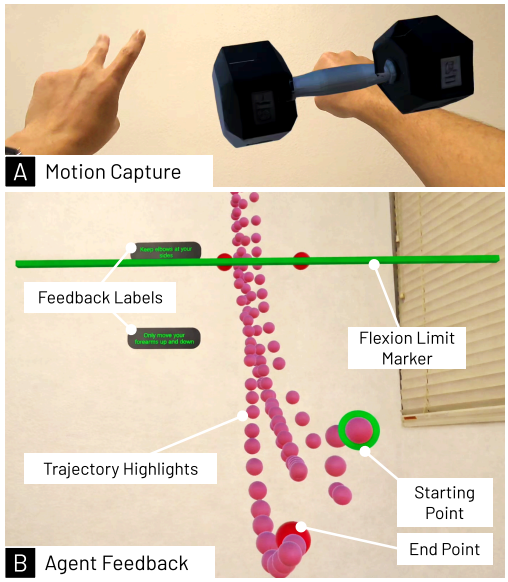


Fig. 4. Fitness coaching application demonstrating agent-generated feedback. (a) A "V" gesture initiates tracking and displays a virtual dumbbell. (b) After the exercise is complete, the agent analyzes motion data and generates multimodal feedback, including motion trajectory highlight, biceps flexion limit marker, text labels with instructions and feedback, and speech output.

3.1.2 Programmatic Control vs. Generative Delegation. XARP opens a new possibility for XR developers by combining programmatic control through the library with the option of delegating application behavior to a runtime agent. Developers must decide which requirements are well-defined enough to implement programmatically and which are better delegated to an agent that can generate code dynamically to meet situated needs at runtime. This decision is shaped by the tradeoff between control and delegation studied in human-AI interaction [25, 41].

XARP supports a full control-delegation spectrum. At one end, developers retain complete control, using XARP purely as a library with no agent involvement, as demonstrated in section 2. In the middle, selective delegation allows developers to implement the core application flow programmatically while delegating specific, context-dependent behaviors to an agent. For instance, this could involve generating personalized visual feedback based on user input, as seen in fig. 4. This approach can serve as a probe [6, 43] at the cost of introducing latency and non-deterministic behavior. At the other extreme, full delegation hands the entire application behavior to an agent, which dynamically generates and executes XR interactions in response to user needs, with minimal predetermined logic.

3.2 Example Application - Fitness Instruction and Feedback

Exercising without professional oversight increases the risk of improper technique and injury. XR-AI systems can analyze sensing data against expert guidelines to enable safer, more effective workouts [38, 39, 42]. This example shows an AI assistant delivering instructions and personalized feedback for bicep curls (fig. 4). Users initiate and stop tracking with a "V" gesture. An XR agent evaluates tracked movements and generates spatial visualizations and speech feedback.

3.2.1 Developer-Defined Interaction. Users employ a V gesture to start tracking. The same gesture stops tracking and triggers AI analysis. When hand motion tracking is active, the system displays a virtual dumbbell in the user's right hand, as shown in fig. 4(a). In this example, developers retain control over interaction and data collection.

3.2.2 Agent-Generated Feedback. The agent analyzes hand position and rotation data, identifies the most critical error using heuristics retrieved from a web search, and then generates corrective feedback through a combination of text labels, geometric primitives, and speech, as shown in fig. 4(b). The layout, elements, and content of the feedback composition fully depended on user performance and agent decisions at runtime.

4 Using the XARP MCP Server

XARP supports no-code XR-AI prototyping workflows, allowing developers and end-users to orchestrate agentic tools via natural language and MCP [3]. XARP exposes previously unavailable runtime XR capabilities to the broader MCP ecosystem of third-party agents and tool providers, enabling agents to sense and act directly on an XR device. The how-to scenario demonstrates how to launch the XARP built-in MCP server and connect it to a local large language model (LLM) chat interface alongside other MCP servers. Inspired by recent work on generative AI for XR authoring [12, 22, 69], we present a minimal example showcasing XR app modification via natural language interaction with an AI agent.

4.1 How-To Scenario - XR MCP Workflow

4.1.1 Starting the Built-in XR MCP Server. After installing XARP as described in section 2, the user launches the built-in MCP server using `xarp_mcp <name> <port>`. This command creates a XARP server and associates it with an MCP server awaiting external client connections. Calls to the MCP server are handled as XARP procedures by the XR device.

4.1.2 Connecting an Agent to the Built-in XR MCP Server. In this how-to, we use LMStudio[7], a local LLM runtime with a chat interface. After starting a new chat with a loaded model, MCP servers can be added to augment the model's capabilities. In LMStudio, this requires editing a JSON configuration file with the XARP MCP server URL. After that, the model can invoke XARP functions as needed. fig. 5 shows a simple example of a user interacting via an MCP client chat interface with a model capable of executing XR operations through the built-in XARP MCP server.

5 Developer API

Developers interact with XARP primarily through a *facade* [21] that abstracts data validation, serialization, transmission, and deserialization, while providing convenient default values. Both synchronous and asynchronous (`asyncio`) facade variants are available and share the same underlying mechanisms. Developers are not limited to the facade layer and may access lower-level abstractions directly, down to the WebSocket transport. Figure 6 shows a UML class diagram of the XARP Python API.

5.1 Scene Management

XARP's scene management system is based on two main concepts: `Elements` and `Assets`.

5.1.1 Elements. Elements are state change *commands* [21] that modify unique virtual entities on the client device. Applications transmit only property changes rather than full states, keeping network messages compact and avoiding retransmission of large assets. As elements are modified

[7]LMStudio: https://lmstudio.ai. Accessed March 29, 2026.

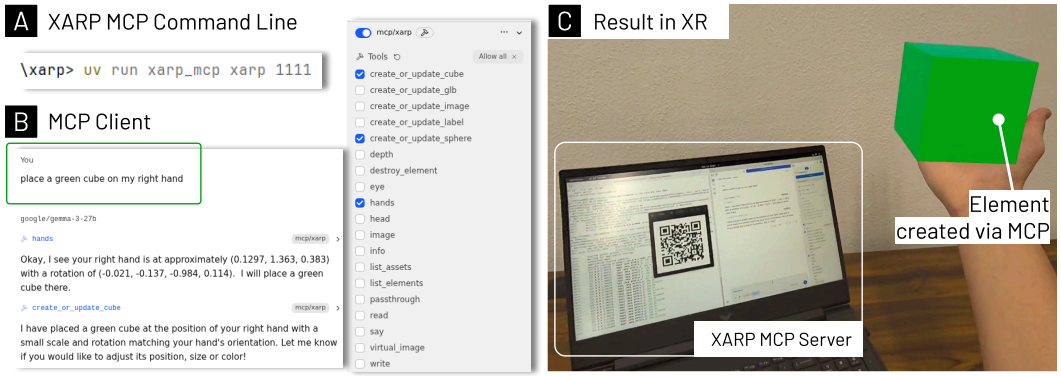


Fig. 5. XARP built-in MCP server. (a) The XARP built-in MCP server is launched via command line, forwarding incoming tool calls to a connected XR device. (b) An MCP client connects to the server using the server address, allowing agents to invoke XARP tools as needed; users interact in natural language through chat interfaces, enabling live no-code XR prototyping directly on the XR device. (c) Results are visualized live on the connected XR device.

server-side, changes do not take effect until `xarp.update(element)` is called. This method has *upsert* semantics: if an Element with the provided key exists, it is updated; otherwise, a new one is created. `xarp.list_elements` lists all existing Element keys, and `xarp.destroy_element()` destroys a specific Element or all Elements. On the client side, Elements are implemented as Unity game objects, making position, rotation, scale, and parenting transforms inherently available.

5.1.2 Assets. Assets are objects representing reusable content that can be rendered to the user. Currently, XARP supports UTF-8 text, JPG and PNG images, OGG audio, GLB 3D models, MP4 video, and 3D primitives. Each asset type has a MIME type that the client uses to decide how to render it using a *state* [21] pattern on the Element. Assets hold bytes of their raw content, but can also be manipulated in Python through their `obj` property, which exposes convenient object types for each asset type, such as Pillow[8] images or Trimesh[9] meshes.

Assets can be reused across Elements and stored persistently on the client using `xarp.save_asset`, so applications transmit an asset only once and reuse its `asset_key` to associate it with multiple Elements. All stored assets can be listed using `xarp.list_assets`, and individual assets can be deleted by `asset_key` or all at once using `xarp.destroy_asset`.

5.2 Sensing

Applications using XARP have easy access to the wide range of sensors available on XR devices.

5.2.1 Head and Eye. `xarp.head` and `xarp.eye` both return a `Pose` object containing position and orientation. `xarp.head` tracks the user's device, whether handheld or head-mounted, while `xarp.eye` tracks the RGB camera, matching the user's point of view or, for head-mounted devices, the left eye camera. These poses can be used to track user movement over time or place Elements relative to the user.

5.2.2 Hands. `xarp.hands` returns a `Hands` object with `left` and `right` attributes, each holding a tuple of joint positions and orientations. If a hand is not tracked, the respective tuple is empty.

[8]Pillow: https://pillow.readthedocs.io/en/stable. Accessed March 27, 2026.
[9]Trimesh: https://trimesh.org. Accessed March 27, 2026.

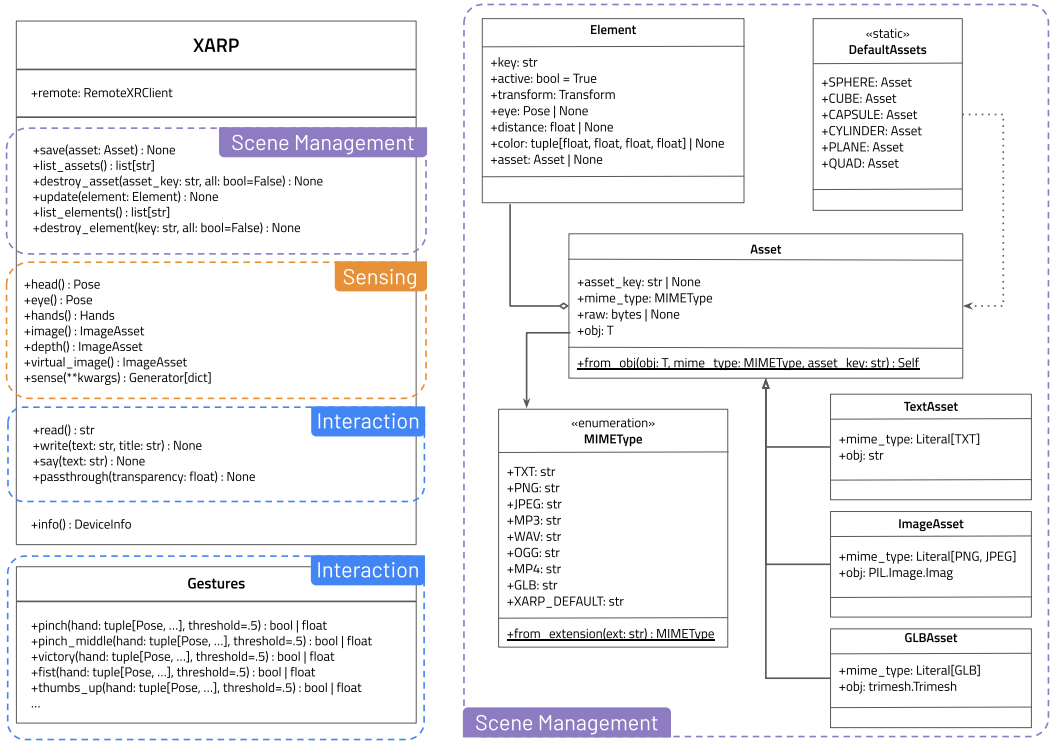


Fig. 6. UML class diagram of the XARP developer Python API.

5.2.3 RGB, Depth, and Virtual Images. XARP provides three image capture methods, each returning an `ImageAsset`: `xarp.image` for RGB, `xarp.depth` for depth, and `xarp.virtual_image` for the rendered virtual scene. Sensor intrinsics vary by device; extrinsics are available via `xarp.info` (see section 5.4).

