Vibe Coding XR: Accelerating AI + XR Prototyping with XR Blocks and Gemini

Ruofei Du*†, Benjamin Hersh, David Li*, Nels Numan‡, Xun Qian‡, Yanhe Chen‡, Zhongyi Zhou‡, Jiahao Ren, Xingyue Chen, Robert Timothy Bettridge, Faraz Faruqi, Xiang 'Anthony' Chen, Steve Toh, David Kim
https://xrblocks.github.io/gem
https://github.com/google/xrblocks
Google XR Labs

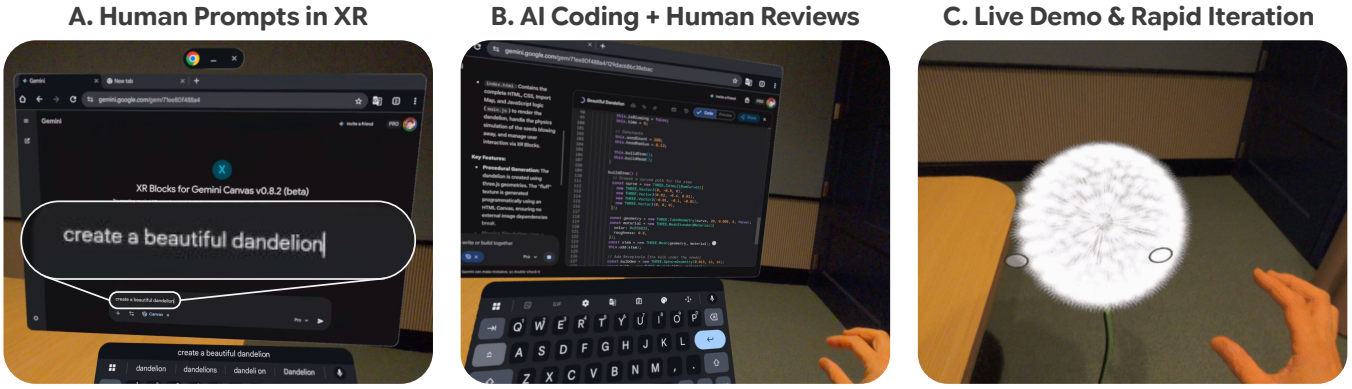

Figure 1: Example user journey of Vibe Coding XR, an end-to-end workflow for creating immersive AI + XR experiences via vibe coding: (A) User types "create a beautiful dandelion" with XR Blocks Gem (http://xrblocks.github.io/gem) on a Galaxy XR headset in Chrome. (B) Gemini translates the input into an interactive XR application within a minute, while the user reviews its reasoning and coding process. (C) User selects the "Enter XR" button and sees an animated dandelion dispersing upon pinch.

Abstract

While large language models (LLMs) have accelerated traditional software development through intent-driven "vibe coding", prototyping intelligent Extended Reality (XR) experiences remains a major challenge. The fundamental barrier is not just the steep learning curve for human creators, but that low-level sensor APIs and complex game engine hierarchies are ill-suited for LLM reasoning, routinely exceeding context windows and inducing syntax hallucinations. To bridge this gap, we contribute XR Blocks, an open-source, *LLM-native* WebXR framework. Unlike traditional engines, XR Blocks introduces a semantic "Reality Model" that aligns spatial computing primitives (users, physical environments, and agents) with natural language, providing a robust, concise vocabulary optimized for generative AI. Building upon this foundation, we present Vibe Coding XR, an end-to-end prototyping workflow that leverages LLMs to translate high-level prompts (*e.g.*, *"create a dandelion that reacts to my hand"*) directly into functional, physics-aware mixed-reality applications. To minimize the friction of on-device testing, the workflow introduces a seamless desktop "simulated reality" to headset deployment loop. Finally, we introduce vcxr60, a pilot dataset of 60 XR prompts paired with an automated evaluation pipeline. Our technical evaluation demonstrates high one-shot execution success, enabling practitioners to bypass low-level hurdles and rapidly move from "idea to reality." Code and live demos are available at https://github.com/google/xrblocks and http://xrblocks.github.io/gem.