5.2.4 Sense Stream. All sensing methods can be called individually to obtain the latest sensor reading. For high-frequency repeated readings, `xarp.sense` is recommended instead, as it requests the client to stream sensor data continuously until stopped, reducing inter-reading delay. The method accepts boolean keyword flags, one per sensing method, selecting which sensors to include in each reading, and returns a Python generator for iterating over successive readings. Once the stream is no longer needed, the application should call `close` on the generator to notify the client to stop transmitting.

The `rt` (real-time) flag controls reading behavior. With `rt=False`, every frame is yielded in sequence with minimal inter-frame delay, though readings may lag behind the current state if server-side processing is slow; this mode is suited for continuous data collection. With `rt=True`, only the latest frame is yielded and older frames are dropped, keeping memory usage minimal and readings current at the cost of potential discontinuities; this mode is preferred for interactive applications.

5.3 Interaction

Read, Write & Say. The method `xarp.read` acquires text input from the user, either through keyboard entry or dictation, depending on the capabilities of the client device. This method blocks

the caller until the client transmits the input. The method `xarp.write` displays text to the user in a panel and is suitable for presenting notifications. The method `xarp.say` accepts a string argument, synthesizes speech from it, and plays the resulting audio on the client device. This method blocks the caller until audio playback is complete.

Gestures. XARP supports a set of static hand gestures detected from hand sensor readings. Supported gestures include: pinch with thumb and any finger, thumbs up, index–middle "V", index–thumb "L", closed fist, open palm, and pointing. Each gesture function computes an internal metric that characterizes the gesture; for example, `xarp.pinch` measures the distance between the thumb tip and index tip and returns `true` if this distance falls below an empirically defined threshold, and `false` otherwise. To retrieve the raw metric value instead of the boolean result, the caller passes `threshold=None` as an argument.

Passthrough. The method `xarp.passthrough` accepts a value in the range from zero to one and controls the opacity of the virtual environment background. A value of zero renders the background fully transparent, allowing the user to see the physical environment through the XR device. A value of one renders the background fully opaque, replacing the physical environment with a solid color background.

5.4 Device Info

While XARP abstracts device-specific details to enable multiplatform development, it also provides access to device-specific information through the method `xarp.info`, which returns a data structure containing information such as the device model and intrinsic camera parameters. Because device capabilities vary and may not fully cover the XARP abstraction, this method can also be used to query which XARP methods the client supports, allowing applications to handle unsupported features gracefully and define fallbacks for broader compatibility.

5.5 Spatial Data Types

XARP provides a set of common data types for spatial operations. `Vector3` represents a point or vector in three-dimensional space. `Quaternion` represents an orientation. Both types support standard algebraic operations. `Pose` composes a position of type `Vector3` and a rotation of type `Quaternion`; a `Pose` can define a ray and compute the position of a point at a given distance along it. `Transform` extends `Pose` with an additional scale component of type `Vector3`, representing the scale of an object along each axis.

6 Architecture

This section presents the internal workings of XARP and contains relevant information for researchers and engineers aiming to modify or replicate the toolkit. Developers do not need to interact with the concepts in this section since they operate through the high-level API described in section 5. XARP implements a client-server architecture that shifts the application logic from XR devices, which is currently the mainstream approach for XR development, into a Python server. The Python application uses XARP to issue remote procedure calls and execute XR functions in a standardized client application deployed to the XR device. Figure 7 illustrates this client-server interaction.

6.1 Server

6.1.1 Session Management. The XARP server is built on FastAPI with asynchronous programming, suited for the I/O-bound workload of maintaining a WebSocket connection with the remote XR

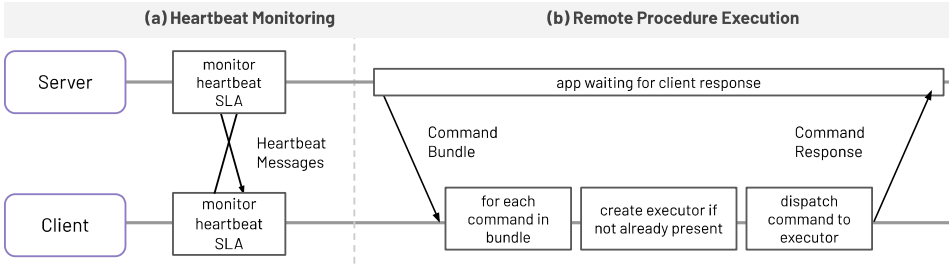


Fig. 7. XARP client-server architecture. (a) Heartbeat Monitoring: XARP uses a client-server architecture with server monitors session state via heartbeat signals to detect disconnections and terminate sessions gracefully. (b) Command Bundling: The server sends commands to the client in bundles, each potentially containing multiple commands. All commands in a bundle execute within the same client frame, with each command dispatched to a specialized executor that handles platform-specific execution on the XR device.

device. When a client connects, XARP instantiates a session object. The session manages the application lifecycle, coordinating both inbound and outbound communication over the connection.

The session runs a dedicated async task that monitors a configurable heartbeat SLA. If the client is silent beyond this threshold, the task triggers a graceful shutdown, preventing both sides from waiting indefinitely and releasing resources held by the unresponsive client. Any message exchanged during normal application flow also resets the SLA timer.

The session also runs a dedicated async task that periodically sends heartbeat messages, keeping the connection alive when the server is silent. A third async task reads from the WebSocket connection and fans out incoming messages to the rest of the system, fulfilling pending results and resetting the heartbeat SLA. Once initialized, the session is wrapped by a facade object, described in section 5, which is then passed to the application.

6.1.2 Command Bundles. Commands are grouped into bundles, the atomic unit of transmission between server and client. All commands in a bundle are guaranteed to execute within the same client frame, minimizing latency between them. Bundles are serialized in MessagePack, a binary format more efficient than JSON. The developer facade does not currently expose bundle creation directly; however, developers who need to guarantee minimal delay between two commands can bypass the facade and create bundles manually through the session.

6.1.3 Response Modes. Bundles support three response modes, illustrated in fig. 8: no-response, single-response, and stream. No-response bundles resolve immediately after being sent, requiring no acknowledgment from the client. They are suited for fire-and-forget actions such as `xarp.write` and `xarp.passthrough`, where the server does not need to await their effect. Single-response bundles create a Python future that the application awaits. When the client finishes executing the bundle, it sends a response back to the server. The session receives it in a dedicated async task, matches it to the waiting caller via a correlation identifier, and fulfills the future. Stream bundles yield results continuously through a Python generator that the application iterates. Upon receiving a stream bundle, the client maintains an internal queue and returns at most one result per frame. Either side can terminate a stream: the client sends an EOS (End-of-Stream) message, while the server closes the generator, which triggers a cancellation message to the client.

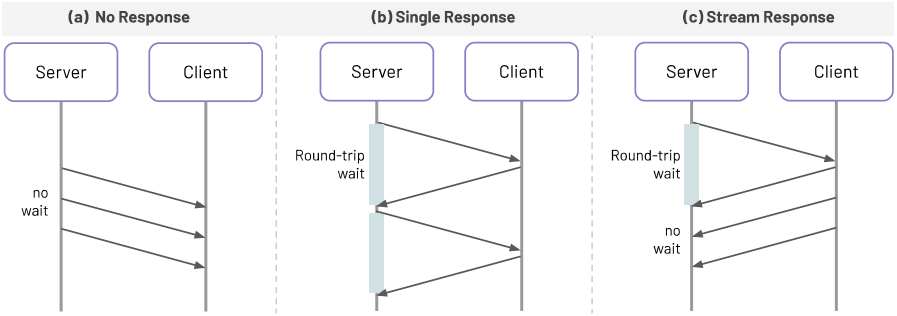


Fig. 8. Command bundles support three response modes: (a) a bundle resolves immediately after being sent, without blocking server-side logic; (b) the server waits for a single asynchronous response before continuing; or (c) a single command receives a stream of responses, reducing server wait time when data arrives sequentially.

6.2 Client

The XARP client is a Unity XR application that mirrors the server-side session. It runs asynchronous tasks that monitor the server heartbeat SLA, send periodic heartbeats, receive bundles from the server, and send back responses.

6.2.1 Command Handling. Once a bundle is received, the client session dispatches its commands to Executors, Unity GameObjects responsible for executing commands with the given parameters. Each server command maps to a corresponding Executor. For example, `xarp.read` is handled by `ReadExecutor`, which manages the platform-specific details of displaying a keyboard and returning a response once the user finishes typing. Executors are instantiated lazily on first use and reused for subsequent calls to the same command. Command-to-Executor mappings are configured through a ScriptableObject and can be remapped in the Unity Editor without code changes. Executors that depend on platform-specific features are isolated in separate C# namespaces to simplify porting. XARP currently ships a Meta Quest 3 reference implementation and a partial Android ARCore implementation. Full clients for additional platforms are on the roadmap.

6.2.2 Connection Management. Clients connect to an XARP server by scanning a QR code that encodes the server address and named parameters accessible to the application. On disconnection, the client attempts to reconnect to the last known address using a backoff-jitter strategy to avoid overloading the server. This auto-reconnection behavior enables the live reloading feature described in section 2.1. Scanning a new QR code connects to a different server, allowing users to switch between XARP applications without typing addresses.

7 Related Work

Our goal is to facilitate prototyping of XR-AI systems for developers and researchers who are fluent in Python, the dominant language of contemporary AI research [44], but unfamiliar with XR development stacks based on C#/C++ and game engines. Achieving this requires lowering the entry barriers imposed by conventional XR development workflows, while preserving the ability to modify and integrate XR interfaces with experimental systems. Prior work has addressed these requirements in isolation, but reconciling accessibility with programmatic control in a single toolkit remains an open challenge. Moreover, Python tools for XR development remain scarce, forcing developers and researchers to bridge ecosystems. XARP addresses this gap with a Python library of common XR operations, simplifying the development and deployment of XR-AI prototypes

and offering a high degree of control. These functionalities are exposed as callable tools and MCP servers for integration with AI agents.

7.1 XR Authoring Tools

Prior work has proposed XR authoring tools that make XR content creation accessible to users unfamiliar with traditional XR development, utilizing domain-specific low- or no-code abstractions to simplify content creation. However, these abstractions either remove programmatic control entirely or still require conventional XR workflows for integration with customized research systems.

DART [40] is a pioneering AR authoring toolkit that allows designers to rapidly turn storyboards into working AR experiences by specifying relationships between physical and virtual elements through the capture and replay of synchronized video and sensor data. Teachable Reality [48] is a web-based mobile AR prototyping tool that enables users to create functional tangible AR interactions with everyday objects by demonstrating them to an on-demand vision classifier and linking recognized states to AR outputs through a trigger-action interface, without programming or predefined markers. Mucho [34] is a no-code immersive prototyping tool for multimodal XR interaction that supports design-by-enacting, letting designers demonstrate inputs such as hand gestures, proximity, speech, and gaze into a timeline, add system actions during playback, and automatically generate a runtime state machine for testing the resulting interactive experience. ProInterAR [66] is an integrated visual programming platform that enables novice AR creators to build general-purpose immersive AR applications by authoring real and virtual content through an AR-HMD and scripting their interactive behaviors with Scratch-inspired blocks on a tablet, without text-based coding.

7.2 Live XR Authoring with AI Agents

Recent research has proposed live XR authoring systems that allow users to describe XR experiences in natural language to an LLM-based agent that generates and executes the corresponding code on the fly, offering an intent-based workflow similar to the one demonstrated in section 4. This approach benefits from the accessibility and expressiveness of natural language as an authoring abstraction, but does not afford direct edits to the generated artifacts. Researchers and developers integrating XR into experimental systems must export generated artifacts to a conventional XR development tool for fine-grained editing.