CCS Concepts

• **Human-centered computing** → *User interface toolkits*; **Mixed / augmented reality**; Natural language interfaces.

Keywords

vibe coding, extended reality, augmented reality, XR Blocks, prototyping, AI, XR, generative XR, toolkit

1 Introduction

Recent advances in Large Language Models (LLMs) [9, 30, 54, 59] and agentic workflows [13, 26, 29] are fundamentally reshaping software engineering and creative computing. We are witnessing the growing prevalence of "vibe coding" [20], a paradigm in which

*Both authors contributed equally to XR Blocks.
†Corresponding author: me@duruofei.com.
‡Equal contribution, sorted alphabetically.

This work is licensed under a Creative Commons Attribution 4.0 International License.
VRST '26, Sendai, Japan
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2811-2/2026/11
https://doi.org/10.1145/3822517.3848668

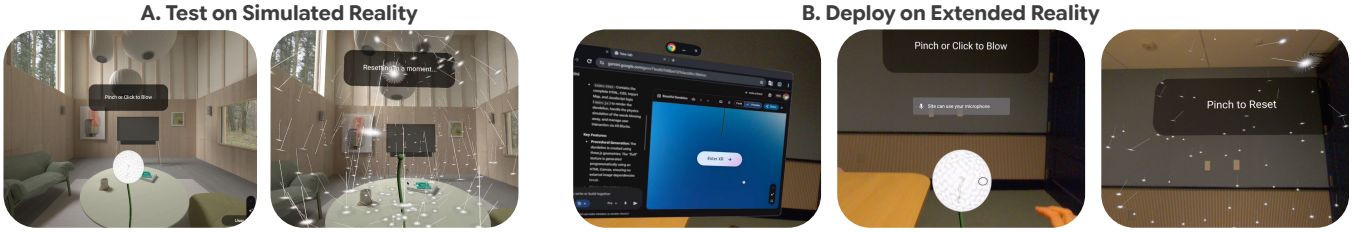

Figure 2: Vibe Coding XR accelerates AI + XR prototyping by allowing users to (A) test their "vibe coding" results on desktop in a "simulated reality" environment, and (B) deploy the same demo on an Android XR headset with body and hand interactions.

high-level human intent is translated directly into functional software. While tools such as Gemini Canvas [27], Antigravity [29], Cursor [13], and Claude Code [3] have successfully expanded this capability for 2D and, more recently, 3D web development [19, 60], the domain of Extended Reality (XR) remains technically inaccessible for rapid prototyping. Creating *intelligent spatial* experiences still requires navigating a fragmented ecosystem of perception pipelines [33, 51, 69], low-level sensor APIs [21, 43], massive game engine hierarchies [56, 58], and the physical friction of constant on-device testing.

Crucially, the absence of "vibe coding" in XR stems from a fundamental mismatch between how spatial software is currently built and how LLMs reason. Current models struggle to generate reliable XR applications because they are forced to operate over massive, low-level syntax trees and fragmented API layers rather than semantic, human-centered spatial concepts. This incidental complexity routinely exhausts context windows, fragments reasoning steps, and induces code hallucinations. To enable reliable intent-driven spatial computing, we need an abstraction layer explicitly designed not just for human accessibility, but for *machine reasoning*.

To bridge this gap, we introduce XR Blocks [42], an open-source WebXR [5] framework designed specifically to make spatial computing intelligible to generative AI. As the foundational contribution of this work, XR Blocks introduces an *LLM-native* "Reality Model" that treats the user, the physical environment, and intelligent agents as first-class, configurable primitives. By encapsulating the complexities of sensor fusion, rendering, and spatial physics into a concise, semantic vocabulary, XR Blocks provides LLMs with the precise architectural constraints necessary to generate complex spatial behaviors without hallucinating underlying implementation.

Building upon this enabling framework, we present Vibe Coding XR, an end-to-end rapid prototyping workflow that leverages the long-context reasoning of LLMs to act as an expert XR designer. Using a specialized system prompt and a web-based interface, this workflow translates natural language directly into deployable, physics-aware XR applications in under 90 seconds. To solve the persistent HCI challenge of spatial testing friction, Vibe Coding XR allows creators to rapidly iterate on generated scripts within a desktop "simulated reality" environment before deploying instantly to an Android XR [25] headset for live hand interaction and environmental sensing.

Finally, evaluating the viability of generated XR software requires moving beyond subjective manual testing. We contribute the VCXR60 dataset, a pilot dataset consisting of 60 diverse spatial computing prompts sourced from four workshops. Coupled with an automated, headless-browser evaluation pipeline, we provide a quantitative baseline for XR code generation.

Our evaluation, alongside diverse application scenarios demonstrating XR realism and multi-modal interaction, illustrates the robustness of the system.

In summary, we contribute:

- **The XR Blocks Framework:** An open-source, LLM-native WebXR architecture that abstracts low-level spatial perception and interaction pipelines into a semantic "Reality Model" optimized for reliable AI code generation.
- **Vibe Coding XR Workflow:** A web-based, rapid prototyping loop that leverages LLMs to translate natural language intent into deployable XR applications, utilizing a novel desktop-simulator-to-headset testing pipeline to reduce iterative friction.
- **The VCXR60 Dataset & Evaluation:** A preliminary dataset and automated testing pipeline for evaluating the runtime execution viability of intent-driven XR code generation, demonstrating the high one-shot pass rates of our framework.

As an initial exploration of vibe coding in XR, this work focuses on technical feasibility and automated runtime viability rather than formal user studies or cross-model evaluations. We hope our open-source framework and benchmark lay the groundwork for next-generation agentic workflows and reproducible AI + XR authoring.

2 Related Work

Vibe Coding XR bridges the gap between generative AI workflows and spatial computing by enabling intent-driven XR prototyping.

2.1 GenXR: Creating AI + XR Experiences

XR development tooling has evolved from early toolkits like VR Juggler [7] and ARToolKit [39] to comprehensive game engines such as Unity [56] and Unreal [58]. While these native engines offer high ceilings for fidelity, they impose steep learning curves, significant friction for rapid prototyping for XR, and high barriers for sharing executable code. Conversely, WebGL libraries like `three.js` [55] and A-Frame [2] offer cross-platform accessibility, but lack the high-level abstractions necessary for complex XR interaction and seamless AI integration.

Recent web-based solutions and SDKs have attempted to bridge this gap, including the Wonderland Engine [62] and Meta's Immersive Web SDK (IWSDK) [45]. While Wonderland Engine is a powerful WebXR-compatible engine optimized for game making,

System	in-situ XR prompting	web-native architecture	LLM-optimized abstraction	interactive perception	intent-driven vibe coding	desktop simulator	zero-build sharing	open source
Unity [56] MCP	-	-	-	-	-	✓	-	-
LLMR [14]	✓	-	-	limited	✓	-	-	✓
DreamCodeVR [23]	✓	-	-	-	✓	-	-	✓
agentAR [69]	✓	-	-	✓	✓	-	-	✓
UniGen [65]	-	-	-	-	✓	✓	-	✓
UnrealLLM [53]	-	-	-	-	✓	-	-	✓
AutoUE* [66]	-	-	-	-	✓	✓	-	✓
A-Frame [2]	-	✓	-	-	-	-	✓	✓
Wonderland Engine [62]	-	✓	-	-	-	-	-	-
Meta IWSDK* [45]	-	✓	ECS interop	-	✓	✓	-	✓
VCXR (XR Blocks)	✓	✓	✓	✓	✓	✓	✓	✓

*Denotes concurrent work to XR Blocks [9/2025] / Vibe Coding XR [2/2026] (e.g., Meta IWSDK [10/2025], AutoUE [4/2026]).