DreamCodeVR [22] captures spoken behavior descriptions, generates corresponding C# Unity scripts, and executes them live in a running VR application. LLMR [12] coordinates a pipeline of specialized agents for planning, scene analysis, skill grounding, code generation, and self-inspection to let users construct and modify interactive mixed reality scenes with natural language prompts at runtime. Ostaad [1] is a conversational agent that allows non-programmers to design interactive VR scenes through embodied prompts. AgentAR [69] is a tool-augmented LLM agent that supports in-situ authoring of end-to-end interactive AR applications from natural language inputs, and can produce systems across several application categories. LLMER [10] follows a similar pattern, but instead of code, it generates structured JSON that is dispatched to preconstructed runtime modules. VRCopilot [67] and ImaginateAR [33] enable immersive authoring through multimodal natural-language interaction with generative AI, targeting indoor furniture layout co-creation and on-site outdoor 3D asset generation respectively, with less emphasis on interactivity.

7.3 XR Toolkits with Game Engine Workflows

Prior research has proposed toolkits built on top of game-engine workflows that reduce the accidental complexity of XR development [29] without sacrificing developer control, typically by

introducing abstractions that simplify laborious steps or accelerate the construction of specific classes of XR systems. These toolkits, however, still require familiarity with conventional game-engine workflows, effectively decoupling AI researchers from their typical stack.

Ubiq [19] is an open-source Unity toolkit for social MR, providing room management, voice, avatars, and component-based messaging. Ubiq-Exp [58] builds on this by providing tools for remote and distributed MR studies. It integrates specialized features such as distributed logging, record-and-replay, and in-world questionnaires. AUIT [14] is a Unity extension that lets developers compose adaptive XR UIs from declarative objectives such as visibility and reachability, with a multi-objective solver resolving conflicts between them in real time. XRtic [49] is a hardware-software prototyping toolkit comprising modular SMA cloth actuators, a controller bus system, and a Unity C# scripting interface for triggering clothing deformations synchronized with virtual content events. XRSpotlight [18] is a Unity Editor panel that automatically abstracts existing XR scene implementations into natural-language trigger-action rules, helping novice developers inspect interaction logic, identify similar examples, and transfer interaction behavior across scenes and toolkits through copy-paste. ConnectVR [11] is a no-code trigger–action authoring Unity plugin that enables non-technical artists and storytellers to create agent-based VR narratives by connecting player actions to virtual character and object behaviors, adding interactivity to scenes assembled in the Unity Editor. CoCreatAR [52] is an asymmetric collaborative AR authoring system that connects an in-situ mobile app for on-site testing, spatial capture, and contextual annotation with a Unity-based ex-situ editor, allowing remote developers to refine site-specific outdoor AR experiences using real-time feedback from the target environment. MCP-Unity [64] connects an LLM-based agent to the Unity Editor via MCP for project authoring, which can later be built and deployed to the target XR device.

7.4 Emerging XR Ecosystems

Recent research has proposed XR toolkits targeting emerging XR ecosystems, offering alternatives beyond the mainstream C#/C++ game-engine stack. Despite their flexibility, these alternatives lack robust integration with the Python ecosystem and AI agents.

WebXR[10] is a W3C browser API that gives JavaScript applications direct access to XR hardware, including headset poses, motion controllers, and stereo rendering, without a native runtime or game engine. p5.xr [24] extends p5.js [45], a JavaScript creative coding library, to run sketches in AR and VR via WebXR, lowering the entry barrier for developers familiar with JavaScript but not 3D development. XR Blocks [36] introduces a "Reality Model" that abstracts users, environments, and agents into first-class primitives, leveraging native AI execution (via TensorFlow and Gemini) to unify spatial perception and interaction. In contrast, IWSDK [46] employs a high-performance ECS architecture integrated with an MCP-connected AI agent, facilitating autonomous, closed-loop development through real-time code generation, scene graph inspection, and self-debugging. While these WebXR toolkits bypass the traditional build-deploy cycle, they remain isolated from the Python ecosystem central to contemporary AI research.

NVIDIA CloudXR [53] is a GPU-accelerated streaming platform that decouples XR rendering from display by running OpenXR applications on high-performance servers and streaming encoded video to lightweight client devices over standard networks. CloudXR Runtime acts as an OpenXR-compliant runtime on Windows and Linux, capturing stereo frames, encoding them via hardware-accelerated codecs, and transmitting them alongside audio and tracking data to native clients on several XR devices and web browsers via CloudXR.js. Server applications are built against the

[10]WebXR: https://www.w3.org/TR/webxr/. Accessed May 10, 2026.

OpenXR C API or integrated through game engines. CloudXR does not target Python developers and does not expose XR functionality to AI agents.

Godot[11] is an open-source game engine that offers XR support through Godot XR Tools [23], a library of reusable XR interaction components including locomotion, object pickup, pointer, and snap zones. Two experimental plugins, py4godot [5] and godot-python-extension [31], add Python as a scripting language inside the editor. This combination, however, does not guarantee portability with the Python ecosystem, as packages with binary dependencies may be incompatible with standalone XR hardware. Development depends on the build-deploy cycle of conventional game engine workflows, and XR functionality is not natively exposed to AI agents.

PyOpenXR [9] provides Python bindings for the OpenXR API and supports tethered VR prototyping on Windows and Linux. It closely exposes the OpenXR C API in Python, allowing developers to manage extension negotiation and integrate custom render loops. This level of control is useful, but it requires platform-specific expertise that is typically outside the scope of AI research workflows. Its reliance on a PC-based runtime also limits standalone XR deployment, and it does not provide built-in support for AI agent integration.

SnapNCode [63] takes a different approach. Rather than targeting the XR runtime layer, it is a browser-based spatial IDE that lets programmers capture physical object states from a live video stream and embed them directly into Python code as images, with object-state changes triggering attached code snippets opportunistically via mobile or headset cameras. SnapNCode primarily contributes a reduced "perceptual distance" between physical state and source code. However, its spatial functionality is limited to 2D video input and object-proximity operators such as `In()` and `On()`, which are applied within 2D bounding boxes; consequently, it lacks support for true 3D spatial reasoning or immersive hardware rendering.

8 Formative Case Studies

To gain first-hand insight into novice XR developers' experience and directly observe pain points faced by this audience, we conducted two case studies with high-school seniors as part of a summer research program at our home university. They built XR-AI systems using a mix of conventional XR workflows and an early version of XARP.

8.1 AI Fencing Coach

This case study produced an AI agent that provides sabre fencing feedback on off-the-line movements based on biomechanical features elicited with fencing coaches [30]. The system depended on full-body tracking beyond what the HMD available for this project offered. To overcome this, the student considered integrating a mounted smartphone for tracking full-body motion in a third-person perspective. The complexity of integrating that external device and synchronizing the incoming data was incompatible with the project's time frame, and they pivoted to analysis of a public dataset of prerecorded bouts.

Impact on the toolkit design. This case study shaped two design choices in XARP, both aligned with Raffaillac and Huot [54]'s recommendations for research-oriented toolkits. Following the *duplicate* principle, which recommends against assuming that elements like XR devices are singular, XARP allows multiple devices to connect to the same server and easily exchange data through simple shared variables. Following the *accumulate* principle, which calls for retaining interaction data in interoperable formats, XARP data types extended Pydantic models [12] for easy data validation and serialization.

[11]Godot Engine: https://godotengine.org/. Accessed May 10, 2026.
[12]Pydantic Models: https://docs.pydantic.dev/latest/concepts/models/. Accessed March 27, 2026.

8.2 Drone Assistant

This case study produced an AI assistant for outdoor pick-and-place embodied as a virtual drone [51]. The system required integration with drone flight simulators, AI agents, and, in future work, hardware platforms. A key challenge was synchronizing the drone's virtual avatar, simulation states, and AI agent perception-action cycle.

Impact on the toolkit design. This case study motivated two design choices. First, we strive to keep XARP flexible without enforcing a particular system integration architecture. This allows developers to leverage the rich Python ecosystem to integrate with external services in any form, including third-party libraries and SDKs, bindings for robotics and simulation platforms, and AI integration beyond API calls through direct access to AI models, callable tools, and MCP. Second, we allow state changes of persistent entities across layers (e.g., XR client, application logic, and simulations) using command objects that carry state deltas, implemented as Elements presented in section 5.

9 Early Acceptance Evaluation

After refining XARP based on insights from the formative process, we conducted an early acceptance evaluation to gather initial feedback from the target audience and assess the toolkit's acceptance potential. This evaluation consisted of a video walkthrough demonstration [13, 32, 55] followed by a UTAUT survey [60] and a custom survey based on Ledo et al. [32]. The videos covered both the fundamental concepts of XARP and practical aspects such as setup process and example implementations using key features of the toolkit, including tracking, speech interaction, AI agents for runtime generative behavior, and MCP integration. This method mirrors how developers often engage with new tools, by watching tutorials before deciding whether to try them in practice. This study was approved by our local institutional review board (UCSB #39-25-0548)

9.1 Participants

We recruited 24 adults (ages 18–44; 14 male, 10 female, 0 non-binary) through social media and online AI-XR developer communities. All participants had at least one year of experience with AI, XR, or both. Participants were distributed across five countries, with varied educational and professional backgrounds. Most participants were trained in Computer Science, AI, or Engineering. XR experience varied from none (n=3) to more than five years (n=5). Participants most commonly reported experience with Meta Quest, iOS, and HoloLens. Unity with AR Foundation and the Mixed Reality Toolkit were frequently mentioned as development tools. In terms of AI experience, two participants reported no prior experience, while five reported more than five years of experience. Popular platforms included the OpenAI API, PyTorch, and TensorFlow.

9.2 Measures

We assessed XARP's *potential acceptance* using the **Unified Theory of Acceptance and Use of Technology (UTAUT)** scale [60] with 7-point Likert items. We excluded Social Influence because the study focused on individual early acceptance after a walkthrough rather than organizational adoption pressure. We complemented the survey with open-ended items asking why participants would adopt or avoid XARP. Figure 9 (a) shows the proportions of UTAUT responses.

To assess the *perceived value* of the toolkit, we designed a custom survey based on the **HCI Toolkit Goals (HCITG)** framework compiled by Ledo et al. [32], as no standardized instrument for this framework exists. We mapped each toolkit goal into a survey construct, further separating creative exploration from solution replication. Each HCITG survey construct was measured with three to six 7-point Likert-scale items, including at least one reverse-coded item. The survey covered

the following constructs: *Reducing Authoring Time*, *Creating Paths of Least Resistance*, *Empowering New Audiences*, *Integrating with Current Practices and Infrastructure*, *Enabling Replication of Existing Solutions*, and *Enabling Creative Exploration*. Our customized HCITG survey is available in the supplementary material. Figure 9 (b) shows the proportions of HCITG responses.

We measured participant *comprehension* of the walkthrough videos to ensure their judgments of the toolkit were based on informed understanding. Each video was followed by three to four comprehension-check items, along with a confidence check: "The video provided enough evidence for me to answer the questions in this section." Our post-study questionnaire included the question "What questions do you still have about the toolkit?" to identify common comprehension gaps.

9.3 Procedure

The study was conducted fully online and asynchronously using Qualtrics[13], allowing participants to pause and resume at any point. The median completion time, including pauses, was 45 minutes, and the fastest participant finished the study in 25 minutes, consistent with our previous in-person pilot.