Table 1: Comparison of key capabilities across XR authoring and generative AI systems. Evaluation criteria: (1) in-situ XR prompting: prompting directly within the XR headset session; (2) web-native architecture: execution entirely via WebXR without native binaries; (3) LLM-optimized abstraction: exposing a concise semantic syntax to minimize token boilerplate for XR interactions; (4) interactive perception: supporting real-time on-device models like gesture recognition and object recognition; (5) intent-driven vibe coding: end-to-end translation from natural language into interactive XR; (6) desktop simulator: built-in browser environment emulating head, hand, and sensor inputs; (7) zero-build sharing: distributable via instant URL without packaging; (8) open source: permissively licensed repository.

it requires creators to download a localized desktop host and does not easily integrate with web-based AI workflows. IWSDK features a built-in desktop simulator and occlusion capabilities [18], but crucially lacks the Interactive Perception (AI integration) required to treat the environment as a first-class primitive for agentic workflows; its lack of camera support also makes it challenging for AI and XR integration.

XR Blocks bridges this divide. By offering a high-level, web-optimized interaction model akin to MRTK [4], VRTK [61], and XRI [57] for Unity, our framework abstracts low-level sensor and perception pipelines so creators can focus on the *what* of an experience rather than the *how*. As a web-native architecture, XR Blocks makes AI + XR prototyping inherently accessible and reproducible, enabling the community to share, fork, and build upon each other's creations. Vibe Coding XR further leverages this framework to create behavioral AI + XR experiences, with complex 3D asset creation [35, 36, 49, 52, 64, 67] remaining out of scope. While the concurrent work IWSDK also provides a high-level WebXR SDK, it usually requires a compilation step and an interop between `three.js` [55] and its own ECS subsystem, while lacking deeper perception runtimes such as gesture recognition and the native vision-language model integration offered by XR Blocks.

2.2 Vibe Coding: Gaps from AI to XR

The AI community benefits from a "flywheel effect" driven by open ecosystems such as Hugging Face [38] and TensorFlow Hub [28], crowd-sourced benchmarks like LMArena [11], and composable frameworks including JAX [8], PyTorch [47], and TensorFlow [1]. XR, however, lacks a comparable substrate for rapid, intent-driven iteration on a reproducible platform.

Recent efforts have begun to address this by leveraging LLMs to generate spatial objects and scenes. Systems like LLMR [14], DreamCodeVR [23], and Thing2Reality [35] target Unity, with commercial tools such as Bezi [6] pursuing a similar direction. On the infrastructure side, Ubiq-Genie [46] proposed a server-client architecture that exposes AI services to Unity-based XR applications. Recent advancements also include AutoUE [66], a multi-agent system that generates 3D games in Unreal Engine and utilizes an automated play-testing pipeline via a Model Context Protocol (MCP) to provide desktop simulation evaluation for the LLM. Furthermore, systems like DreamGarden [19] demonstrate LLM-driven hierarchical planning to generate functional 3D game environments and C++ logic in Unreal Engine.

While these approaches successfully push the boundaries of automated game design and 3D scene composition, they predominantly rely on asynchronous compilation workflows within heavy, closed-engine ecosystems like Unity and Unreal. As a result, they are frequently limited to one-time generation tasks and lack deep AI capabilities, such as generating an agent that can dynamically recognize objects in the physical room. This friction limits the collective "vibe coding" paradigm [20], in which an agent's utility is fundamentally tied to its ability to synthesize and rapidly share knowledge across the entire relevant domain.

To overcome this, we develop XR Blocks as a spatial computing platform on the open web inspired by Visual Blocks [16]. By coupling LLMs [68] directly with our "Reality Model", Vibe Coding XR enables creators to generate immersive, intelligent, and physics-aware interactions. Unlike prior art, our framework supports in-situ XR prompting directly within the environment and features a seamless desktop simulator for frictionless testing with hand inputs, camera simulation, and cross-platform machine learning runtimes. This web-based approach ensures that every contribution is (1) built upon a transparent, web-standard codebase, (2) designed for rapid iteration via zero-build (contrasting Wonderland Engine and IWSDK) sharing, and (3) instantly distributable across diverse devices and users. By prioritizing these pillars, we move toward an ecosystem where the community does not merely consume agents but actively drives their evolution, supporting a "flywheel effect" for XR. Refer to Table 1 for a recapitulation of key differentiators.