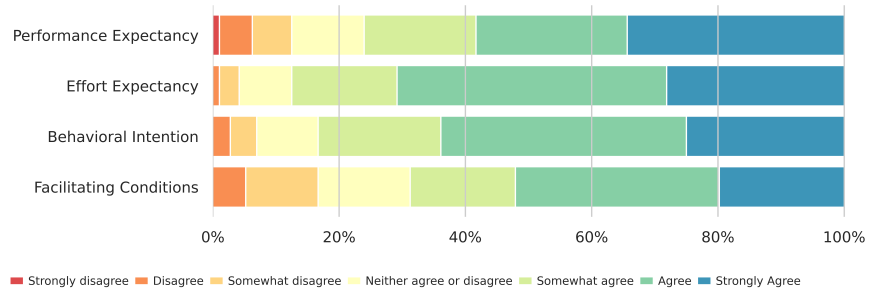
After providing informed consent and demographic information, participants were screened to ensure they were at least 18 years old and had at least one year of experience with AI or XR development. Eligible participants proceeded to watch seven short videos demonstrating XARP, presented in a fixed didactic order: *Overview*, *Setup*, *Visual Q&A*, *Virtual Drone*, *AI Agent*, *MCP Integration*, and *Modifying XARP*. Each video was approximately two minutes long, totaling around 15 minutes of content. Except for the overview video, which introduced core concepts and the toolkit architecture, each segment consisted of a brief introduction, a step-by-step walkthrough, and an example application. After each video, participants answered comprehension-check questions. Videos could be replayed while responding, as our goal was to ensure informed exposure to the toolkit and its capabilities, rather than assessing learning outcomes. Following the videos, participants completed the UTAUT and HCITG surveys. In this final section, they provided reasons for potentially adopting or avoiding XARP in their future AI-XR projects and could ask additional questions about the toolkit.

9.4 Data Analysis

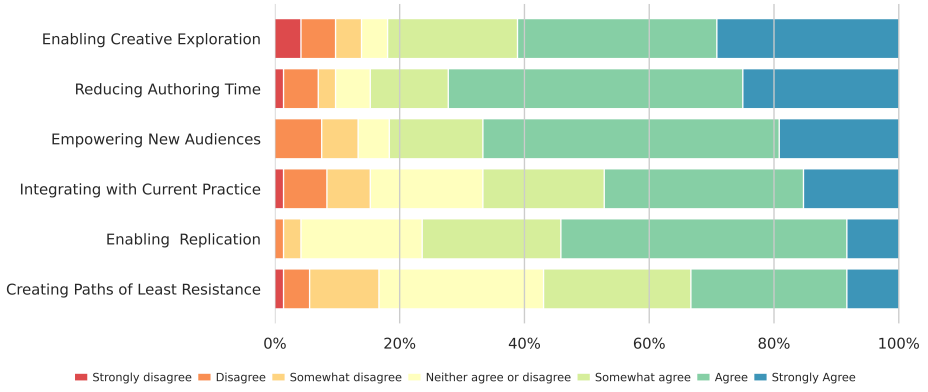
All participants met the screening criteria for quality of responses: consistency on reverse-coded items, a minimum completion time of 20 minutes (15 minutes for video content plus 5 minutes for responses), open-text entries with more than a single word, and above-chance performance on comprehension checks. We verified the reliability of items within the same construct with McDonald's ω before aggregating them into composite 0-1 scores. We computed participant-level construct composites and used two-sided Wilcoxon signed-rank tests with Holm correction to assess deviations from the neutral midpoint.

Guided by the theoretical structure of UTAUT, we used multiple linear regression to examine associations between UTAUT constructs, moderators (age, gender, expertise), and *Behavioral Intention*. We analyzed participants' responses about reasons for potentially adopting or avoiding XARP, as well as their open-ended questions, using a rapid variant of the Framework Method [20]. This approach provides a transparent, systematic way to code and chart small corpora, enabling clear comparison across responses while minimizing analytic overhead.

[13]Qualtrics: https://www.qualtrics.com/. Accessed March 27, 2026.



(a)



(b)

Fig. 9. Distribution of item-level Likert responses to the four UTAUT constructs (a) and six HCITG constructs (b). Participant-level construct composites were significantly above the neutral midpoint.

9.5 Results

9.5.1 Potential Acceptance. Reliability of the UTAUT constructs was confirmed by McDonald's ω: *Performance Expectancy* ($\omega = 0.948,\ 4$ items); *Effort Expectancy* ($\omega = 0.928,\ 4$ items); *Facilitating Conditions* ($\omega = 0.928,\ 4$ items); *Behavioral Intention* ($\omega = 0.916,\ 3$ items). Participant-level UTAUT construct composites were significantly above the neutral midpoint in two-sided Wilcoxon signed-rank tests with Holm correction, with large rank-biserial effect sizes: *Performance Expectancy* ($M = 0.78$, $Mdn = 0.84$, $W = 9.0$, $p_{\text{Holm}} < 0.001$, $r_{\text{rb}} = 0.96$), *Effort Expectancy* ($M = 0.83$, $Mdn = 0.84$, $W = 2.0$, $p_{\text{Holm}} < 0.001$, $r_{\text{rb}} = 0.99$), *Facilitating Conditions* ($M = 0.74$, $Mdn = 0.75$, $W = 1.5$, $p_{\text{Holm}} < 0.001$, $r_{\text{rb}} = 0.99$), and *Behavioral Intention* ($M = 0.80$, $Mdn = 0.86$, $W = 4.0$, $p_{\text{Holm}} < 0.001$, $r_{\text{rb}} = 0.97$). The mean normalized UTAUT score across participants was $M = 0.78$, $SD = 0.12$, 95% $CI\,[0.73,\ 0.84]$, and the mean *Behavioral Intention* score was $M = 0.80$, $SD = 0.15$, 95% $CI\,[0.73,\ 0.86]$. Figure 10 (a) shows boxplots of the UTAUT scores.

Following the UTAUT framework, we fitted multiple linear regressions with and without moderators. However, due to the small sample size, all the regression analyses remain exploratory. A multiple linear regression without the moderators accounted for 67% of the variance in *Behavioral Intention* ($R^2 = 0.67$, $R^2_{\text{adj}} = 0.62$, $F = 6.06$, $p < 0.01$). In this model, *Performance Expectancy*

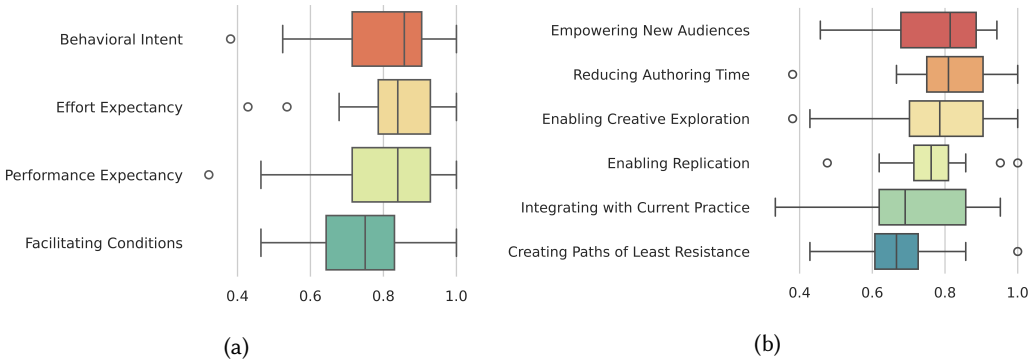


Fig. 10. (a) UTAUT Score: *Performance Expectancy* was a significant predictor of *Behavioral Intention* on an exploratory multiple linear regression (b) Distribution of HCITG Scores.

showed the strongest association with *Behavioral Intention* ($\beta = 0.67,\ p = 0.014$), whereas *Facilitating Conditions* ($\beta = 0.10,\ p = 0.48$) and *Effort Expectancy* ($\beta = 0.12,\ p = 0.48$) were not significant. A multiple linear regression including age, gender, and expertise as moderators accounted for 83% of the variance in *Behavioral Intention*, but a substantially lower R^2_{adj} ($R^2 = 0.83$, $R^2_{\text{adj}} = 0.57,\ F = 2.83,\ p = 0.060$) and did not show reliable associations with *Behavioral Intention*.

9.5.2 HCI Toolkit Goals. Some constructs of our customized HCITG scale showed acceptable reliability ($\omega \geq 0.7$): *Enabling Creative Exploration* ($\omega = 0.830,\ 3$ items); *Empowering New Audiences* ($\omega = 0.828,\ 5$ items); *Integrating with Current Practices and Infrastructure* ($\omega = 0.786,\ 3$ items); *Enabling Replication of Existing Solutions* ($\omega = 0.784,\ 3$ items). However, others fell below this threshold: *Reducing Authoring Time* ($\omega = 0.687,\ 3$ items); *Creating Paths of Least Resistance* ($\omega = 0.655,\ 3$ items). Participant-level HCITG construct composites were significantly above the neutral midpoint in two-sided Wilcoxon signed-rank tests with Holm correction. All effects were large by rank-biserial correlation: *Reducing Authoring Time* ($M = 0.81, Mdn = 0.81, W = 1.0, p_{\text{Holm}} < 0.001$, $r_{\text{rb}} = 0.99$), *Creating Paths of Least Resistance* ($M = 0.68, Mdn = 0.67, W = 6.0, p_{\text{Holm}} < 0.001$, $r_{\text{rb}} = 0.96$), *Empowering New Audiences* ($M = 0.78, Mdn = 0.81, W = 1.0, p_{\text{Holm}} < 0.001$, $r_{\text{rb}} = 0.99$), *Integrating with Current Practices and Infrastructure* ($M = 0.72, Mdn = 0.69, W = 8.0$, $p_{\text{Holm}} < 0.001, r_{\text{rb}} = 0.95$), *Enabling Replication of Existing Solutions* ($M = 0.76, Mdn = 0.76$, $W = 1.0, p_{\text{Holm}} < 0.001, r_{\text{rb}} = 0.99$), and *Enabling Creative Exploration* ($M = 0.78, Mdn = 0.79$, $W = 4.0, p_{\text{Holm}} < 0.001, r_{\text{rb}} = 0.97$). The mean HCITG score across participants was $M = 0.75$, $SD = 0.09$, 95% $CI\,[0.71,\ 0.79]$. Figure 10 (b) shows boxplots of the HCITG scores.

We examined marginal associations between the HCITG constructs and *Behavioral Intention*. All six constructs showed positive Pearson and Spearman correlations with *Behavioral Intention*. The strongest Pearson correlations were observed for *Reducing Authoring Time* ($r = 0.59$), *Enabling Replication of Existing Solutions* ($r = 0.57$), *Enabling Creative Exploration* ($r = 0.55$), and *Integrating with Current Practices and Infrastructure* ($r = 0.55$). The strongest Spearman correlations were observed for *Creating Paths of Least Resistance* ($\rho = 0.52$) and *Integrating with Current Practices and Infrastructure* ($\rho = 0.52$), followed by *Enabling Replication of Existing Solutions* ($\rho = 0.42$) and *Reducing Authoring Time* ($\rho = 0.36$). Overall, the correlations suggest that all HCITG constructs were positively associated with *Behavioral Intention*, with *Reducing Authoring Time* showing the strongest linear association and *Creating Paths of Least Resistance* and *Current Practices* showing the strongest monotonic associations.

9.5.3 Toolkit Comprehension. The average *comprehension* score across participants was $M = 0.92$, $SD = 0.09$, 95% CI [0.88, 0.96]. The numerically highest walkthrough comprehension score was obtained on the tasks *Toolkit Overview* ($M = 0.97$, $SD = 0.13$) and *Setup* ($M = 0.97$, $SD = 0.09$). The numerically lowest video walkthrough comprehension score was obtained on the tasks *MCP Integration* ($M = 0.83$, $SD = 0.28$) and *Modifying XARP* ($M = 0.85$, $SD = 0.27$). The *self-rated confidence on UTAUT answers* average across participants was $M = 0.80$, $SD = 0.15$, 95% CI [0.73, 0.87]. The *self-rated confidence on HCITG answers* average across participants was $M = 0.80$, $SD = 0.12$, 95% CI [0.75, 0.86].

A rapid Framework analysis [20] of participant responses to the question, "What other questions do you have that were not answered in the videos?" produced seven main FAQ topics. The most frequent topic was *Compatibility & Integration* (42%), e.g., "Is XARP compatible with other AR headsets beyond the Quest?" and "Does the [toolkit] have anything for integrating with Unity Sentis and on-device AI models?" Another frequent topic was *Limitations* (15%), e.g., "(...) In what circumstances have you found the built-in commands, LLM prompting, or motion/data tracking to be unreliable?"

9.5.4 Qualitative Basis for Behavioral Intention. Consistent with the theoretical structure of UTAUT, our regression analysis suggests that *Behavioral Intention* was associated with *Performance Expectancy*. Qualitative accounts corroborate this finding and further reveal reasons aligned with *Reducing Authoring Time*, with participants identifying three main sources of perceived gains:

- **High-level abstractions**: Participants valued the simplicity and clarity of XARP's functions and its ability to hide unnecessary technical detail. P3 highlighted "organization/simplicity of commands, and ease of integration with external tools," adding that they prefer to "use Python when possible." P7 commented that XARP can "expose the least low-level details to the users who are not experts in AI or XR." P9 expected that they "would not get bogged down with reading up on various concepts or worrying about smaller details" when using XARP.
- **Live reloading**: Several participants associated performance gains with the ability to code server-side apps without redeploying clients. P2 remarked that with XARP, there is "no need to build and rebuild Unity Projects." Similarly, P24 appreciated that XARP "allows developers to avoid constantly pushing new builds in the development of an XR app."
- **AI agent integration**: Participants also saw performance benefits in XARP's AI agent integration. P6 valued how "the agent understands my need and my current status." P17 emphasized time savings for prototyping: "the main value add would be the amount of time it would save to come up with quick prototypes (...) I think a strength is the flexibility to swap out models and plug-and-play MCPs."

When questioned about reasons why they might avoid using XARP, participants identified reasons aligned with *Facilitating Conditions*:

- **Compatibility**: Participants expressed concerns about device and client support. P2 noted "Client compatibility with different headsets," while P4 remarked, "I probably would not use it in my current role, because I do not have access to a Meta Quest device." Similarly, P24 emphasized, "I would not use the platform if I did not own a Meta Quest." These concerns highlight that the toolkit's portability does not benefit users whose target platform we have not yet ported the client to. We currently support Meta Quest and Android ARCore, but new clients can be added through the Unity multiplatform pipeline and inherit compatibility with any XARP server application.

- **Learning Curve and Documentation**: Several participants highlighted the need for more learning support and documentation. P3 commented, "If there were more fitting alternative technologies in certain circumstances (or technologies with more thorough documentation or other similar benefits), then I might be preferential to those. I don't think I have enough knowledge/info to respond specifically to the downsides of this platform." Others pointed to the difficulty of learning and using the system effectively: P6 mentioned "The sharp learning curve or the long running time it takes," P10 said they "Need more time to understand and learn how to use it and design more functions," and P14 noted "Difficulty in learning the language syntax." We plan to create comprehensive documentation for XARP once it is expanded to include the full planned range of functionalities, which we anticipate will ease the concerns we observed regarding learning support.
- **Performance**: Although *Performance Expectancy* was the strongest predictor of *Behavioral Intention*, participants also noted constraints related to computational performance. P7 observed, "If the token usage is too high, it might stop users from creating more creative or complex tasks, (because) not too many users have a locally deployed GPT-oss model, or they could not afford the cost of using token APIs." Likewise, P12 raised concerns about performance and reliability: "Performance for real-time tasks, not sure how an AI coding agent will be able to optimize an XR app to run in real-time or improve performance. I feel that client-server interaction is unreliable." P23 was concerned that the toolkit "performs I/O and rendering in a (...) potentially non-performant way."

10 Longitudinal Evaluation

After incorporating participant feedback from the early acceptance evaluation, we sought opportunities to observe XARP in use on independent research projects. To disseminate the toolkit and prospect early adopters, we conducted a demonstration open to students in the laboratory of one of the authors. Following the demonstration, a researcher not among the authors of this work decided to adopt XARP in their own project, which was originally intended to use Unity as its development platform. This opportunity allowed us to conduct a longitudinal study examining toolkit utilization in an ecologically valid setting, where the objectives, timeline, and personnel were established entirely independently of our evaluation goals.

We followed a team of two researchers over six weeks as they developed their independent research project with XARP. Through repeated checkpoints and a final interview, we captured both initial experiences and evolving perceptions of the toolkit. Our analysis focused on identifying barriers to initial adoption, limitations, and patterns of toolkit appropriation through sustained use [4]. Participants provided informed consent and reviewed this section to ensure anonymization and nondisclosure of details related to their unpublished research. This evaluation was approved by our local institutional review board (UCSB #39-25-0548).

10.1 Participants

The study was conducted with a team of two XR-AI developers. PL1 is an undergraduate in Computer Science (18–24, F) with less than one year of Unity development experience on Meta Quest 3, and AI experience deploying 3D generation models [59] and using AI APIs (Hugging Face, OpenAI). PL2 is a PhD student (18–24, M) with one to three years of XR development across Meta Quest, HTC Vive, and Magic Leap using Unity, and experience training and deploying AI models using tools such as Hugging Face, TensorFlow, and Ollama. Both are close collaborators at our university and not authors of this article.

10.2 Case Study Overview

Because the research project used as the case study in this evaluation remains unpublished at the time of writing, we refrain from reporting its motivation or hypotheses. Instead, we focus on how XARP was used in the project. Participants used the toolkit to build a system that enabled users to capture egocentric RGB images through hand gestures. These images are processed by a vision-language model and presented back to the user alongside textual and audio information in a structured layout. We report on six weeks of case study development.

10.3 Procedure

Our data collection methodology consisted of impromptu meetings with participants throughout six weeks, with touchpoints adapted to their natural workflow rather than imposed at fixed intervals [4]. On the first day, participants received the Python package, Unity client binary for Meta Quest, and an installation tutorial covering content equivalent to section 2. One author was available during business hours throughout the study to provide technical support when needed. These interactions forced us to refine our documentation, a concern raised in the early adoption study, and shaped the content presented in section 5 and section 6. Identified bugs were resolved iteratively, including issues with gesture detection and element hierarchy construction. After six weeks of sustained engagement with the toolkit, the supporting author conducted a one-hour semi-structured interview with participants (script available in the supplemental materials). The interview was transcribed and complemented with field notes, covering topics including workflow evolution, comparisons with Unity, pain points encountered during development, and feature requests, with particular emphasis on initial setup experiences and limitations that emerged through sustained use.

10.4 Data Analysis

The same author who supported participants during this study conducted the data analysis. They started by compiling notes of their touchpoints with participants. Interview transcriptions were organized using the interviewer's notes as an initial outline. These data sources were combined in a unified dataset and indexed by week, facilitating the identification of changes in behavior over time [4].

10.5 Results

10.5.1 Weeks 1–2 - Gaining Momentum. Both participants found XARP easier to learn than Unity. PL1, who had recently set up an XR project in Unity, called XARP setup *"way easier"*, recalling that their *"first build took 20 minutes"* in Unity, with troubleshooting the headset connection taking *"an hour."* With XARP, initial deployment happened within minutes. PL2, with more extensive XR experience, observed that XARP bypasses common Unity hurdles such as installing XR packages, resolving version conflicts, and configuring headset connections. Both followed a tutorial similar to section 2 and began developing without difficulty.

Participants established a workflow that mirrored their existing Python practices: modify code, run, observe results on the headset, repeat. They debugged using print statements and error messages, with occasional client restarts to ensure a fresh client state. PL2 highlighted the convenience of connecting via QR code, eliminating the need for cables.

10.5.2 Weeks 3–5 - Limitations & Workarounds. As participants tackled more complex development, they encountered limitations addressed through workarounds or feature requests. PL2 requested standard UI elements (buttons, toggles, sliders) and visual effects such as line renderers and hand occlusion, drawing on features available in other XR toolkits. Both participants wanted built-in

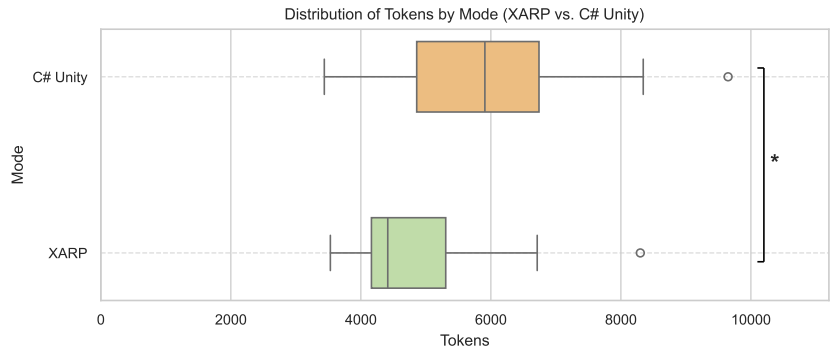


Fig. 11. Token consumption of AI agents for the baseline C#/Unity compared to XARP. Agents consumed significantly more tokens in the baseline condition than with XARP.

poke/grab interactions and co-location support. These requests are currently part of our development roadmap.

Access to source code proved valuable: participants extended XARP rather than waiting for fixes. PL2 modified `xarp.update` to accept lists after discovering a performance bottleneck from single-object updates. They customized the look-and-feel of their interface using Python libraries to generate images displayed as `ImageAssets`. The need for more thorough documentation—particularly conceptual guides and task-oriented tutorials—also became apparent as source reading alone failed to scale.

10.5.3 Week 6 - Reflecting on Sustained Use. Prolonged headset wear became fatiguing, prompting requests for a desktop simulator for small headset-free tests. Participants reflected on ecosystem tradeoffs. Python's AI ecosystem was a clear asset for their vision-language model integration, while the absence of game engine asset marketplaces was a recognized but non-critical limitation in their project. They identified that asset-intensive or performance-critical projects might be a poor fit for XARP.

Both participants responded affirmatively when asked whether they would choose XARP again, citing easier comprehension, faster startup, better iteration speed, and natural AI integration. Neither wished they had started with Unity despite the limitations encountered. They recommended XARP most strongly for beginners and developers unfamiliar with Unity. PL2 added that experienced developers might benefit from XARP for rapid prototyping.

11 Agent Performance Benchmark

Addressing questions about token consumption raised in our early acceptance evaluation, we conducted a technical benchmark comparing the token consumption of AI agents using our toolkit and conventional C# Unity in a series of XR application generation tasks. Results indicate XARP enabled a 19% median reduction in token consumption.

11.1 Apparatus

We implemented two autonomous code-generation agents using Devstral Small 2 and the Hugging Face Smolagents framework. Devstral Small 2 was selected for its strong coding performance relative to parameter count [47], vision capability, open-weight availability, and training on tool use, making it suitable for local execution and runtime integration with XARP.

One agent generated XR applications in Python using XARP, while the baseline agent generated functionally equivalent applications in C# for Unity. The context window size was fixed at 40,960 tokens, and the sampling temperature was set to 0.7. All experiments were executed on a single NVIDIA RTX 3090 GPU. Agents were implemented as XARP XR agents as seen in section 3. To avoid network noise, the XARP client ran in the Unity editor during the experiments.

11.2 Tasks

We compared conditions across 14 tasks, eight designed by us and six taken from prior work [12]. Tasks require interactive behavior and the manipulation of primitive geometries. For example, spawning a sphere when the user enters a cubic trigger volume and animating a grid of spheres using a time-varying sinusoidal function. All tasks were constructed to be solvable in both conditions and to require comparable levels of control flow, spatial reasoning, and state management. Tasks are available in the supplemental material.

11.3 Procedure

Each agent completed all 14 tasks independently. For each task, the agent was allowed to iteratively generate code until a correct, functioning application was produced. All trials resulted in successful task completion by design, ensuring that comparisons of token consumption reflected differences in generation efficiency rather than task success or failure. For each trial, we recorded the total number of tokens consumed across the complete interaction required to generate a working XR application.

11.4 Analysis and Results

We conducted a paired-samples comparison of token usage across the 14 tasks ($n = 14$ pairs). The baseline condition consumed a mean of 7,000 tokens ($SD = 4,384$; $Mdn = 5,908$) per task, while XARP consumed a mean of 5,283 tokens ($SD = 2,199$; $Mdn = 4,414$) (see fig. 11). In 10 of the 14 tasks, XARP required fewer tokens than the baseline. Because paired differences were not normally distributed (Shapiro-Wilk; $SW = 0.86$, $p = 0.034$), we used the Wilcoxon signed-rank test rather than a paired t-test. The Wilcoxon signed-rank test supported the directional hypothesis that XARP would consume fewer tokens than the baseline ($W = 21$, $p = 0.025$, one-tailed). The rank-biserial correlation of $r_{\text{rb}} = 0.60$ indicates a large effect [16]. The median token reduction was 1,081 tokens per task, corresponding to a 19% decrease from baseline. fig. 11 shows the token distributions in both conditions.