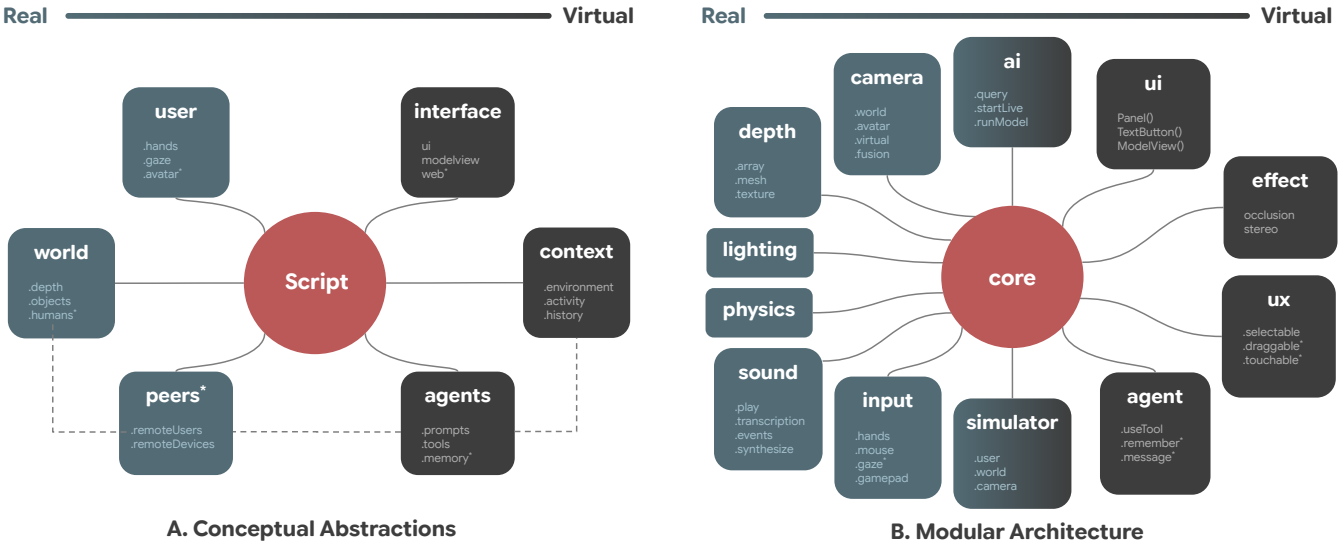

Figure 3: Design of the XR Blocks Framework: (A) The "Reality Model" conceptual abstraction, which aligns spatial computing primitives 1:1 with natural language concepts to prevent LLM hallucination over fragmented syntax trees. (B) The modular architecture of the "core" engine encapsulating low-level perception and interaction logic. * denotes "not yet implemented".

3 System Architecture

We introduce Vibe Coding XR, a rapid prototyping workflow that combines the generative reasoning capabilities of LLMs with a high-level modular XR framework. This combination allows creators to bypass the friction of low-level systems programming and focus on designing spatial behaviors, AI logic, and interactions.

3.1 The XR Blocks Reality Model

At the foundation of our system is XR Blocks [42], an open-source, LLM-native abstraction layer explicitly designed to make spatial computing intelligible to both human creators and generative AI. Traditional game engines and low-level WebGL APIs represent scenes as massive, deeply nested syntax trees and fragmented data streams from gamepad and hands. When forced to navigate these structures, LLMs routinely exhaust their context windows, lose track of spatial relationships, and hallucinate API calls.