12 Sensing Benchmark

The early acceptance evaluation raised concerns about I/O performance over the network, as noted by P12 and P23. To address these concerns, we evaluated the sensing performance of XARP by measuring throughput in frames per second (FPS) across combinations of sensing modalities under repeated single-response calls and streaming response mode. Experiments were conducted in two hardware setups. The *local editor* setup consisted of a Unity editor connected to a localhost server, offering a baseline free of network effects. The *on-device* setup ran a Meta Quest 3 connected via a university Wi-Fi network (2 Gbps upload and download) during business hours, approximating typical operational conditions for both network bandwidth and on-device processing. The Meta Quest 3 device operated on a 72 Hz display refresh rate.

12.1 Procedure

Each combination of the modalities `head`, `hands`, and `image` was evaluated under both response modes and platforms, yielding $7 \times 2 \times 2 = 28$ configurations in total. For each configuration, three

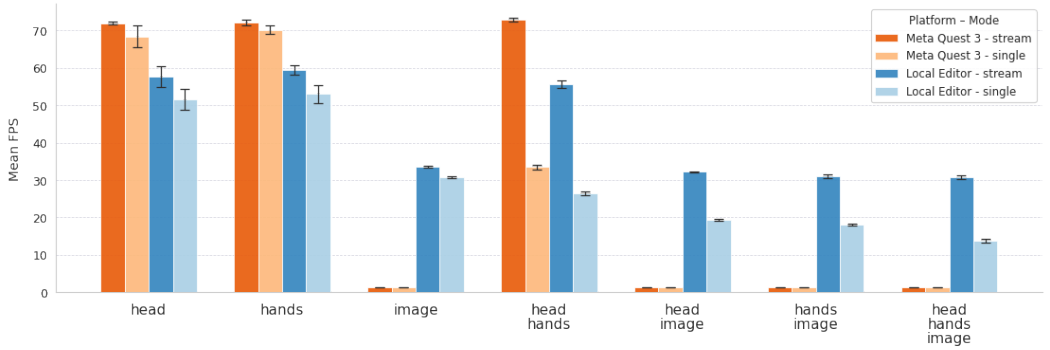


Fig. 12. Throughput performance across modality combinations and execution modes (FPS, higher is better). Stream mode consistently outperforms single-response mode. Image capture shows substantial hardware dependency, with throughput dropping from 33.6 FPS in the local editor to 1.4 FPS on Meta Quest 3.

independent runs were performed. In each run, the first 10 frames were discarded as warm-up; the measurement window then consisted of $N = 1000$ consecutive frame intervals. Timestamps were recorded immediately after a data frame was decoded and available to the XARP application. Throughput was computed as $\text{FPS} = N/(t_N - t_0)$, where t_0 and t_N are the timestamps of the first and last measured frames, respectively. All reported values are the mean across the three runs. The procedure was identical for both the local editor and on-device setups.

12.2 Results

Throughput (FPS; higher is better) was higher in the stream response mode across data modality combinations and hardware conditions. The streaming advantage compared to repeated single responses was most pronounced when reading `head` and `hands` together on Meta Quest 3, achieving 66.7 FPS while single-response mode dropped to 33.4 FPS, representing a 50% reduction in throughput when switching from streaming to single responses. In the local editor, `image` streaming reached 33.6 FPS, but there was a drastic drop on Meta Quest 3 to 1.36 FPS. Any modality combination including `image` was bounded by this rate on Meta Quest 3, ranging between 1.31 and 1.42 FPS regardless of mode or additional modalities. The highest measured throughput was 72.1 FPS for streaming `hands` data on Meta Quest 3 compared to 59.4 FPS on the local editor. Figure 12 shows the mean throughput for each configuration.

13 Discussion

13.1 Lowering Barriers in XR Development

The main goal of XARP is to broaden access to XR development, which is currently limited to those familiar with game engines and C#/C++ coding. By exposing XR operations through a Python library, XARP allows Python developers, including a considerable number of AI researchers, to create XR interfaces using their familiar workflows. The example applications in Sections 2 to 4 demonstrate how XARP effectively enables XR development in Python. Our early acceptance study showed that XR and AI developers expected XARP to reduce their effort and accelerate their development process. The XR abstraction available in the toolkit, the decoupling of XR device and application logic, and the live reloading feature all contributed to this early acceptance. These patterns were observed again after sustained six-week usage as reported in our longitudinal study.

The expected benefits of XARP extend beyond novices to experienced XR developers who can use XARP for rapid prototyping. The emerging MCP ecosystem offers a common integration layer to quickly compose off-the-shelf AI capabilities for no-code prototyping. The ability to delegate parts of application behavior to AI agents who can make decisions and use XR capabilities through XARP represents another emerging design approach to tackle requirements uncertainty at runtime.

13.2 Toolkit Ceiling

The formative and evaluative processes revealed both temporary and strict ceilings [50] of XARP. The temporary ceiling encompasses missing features that can be addressed through planned development, while the strict ceiling limitations on what the toolkit can achieve are derived from core design tradeoffs.

13.2.1 Temporary Ceiling: Roadmap Items. Sustained use in the longitudinal study identified a roadmap of features that are technically feasible but not yet implemented. Participant requests included standard UI widgets, visual effects such as line renderers, and particles. Similarly, the toolkit does not yet support lighting, shader authoring, particle systems, and animations. The need for more comprehensive documentation also became apparent. These temporary limitations reflect the current maturity level of a research prototype rather than fundamental constraints. In the timespan of this article, several limitations were addressed, for example, the inclusion of stream response mode that addresses concerns about I/O efficiency identified in the early acceptance evaluation.

13.2.2 Strict Ceiling: Architectural Limitations. Our evaluations revealed use cases that have a poor fit with XARP due to core toolkit design decisions. While participants acknowledged faster on-device iteration compared to traditional XR workflows, prolonged headset use became tiring, leading them to request a desktop simulator for headset-free testing. Participants also reflected on ecosystem tradeoffs. Gaining access to the Python ecosystem for AI integration meant losing access to game engine asset marketplaces and fine-grained control over rendering. As a result, asset-intensive projects might not benefit from XARP. Similarly, performance-critical applications requiring lower-level optimizations are poor fits for the toolkit. Composing large numbers of assets without a visual editor becomes challenging.

Sensing benchmarks revealed a throughput gap between the local editor and the Meta Quest 3 during image streaming, likely caused by images being transmitted as uncompressed 1280x720 bitmaps. Adding JPG compression to captured images brought frame rates above 30 FPS in a preliminary image streaming performance test, although thorough benchmarking remains future work. XARP's streaming mechanism was designed for small payloads and lacks throttling or compression. For long-term multimedia streaming, alternatives such as WebRTC may be considered.

13.3 Design Reflection

Throughout the development of this toolkit, we identified a series of design tradeoffs that may be relevant for other researchers and interactive system engineers. We describe those tradeoffs and offer possible alternatives when applicable.

13.3.1 Human-First vs. Agent-Ready. Since the initial stages, XARP has prioritized keeping a human-friendly public API while allowing AI agents to share that same API. As we implemented examples and benchmarks with AI agents and MCP, we noticed a tension between these goals. Rich human-oriented APIs rely on complex data models such as `Element` and implicit conventions such as iterating over a sense stream as the main application loop. Agent-oriented APIs favor simpler primitives with less polymorphic behavior, in line with best practices for callable tools.

MCP is still maturing, and its evolving patterns introduce additional pressure. Current reference implementations such as FastMCP[14] may require MCP-specific data types such as `MCPImage`[15] that are not ergonomic for human use. Allowing fast-changing third-party interfaces to leak into the toolkit risks eroding the Human-First principle and undermining XARP's primary utility as a Python-based XR development platform. Our strategy is to monitor how MCP APIs and patterns evolve before committing to deeper integration. Looking further ahead, as AI coding agents continue to improve, they may reach a level where they can consume human-oriented APIs as naturally as human developers do, at which point the distinction between human-first and agent-ready design may disappear.

13.3.2 Portability. XARP's portability rests on the separation between server-side application logic and the platform-specific client. The client is implemented in Unity, with prebuilt binaries currently available for Meta Quest and Android ARCore. Supporting a new platform requires porting only the client, a manageable task given Unity and OpenXR's broad device coverage; Magic Leap, Pico, and HoloLens are all candidates. Once ported, all existing XARP applications become compatible with that platform without further modification.

This strategy offers a considerable portability range, but has limitations. Porting to a less capable device, such as one without 6DOF head tracking, leaves the corresponding API methods present but non-functional at the device level, pulling API design toward the lowest common denominator of all target platforms. The symmetric pressure occurs when a target platform exposes a unique capability absent from the XARP API. Including it breaks compatibility with devices that lack that feature and leads to API bloat. Platforms without a Unity pipeline introduce a third, independent limitation. Supporting them requires a full client reimplementation, and only the Python server code carries over. Implementation differences between the original and the reimplemented client may produce behavioral divergence that is difficult to anticipate.

The right approach to managing this tension is not yet settled. Two plausible models are OpenXR's extension mechanism, where applications query which extensions the underlying runtime supports at initialization and enable only a compatible subset, and AR Foundation's subsystem descriptor model, which maps capabilities to platforms explicitly so that developers can query feature availability at startup and implement their own fallback logic accordingly.

13.3.3 Development Mental Model. The server-client split requires experienced XR developers to adjust their mental model in several ways. XARP Elements resemble Unity GameObjects and are, in fact, implemented as such on the client, but their semantics differ. `xarp.update` applies state deltas rather than replacing full entity state, and each update triggers a network call rather than a direct state mutation rendered in the next frame. Update frequency, therefore, carries a cost that developers must account for explicitly. A related shift concerns the application loop. Unity hides per-frame logic behind lifecycle methods such as `GameObject.Update`. In XARP, an application may run sequentially or structure its main loop explicitly around a sense stream, keeping control flow visible and central. A third difference is that XARP does not impose an entity-component architecture, leaving developers free to structure their codebase as it grows. For novice XR developers with no prior Unity experience, none of these are adjustments. They are simply defaults, which may be one reason novices might be more willing to use XARP.

Token count is a standard efficiency metric in the agent evaluation literature, used to quantify computational cost and verbosity per task. Because tokenizers differ across models, raw counts are

[14]FastMCP: https://gofastmcp.com. Accessed March 27, 2026.
[15]MCPImage FastMCP: https://gofastmcp.com/python-sdk/fastmcp-utilities-types. Accessed March 27, 2026.

not directly comparable between them. Within a single model, however, token count is consistent and reproducible, and this within-model comparison is the basis of our benchmark.

14 Limitations

Internal Validity. The acceptance evaluation mirrors how developers approach a new tool, but the absence of direct usage constrains the data to self-reported perceptions rather than observed behavior. The sample (n=24) is small, making most of its analysis exploratory. The participants were nonetheless diverse and representative of the target audience, spanning developers and researchers with varying XR backgrounds. The longitudinal study was conducted by researchers independent from this paper, though institutional proximity and experimenter expectation effects may still have influenced outcomes. The study ran for six weeks and included both a novice and an expert participant, covering a meaningful range of experience levels over a non-trivial duration.

External Validity. Technical evaluations were performed with a Meta Quest 3 and Devstral Small 2 on a university network, and results may differ with other devices, models, or deployment conditions. Both configurations are nonetheless accessible and widely used, which strengthens the practical relevance of the findings. The early acceptance study relied on recruitment through the authors' social media channels, which may have introduced self-selection bias toward participants already familiar with or favorably disposed to the work. The longitudinal participants were drawn from a single institution, which may limit generalizability to professional developers or broader industry contexts.

Construct Validity. Social Influence was excluded from the UTAUT model because the study targeted individual early acceptance following a walkthrough session rather than organizational adoption, making the factor conceptually inappropriate rather than merely inconvenient. The exclusion nonetheless limits direct comparability with prior UTAUT studies. Token count was used as a proxy for agent efficiency, a standard measure in the agent evaluation literature for capturing verbosity and computational cost per task. It is internally consistent when comparing runs of the same model, though it does not generalize across models due to differences in tokenization. The HCITG instrument was a customized survey without prior psychometric validation. It demonstrated reasonable internal consistency (McDonald's omega above 0.7) and included reverse-coded items to reduce acquiescence bias.

15 Future Work

In addition to implementing the prioritized missing features gathered throughout the development and evaluation process of XARP, we are currently working on several directions that extend the toolkit's impact and sustainability.

Game Engine Interoperability. Although XARP offers a self-contained development workflow, we plan to support interoperability with game engines. The primary motivation is access to engine-level assets, XR toolkits, and legacy projects not covered by the XARP API. Because XARP is built on Unity and OpenXR, the natural first step is integrating the XARP client into existing Unity projects, enabling gradual migration or coexistence of Unity and XARP components within the same application. This would address an ecosystem tradeoff identified in our evaluations, allowing developers to draw on rich game engine assets while benefiting from XARP's rapid prototyping workflow.

Open-Source Community. XARP is an open-source project, with both the Unity client and the Python backend library publicly available. Community contributions are central to its long-term trajectory. They accelerate roadmap delivery, surface diverse use cases, expand platform support,

and keep the toolkit current with advances in AI and XR hardware. Broader participation also improves documentation grounded in real-world usage, and extends the reach of XARP's core goal of lowering the barrier to XR development.

Supporting Spatial Human-AI Interaction Research. Beyond lowering barriers to XR development, XARP has broader implications for spatial human-AI interaction research. A shared abstraction layer for XR accelerates hypothesis testing through faster prototyping and improves the replicability of research systems, as studies built on XARP can be reproduced across platforms without reimplementing device-specific infrastructure. The multiplatform architecture also supports more rigorous experimental methodology by enabling systematic comparison of design alternatives across devices. As new XR hardware capabilities are integrated into the library, the research community gains immediate access to them without rebuilding low-level integrations. Because XARP decouples research systems from any single vendor's SDK or commercial roadmap, studies remain reproducible even as device ecosystems evolve or specific platforms are discontinued. XARP additionally brings AI researchers into the spatial computing community, potentially catalyzing novel interaction paradigms. Its agent-ready design positions the toolkit as a platform for exploring human-AI collaboration in spatial contexts, from runtime behavior generation to automated XR application testing.

16 Conclusion

We presented XARP, a Python toolkit for prototyping XR-AI systems. Application logic runs on a Python server that controls a Unity client over WebSocket, bypassing game engines and supporting multiple devices through a single client port. The same abstraction is exposed to AI agents as callable tools and through the Model Context Protocol. An acceptance survey with 24 developers and a six-week longitudinal deployment indicated faster setup and iteration than conventional XR workflows, while identifying asset-intensive and performance-critical projects as the toolkit's ceiling. In benchmarks, AI agents using XARP consumed 19% fewer tokens at the median than agents producing equivalent C#/Unity code. XARP is open source and available at https://github.com/hal-ucsb/xarp.

Acknowledgments

We thank the U.S. National Science Foundation for supporting this research through the Early CAREER Award 2023 no. 2240133. We especially thank Yunhao Luo, Irene Li, Avinash Nargund, Celine Zhao, and the students of the UCSB Winter 2026 CS291I class, whose input helped shape XARP.

References

[1] Setareh Aghel Manesh, Tianyi Zhang, Yuki Onishi, Kotaro Hara, Scott Bateman, Jiannan Li, and Anthony Tang. 2024. How people prompt generative ai to create interactive vr scenes. In *Proceedings of the 2024 ACM Designing Interactive Systems Conference*. 2319–2340. https://doi.org/10.1145/3643834.3661547

[2] Earl A. Alluisi. 1986. Human Factors Implications of Project Forecast II: Aircrew Combat Mission Enhancement (ACME) Technology. *Proceedings of the Human Factors Society Annual Meeting* 30, 1 (1986), 45–47. doi:10.1177/154193128603000111

[3] Anthropic. 2024. *Introducing the Model Context Protocol.* https://www.anthropic.com/news/model-context-protocol

[4] Åsa Audulv, Thomas Westergren, Mette Spliid Ludvigsen, Mona Kyndi Pedersen, Liv Fegran, Elisabeth O. C. Hall, Hanne Aagaard, Nastasja Robstad, and Åsa Kneck. 2023. Time and change: a typology for presenting research findings in qualitative longitudinal research. *BMC Medical Research Methodology* 23, 1 (2023), 284. doi:10.1186/s12874-023-02105-1

[5] Niklas Bergmann. 2026. py4godot: Python for Godot. https://godotengine.org/asset-library/asset/3234. Version 4.5-alpha14.

[6] Kirsten Boehner, Janet Vertesi, Phoebe Sengers, and Paul Dourish. 2007. How HCI interprets the probes. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1077–1086.

[7] Ingo Börsting, Markus Heikamp, Marc Hesenius, Wilhelm Koop, and Volker Gruhn. 2022. Software Engineering for Augmented Reality - A Research Agenda. *Proc. ACM Hum.-Comput. Interact.* 6, EICS, Article 155 (June 2022), 34 pages. doi:10.1145/3532205

[8] Dibyendu Brinto Bose and Chris Brown. 2024. An Empirical Study on Current Practices and Challenges of Core AR/VR Developers. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering Workshops* (Sacramento, CA, USA) *(ASEW '24)*. Association for Computing Machinery, New York, NY, USA, 233–238. doi:10.1145/3691621.3694956

[9] Christopher M. Bruns. 2022. pyopenxr: Python Bindings for OpenXR. https://cmbruns.github.io/pyopenxr/.

[10] Jiangong Chen, Xiaoyi Wu, Tian Lan, and Bin Li. 2025. LLMER: Crafting Interactive Extended Reality Worlds with JSON Data Generated by Large Language Models. *IEEE Transactions on Visualization and Computer Graphics* 31, 5 (2025), 2715–2724. doi:10.1109/TVCG.2025.3549549

[11] Mengyu Chen, Marko Peljhan, and Misha Sra. 2024. ConnectVR: A Trigger-Action Interface for Creating Agent-based Interactive VR Stories. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*. 286–297. doi:10.1109/VR58804.2024.00051

[12] Fernanda De La Torre, Cathy Mengying Fang, Han Huang, Andrzej Banburski-Fahey, Judith Amores Fernandez, and Jaron Lanier. 2024. LLMR: Real-time Prompting of Interactive Worlds using Large Language Models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 600, 22 pages. doi:10.1145/3613904.3642579

[13] João Marcelo Evangelista Belo, Anna Maria Feit, Tiare Feuchtner, and Kaj Grønbæk. 2021. XRgonomics: Facilitating the Creation of Ergonomic 3D Interfaces. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 290, 11 pages. doi:10.1145/3411764.3445349

[14] João Marcelo Evangelista Belo, Mathias N. Lystbæk, Anna Maria Feit, Ken Pfeuffer, Peter Kán, Antti Oulasvirta, and Kaj Grønbæk. 2022. AUIT – the Adaptive User Interfaces Toolkit for Designing XR Applications. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology* (Bend, OR, USA) *(UIST '22)*. Association for Computing Machinery, New York, NY, USA, Article 48, 16 pages. doi:10.1145/3526113.3545651

[15] Steven Feiner, Blair Macintyre, and Dorée Seligmann. 1993. Knowledge-based augmented reality. *Commun. ACM* 36, 7 (July 1993), 53–62. doi:10.1145/159544.159587

[16] Fernanda Fiel Peres. 2026. Effect sizes for nonparametric tests. *Biochemia Medica* 36, 1 (feb 2026). doi:10.11613/BM.2026.010101

[17] Steven D. Fraser, Frederick P. Brooks, Martin Fowler, Ricardo Lopez, Aki Namioka, Linda Northrop, David Lorge Parnas, and David Thomas. 2007. "No silver bullet" reloaded: retrospective on "essence and accidents of software engineering". In *Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion* (Montreal, Quebec, Canada) *(OOPSLA '07)*. Association for Computing Machinery, New York, NY, USA, 1026–1030. doi:10.1145/1297846.1297973

[18] Vittoria Frau, Lucio Davide Spano, Valentino Artizzu, and Michael Nebeling. 2023. XRSpotlight: Example-based Programming of XR Interactions using a Rule-based Approach. *Proc. ACM Hum.-Comput. Interact.* 7, EICS, Article 185 (June 2023), 28 pages. doi:10.1145/3593237

[19] Sebastian J Friston, Ben J Congdon, David Swapp, Lisa Izzouzi, Klara Brandstätter, Daniel Archer, Otto Olkkonen, Felix Johannes Thiel, and Anthony Steed. 2021. Ubiq: A System to Build Flexible Social Virtual Reality Experiences. In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology* (Osaka, Japan) *(VRST '21)*. Association for Computing Machinery, New York, NY, USA, Article 6, 11 pages. doi:10.1145/3489849.3489871

[20] Nicola K Gale, Gemma Heath, Elaine Cameron, Sabina Rashid, and Sabi Redwood. 2013. Using the framework method for the analysis of qualitative data in multi-disciplinary health research. *BMC medical research methodology* 13, 1 (2013), 117.

[21] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., USA.

[22] Daniele Giunchi, Nels Numan, Elia Gatti, and Anthony Steed. 2024. DreamCodeVR: Towards Democratizing Behavior Design in Virtual Reality with Speech-Driven Programming. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*. 579–589. doi:10.1109/VR58804.2024.00078

[23] GodotVR contributors. 2026. Godot XR Tools. https://godotvr.github.io/godot-xr-tools/.

[24] Stalgia Grigg. 2019. p5.xr: A Library for Running p5.js Sketches in AR/VR Using WebXR. https://p5xr.org. Accessed: 2026-05-10.

[25] Eric Horvitz. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Pittsburgh, Pennsylvania, USA) *(CHI '99)*. Association for Computing Machinery, New

York, NY, USA, 159–166. doi:10.1145/302979.303030

[26] Xincheng Huang, Michael Yin, Ziyi Xia, and Robert Xiao. 2024. VirtualNexus: Enhancing 360-Degree Video AR/VR Collaboration with Environment Cutouts and Virtual Replicas. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 55, 12 pages. doi:10.1145/3654777.3676377

[27] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=VTF8yNQM66

[28] Tanisha Jowsey, Virginia Braun, Victoria Clarke, Deborah Lupton, and Michelle Fine. 2025. We Reject the Use of Generative Artificial Intelligence for Reflexive Qualitative Research. *Qualitative Inquiry* 0, 0 (2025), 10778004251401851. doi:10.1177/10778004251401851

[29] Frederick Phillips Brooks Jr. 1987. No Silver Bullet Essence and Accidents of Software Engineering. *Computer* 20, 4 (1987), 10–19. doi:10.1109/MC.1987.1663532

[30] Nathan Kessler, Arthur Caetano, and Misha Sra. 2025. Towards AI Agents for Sabre Fencing Coaching in Extended Reality. In *Proceedings of the 2025 ACM Symposium on Spatial User Interaction (SUI '25)*. Association for Computing Machinery, New York, NY, USA, Article 44, 3 pages. doi:10.1145/3694907.3765956

[31] Mai Lavelle. 2023. godot-python-extension. https://godot-python-extension.readthedocs.io/. Accessed: 2026-05-10.