To solve this, XR Blocks introduces a semantic *Reality Model*. This unified abstraction absorbs the incidental complexity of perception, rendering, and interaction behind high-level, human-centered primitives (*e.g.*, `user`, `world`, `ui`, `ai`), providing a robust, concise vocabulary optimized for machine reasoning. By encapsulating WebXR session initialization, render loops, and input polling into declarative lifecycle hooks within `xb.Script`, the LLM spends its reasoning budget mostly on *application logic* rather than *boilerplate plumbing*. The following snippet demonstrates XR Blocks's design principle of keeping syntax semantically dense for the LLM. It creates an object at the user's eye level with a preferred distance, which changes color when the user performs a hand pinch. Crucially, this code is *device-agnostic*—the `onSelectEnd` event fires seamlessly whether triggered by a physical hand gesture in an Android XR headset, a touch event on an Android phone, or a mouse click in the desktop simulator:

```
class MainScript extends xb.Script {
  init() {
    // geometry and lighting setup is omitted.
    this.player = new THREE.Mesh(geometry, material);
    this.player.position.set(0, xb.user.height - 0.2,
    ↪  -xb.user.objectDistance);
    this.add(this.player);
  }
  onSelectEnd(event) {
    this.player.material.color.set(Math.random() * 0xffffff);
  }
}
```

As shown in Figure 3, the framework's *core* engine manages a rich, plug-and-play architecture of subsystems required for spatial computing, significantly expanding the bounds of what LLMs can instantly generate:

- **Hands & Gestures:** Cross-platform 25-joint 6-DoF hand tracking, with real-time detection for gestures (*e.g.*, pinch, grab, palm-up, point), alongside hooks for custom on-device models.
- **World Understanding & Physics:** Real-time environmental depth meshing, dynamic occlusion, and surface plane estimation. Powered by the Rapier [12] physics engine, virtual rigid bodies dynamically collide with and bounce off physical room geometry.
- **Spatial UI & Audio:** Declarative 3D flexbox panels with Signed Distance Field (SDF) rendering for crisp text, combined with speech synthesis, and voice recognition APIs.
- **AI Integration:** A hybrid edge-cloud architecture supporting both on-device models in LiteRT.js and cloud multimodal LLMs (Gemini) for live spatial reasoning and vision-language queries.
- **Multiplayer (Peers):** Built-in WebRTC networking for cross-platform presence, shared spatial state objects, and spatialized voice to facilitate multi-user collaborative experiences.

By abstracting these technical hurdles, XR Blocks lowers the barrier to entry while retaining a high ceiling for expressivity, positioning it as the ideal foundation for Vibe Coding XR. The

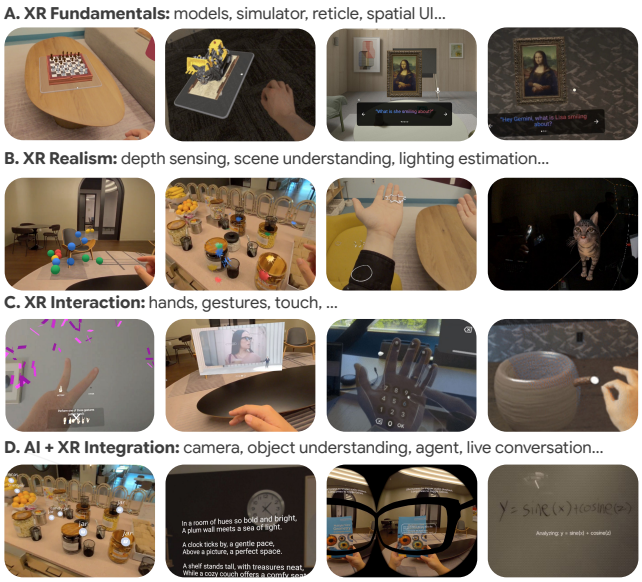

Figure 4: Human-coded templates and samples in XR Blocks provide the foundational best practices and API grounding for Vibe Coding XR.