[32] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–17. doi:10.1145/3173574.3173610

[33] Jaewook Lee, Filippo Aleotti, Diego Mazala, Guillermo Garcia-Hernando, Sara Vicente, Oliver James Johnston, Isabel Kraus-Liang, Jakub Powierza, Donghoon Shin, Jon E. Froehlich, Gabriel Brostow, and Jessica Van Brummelen. 2025. ImaginateAR: AI-Assisted In-Situ Authoring in Augmented Reality. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology (UIST '25)*. Association for Computing Machinery, New York, NY, USA, Article 52, 21 pages. doi:10.1145/3746059.3747635

[34] Germán Leiva, Vittoria Frau, Unnikrishnan Radhakrishnan, and Mille Skovhus Lunding. 2025. Mucho: A Timeline-Based Immersive Tool for Prototyping Interactive Multimodal XR Experiences. *Proc. ACM Hum.-Comput. Interact.* 9, 4, Article EICS008 (June 2025), 20 pages. doi:10.1145/3734185

[35] Chenyi Li, Guande Wu, Gromit Yeuk-Yin Chan, Dishita Gdi Turakhia, Sonia Castelo Quispe, Dong Li, Leslie Welch, Claudio Silva, and Jing Qian. 2025. Satori: Towards Proactive AR Assistant with Belief-Desire-Intention User Modeling. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 1229, 24 pages. doi:10.1145/3706598.3714188

[36] David Li, Nels Numan, Xun Qian, Yanhe Chen, Zhongyi Zhou, Evgenii Alekseev, Geonsun Lee, Alex Cooper, Min Xia, Scott Chung, Jeremy Nelson, Xiuxiu Yuan, Jolica Dias, Tim Bettridge, Benjamin Hersh, Michelle Huynh, Konrad Piascik, Ricardo Cabello, David Kim, and Ruofei Du. 2025. XR Blocks: Accelerating Human-centered AI + XR Innovation. arXiv:2509.25504 [cs.HC] https://arxiv.org/abs/2509.25504

[37] Zisu Li, Jiawei Li, Zeyu Xiong, Shumeng Zhang, Faraz Faruqi, Stefanie Mueller, Chen Liang, Xiaojuan Ma, and Mingming Fan. 2025. InteRecon: Towards Reconstructing Interactivity of Personal Memorable Items in Mixed Reality. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 833, 19 pages. doi:10.1145/3706598.3713882

[38] Chen-Chieh Liao, Zhihao Yu, and Hideki Koike. 2025. ShiftingGolf: Gross Motor Skill Correction Using Redirection in VR. *IEEE Transactions on Visualization and Computer Graphics* 31, 5 (2025), 3429–3439. doi:10.1109/TVCG.2025.3549170

[39] Dizhi Ma, Xiyun Hu, Jingyu Shi, Mayank Patel, Rahul Jain, Ziyi Liu, Zhengzhe Zhu, and Karthik Ramani. 2024. avaTTAR: Table Tennis Stroke Training with Embodied and Detached Visualization in Augmented Reality. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 35, 16 pages. doi:10.1145/3654777.3676400

[40] Blair MacIntyre, Maribeth Gandy, Steven Dow, and Jay David Bolter. 2004. DART: a toolkit for rapid design exploration of augmented reality experiences. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology* (Santa Fe, NM, USA) *(UIST '04)*. Association for Computing Machinery, New York, NY, USA, 197–206. doi:10.1145/1029632.1029669

[41] Pattie Maes, Ben Shneiderman, and Jim Miller. 1997. Intelligent software agents vs. user-controlled direct manipulation: a debate. In *CHI '97 Extended Abstracts on Human Factors in Computing Systems* (Atlanta, Georgia) *(CHI EA '97)*. Association for Computing Machinery, New York, NY, USA, 105–106. doi:10.1145/1120212.1120281

[42] Sara Mandic, Rhys Tracy, and Misha Sra. 2023. ARFit: Pose-based Exercise Feedback with Mobile AR. In *Proceedings of the 2023 ACM Symposium on Spatial User Interaction* (Sydney, NSW, Australia) *(SUI '23)*. Association for Computing Machinery, New York, NY, USA, Article 45, 3 pages. doi:10.1145/3607822.3618008

[43] Nikolas Martelaro and Wendy Ju. 2017. The Needfinding Machine. In *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction* (Vienna, Austria) *(HRI '17)*. Association for Computing Machinery, New York, NY, USA, 355–356. doi:10.1145/3029798.3034811

[44] Nestor Maslej, Loredana Fattorini, Raymond Perrault, Yolanda Gil, Vanessa Parli, Njenga Kariuki, Emily Capstick, Anka Reuel, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Russell Wald, Tobi Walsh, Armin Hamrah, Lapo Santarlasci, Julia Betts Lotufo, Alexandra Rome, Andrew Shi, and Sukrut Oak. 2025. *Artificial Intelligence Index Report 2025*. Technical Report. Stanford Institute for Human-Centered Artificial Intelligence (HAI). https://hai.stanford.edu/assets/files/hai_ai_index_report_2025.pdf Eighth edition of the AI Index Report.

[45] Lauren Lee McCarthy. 2014. p5.js: A JavaScript Library for Creative Coding. https://p5js.org. Accessed: 2026-05-10.

[46] Meta Platforms, Inc. 2025. Immersive Web SDK: AI-native WebXR Development. https://iwsdk.dev. Version 0.3.1. Accessed: May 2026.

[47] Mistral AI. 2025. Introducing: Devstral 2 and Mistral Vibe CLI. https://mistral.ai/news/devstral-2-vibe-cli. Accessed: 2026-02-02.

[48] Kyzyl Monteiro, Ritik Vatsal, Neil Chulpongsatorn, Aman Parnami, and Ryo Suzuki. 2023. Teachable Reality: Prototyping Tangible Augmented Reality with Everyday Objects by Leveraging Interactive Machine Teaching. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 459, 15 pages. doi:10.1145/3544548.3581449

[49] Sachith Muthukumarana, Alaeddin Nassani, Noel Park, Jürgen Steimle, Mark Billinghurst, and Suranga Nanayakkara. 2022. XRtic: A prototyping toolkit for xr applications using cloth deformation. In *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 548–557. doi:10.1109/ISMAR55827.2022.00071

[50] Brad Myers, Scott E. Hudson, and Randy Pausch. 2000. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.* 7, 1 (March 2000), 3–28. doi:10.1145/344949.344959

[51] Samantha Narchetty, Arthur Caetano, and Misha Sra. 2025. Towards AI Drone Assistants in Extended Reality. In *2025 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. 885–886. doi:10.1109/ISMAR-Adjunct68609.2025.00239

[52] Nels Numan, Gabriel Brostow, Suhyun Park, Simon Julier, Anthony Steed, and Jessica Van Brummelen. 2025. CoCreatAR: Enhancing Authoring of Outdoor Augmented Reality Experiences Through Asymmetric Collaboration. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 1232, 22 pages. doi:10.1145/3706598.3714274

[53] NVIDIA. 2024. NVIDIA CloudXR SDK 6.0: GPU-Accelerated XR Streaming Platform. https://docs.nvidia.com/cloudxr-sdk/latest/. Accessed: 2026-05-10.

[54] Thibault Raffaillac and Stéphane Huot. 2022. What do Researchers Need when Implementing Novel Interaction Techniques? *Proc. ACM Hum.-Comput. Interact.* 6, EICS, Article 159 (June 2022), 30 pages. doi:10.1145/3532209

[55] Raf Ramakers, Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2016. RetroFab: A Design Tool for Retrofitting Physical Interfaces using Actuators, Sensors and 3D Printing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 409–419. doi:10.1145/2858036.2858485

[56] Sruti Srinidhi, Edward Lu, and Anthony Rowe. 2024. XaiR: An XR Platform that Integrates Large Language Models with the Physical World. In *2024 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 759–767. doi:10.1109/ISMAR62088.2024.00091

[57] Stack Overflow. 2025. *Technology: Most Popular Technologies — 2025 Developer Survey*. Stack Overflow. https://survey.stackoverflow.co/2025/technology#most-popular-technologies-language-learn-ai Accessed: 2026-01-25.

[58] Anthony Steed, Lisa Izzouzi, Klara Brandstätter, Sebastian Friston, Ben Congdon, Otto Olkkonen, Daniele Giunchi, Nels Numan, and David Swapp. 2022. Ubiq-exp: A toolkit to build and run remote and distributed mixed reality experiments. *Frontiers in Virtual Reality* Volume 3 - 2022 (2022). doi:10.3389/frvir.2022.912078

[59] SAM 3D Team, Xingyu Chen, Fu-Jen Chu, Pierre Gleize, Kevin J Liang, Alexander Sax, Hao Tang, Weiyao Wang, Michelle Guo, Thibaut Hardin, Xiang Li, Aohan Lin, Jiawei Liu, Ziqi Ma, Anushka Sagar, Bowen Song, Xiaodong Wang, Jianing Yang, Bowen Zhang, Piotr Dollár, Georgia Gkioxari, Matt Feiszli, and Jitendra Malik. 2025. SAM 3D: 3Dfy Anything in Images. (2025). arXiv:2511.16624 [cs.CV] https://arxiv.org/abs/2511.16624

[60] Viswanath Venkatesh, Michael G. Morris, Gordon B. Davis, and Fred D. Davis. 2003. User Acceptance of Information Technology: Toward a Unified View. *MIS Quarterly* 27, 3 (2003), 425–478. http://www.jstor.org/stable/30036540

[61] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 2054, 25 pages.

[62] Youjia Wang, Ruixiang Cao, Teng Xu, Yifei Liu, Dong Zhang, Yiwen Wu, and Jingyi Yu. 2025. DreamPrinting: Volumetric Printing Primitives for High-Fidelity 3D Printing. In *ACM SIGGRAPH 2025 Emerging Technologies (SIGGRAPH '25)*.

Association for Computing Machinery, New York, NY, USA, Article 4, 2 pages. doi:10.1145/3721257.3734025

[63] Xiaoyan Wei, Zijian Yue, and Hsiang-Ting Chen. 2025. SnapNCode: An Integrated Development Environment for Programming Physical Objects Interactions. In *Distributed, Ambient and Pervasive Interactions*, Norbert A. Streitz and Shin'ichi Konomi (Eds.). Springer Nature Switzerland, Cham, 161–177. doi:10.1007/978-3-031-92977-9_11

[64] Shutong Wu and Justin P. Barnett. 2025. MCP-Unity: Protocol-Driven Framework for Interactive 3D Authoring. In *Proceedings of the SIGGRAPH Asia 2025 Technical Communications (SA Technical Communications '25)*. Association for Computing Machinery, New York, NY, USA, Article 7, 4 pages. doi:10.1145/3757376.3771417

[65] Chengyuan Xu, Radha Kumaran, Noah Stier, Kangyou Yu, and Tobias Höllerer. 2024. Multimodal 3D Fusion and In-Situ Learning for Spatially Aware AI. In *2024 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 485–494. doi:10.1109/ISMAR62088.2024.00063

[66] Hui Ye, Jiaye Leng, Pengfei Xu, Karan Singh, and Hongbo Fu. 2024. ProInterAR: A Visual Programming Platform for Creating Immersive AR Interactions. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 610, 15 pages. doi:10.1145/3613904.3642527

[67] Lei Zhang, Jin Pan, Jacob Gettig, Steve Oney, and Anhong Guo. 2024. VRCopilot: Authoring 3D Layouts with Generative AI Models in VR. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 96, 13 pages. doi:10.1145/3654777.3676451

[68] Ada Yi Zhao, Aditya Gunturu, Ellen Yi-Luen Do, and Ryo Suzuki. 2025. Guided Reality: Generating Visually-Enriched AR Task Guidance with LLMs and Vision Models. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology (UIST '25)*. Association for Computing Machinery, New York, NY, USA, Article 146, 15 pages. doi:10.1145/3746059.3747784

[69] Chenfei Zhu, Shao-Kang Hsia, Xiyun Hu, Ziyi Liu, Jingyu Shi, and Karthik Ramani. 2025. agentAR: Creating Augmented Reality Applications with Tool-Augmented LLM-based Autonomous Agents. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology (UIST '25)*. Association for Computing Machinery, New York, NY, USA, Article 54, 23 pages. doi:10.1145/3746059.3747676