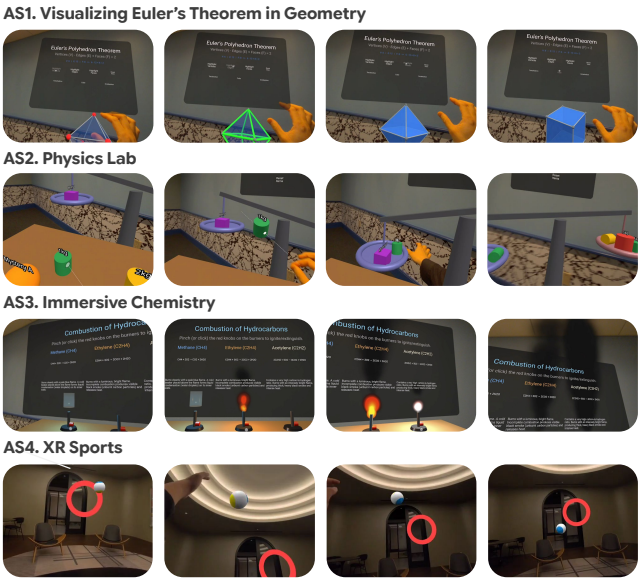

Figure 5: AI-generated application for educational and fitness use cases. See Appendix B for full prompts.

framework's architecture was iteratively refined through the development of over 30 open-source templates (examples: Figure 4), incorporating feedback from weekly workshops and live demos.

3.2 Agentic Alignment & System Prompt

Vibe Coding XR leverages the long-context capabilities and advanced reasoning processes of modern LLMs like Gemini [54] to function as an expert XR designer and engineer. However, merely providing an API is insufficient; the LLM must be explicitly constrained and guided to ensure reliable code generation.

To achieve this, we first developed a specialized system prompt architecture (detailed in Appendix C) that explicitly grounds the LLM in the XR Blocks Reality Model through an open-source web interface. As the framework expanded, we formalized these prompt-driven constraints natively into the repository as **Agent Alignment Specifications** (*e.g.*, `AGENTS.md`). These machine-readable files now serve as the ultimate source of truth for downstream AI coding agents, automatically teaching them the repository structure, key files, testing protocols, and spatial conventions. The foundational system prompt architecture consists of:

- **Persona & Guidelines:** Configures the LLM with a domain expert persona adhering to the rules of engagement in the system prompt, focusing on best practices for room-scale XR environments (*e.g.*, spatial layout, human-scale proportions, safety bounding, and interaction distances).
- **Architectural Constraints:** Dictates how dependencies and subsystems (*e.g.*, `xb-core`, `xb-ui`, `xb-ai`) are handled to enforce the repository's strict declarative styles, directly informed by the system prompt to ensure generated code executes flawlessly without context loss.
- **Source Code & Templates:** Grounds the model with the full source code of the core `xrblocks.js` library, alongside curated templates and samples (illustrated in Figure 4). This strict grounding acts as a technical constraint—enforcing the "only call APIs that exist" rule to minimize hallucination and encourage adherence to established spatial design patterns.

3.3 The Simulated-to-Extended Reality Loop

Users interact with Vibe Coding XR through high-level natural language prompts (or "vibes"). The LLM translates this intent directly into functional XR Blocks scripts [68]. Crucially, to solve the persistent HCI friction of iterative spatial testing—where creators must constantly compile, build, and don/doff a headset—Vibe Coding XR introduces a seamless *simulated-to-extended reality loop.*

Scripts can be previewed immediately in a built-in desktop simulated reality environment running directly in the browser. This injects a proxy of the physical environment (complete with mock depth maps and scene meshes). This allows for rapid visual validation of physics, logic, spatial UI layout, and AI reasoning without leaving the desktop. Once verified, the exact same URL is deployed instantly to an Android XR headset (*e.g.*, Galaxy XR) for real-world environmental sensing and live hand interaction. This cross-platform, write-once-run-anywhere pipeline eliminates build times and enables a frictionless "vibe coding" loop.

3.4 Application Scenarios: From Prompt to XR

To demonstrate the versatility and expressive range of Vibe Coding XR, we present representative application prototypes generated during four hands-on workshops with 20 participants. These prototypes (with full prompts listed in Appendix B) exercise the core capabilities of the XR Blocks Reality Model across interactive learning, room-scale physics, and procedural gaming (Figure 5):

Educational AI + XR Experiences. As shown in Figure 5, Vibe Coding XR supports rapid prototyping of interactive spatial learning environments. Prompted with *"visualize Euler's theorem in geometry and explain vertices, edges, and facets"* (AS1), the system generated an immersive *Math Tutor*. The LLM instantiated polyhedral geometries (tetrahedron, cube, octahedron), wired pinch interactions via `onSelect()` to toggle face and edge highlights, and rendered accompanying spatial explanation cards using `xb.ui`. Similarly, prompting for an *Immersive Chemistry* experience (AS3) produced educational formula cards and 3D volumetric particle effects, enabling users to trigger and observe simulated hydrocarbon combustion (e.g., methane) safely via hand-pinch gestures.

Physics-Aware XR Realism & Interaction. By grounding the model in XR Blocks's physics and depth subsystems, users can generate tactile, environment-aware experiences (Figure 5). Prompted for an *XR Sports* application (*"Let me play volleyball with hands and collide with my environment"*, AS4), the system generated a textured volleyball that reacts dynamically to both 6-DoF tracked hand joints and real-world room geometry extracted via depth sensing (`xb.depth.mesh`). Furthermore, a *Physics Lab* prompt (AS2) generated a mixed-reality balance scale where users grasp, lift, and drop labeled weights to explore mechanical equilibrium in real time.

Game Prototyping & Procedural Generation. The workflow drastically compresses the time required to stand up interactive XR game mechanics. Prompted to *"create the Chrome Dino game in XR"* (AS5), Vibe Coding XR generated a voxelized runner complete with rushing obstacles on a floating translucent track, spatial audio cues, and collision triggers in under 90 seconds. Prototypers also leveraged the workflow for procedural spatial compositions, including rendering interactive astronomical star charts and explorable architectural structures directly in the headset.

4 Preliminary Evaluation with VCXR60

Evaluating XR applications poses unique challenges, as it typically requires manual, on-device testing and subjective human evaluation. However, assessing the viability of generative XR systems requires fundamentally new, scalable technical metrics that can keep pace with the rapid iteration of large language models. To meet this need, we introduce the **VCXR60** dataset, an open-source benchmark designed to automate the testing of intent-driven spatial computing.

Sourced from four one-hour workshops, VCXR60 consists of 60 XR application prompts provided by 20 participants (detailed in Appendix A). Using this dataset with XR Blocks's desktop simulator, we measured inference time and one-shot pass rates (pass@1). Inspired by code LLM benchmarks like HumanEval [10], we define a "pass" strictly as a zero-error execution in an automated runtime test. In 3D graphics and WebXR, the rendering pipeline serves as an initial syntactic and execution validator, as runtime errors during initialization, asset loading, or per-frame rendering immediately flag code failures. We monitor uncaught exceptions and console error logs across scene initialization and active animation frames in a headless Chromium browser via Playwright [48]. Crucially, we emphasize that the pass@1 metric is solely an automated runtime check (i.e., verifying that the generated code compiles, initializes, and runs without uncaught errors or crashing); it is not a measure of semantic correctness, spatial fidelity, or user intent fulfillment.

Early exploratory testing in September 2025 using XR Blocks v0.1.0 with Gemini 2.5 Pro yielded an approximate 70% pass rate, where failures predominantly stemmed from edge-case framework bugs (e.g., coordinate frame mismatches in the simulator and missing gesture callbacks) and LLM API hallucinations. These insights informed a rapid six-month iteration cycle. After 10 major releases, we evaluated XR Blocks v0.11.0 on VCXR60 using the latest available models at the time of benchmarking: `gemini-3.1-pro-preview` and `gemini-3-flash-preview` (accessed March 19–22, 2026). We selected these preview backends because their extended reasoning budgets (thinking mode) were essential for complex multi-component spatial scene synthesis. While preview endpoints undergo deprecation cycles as stable models emerge, we ensure full empirical reproducibility by releasing the versioned SDK and the complete prompt suite on our project repository. We also note a methodological caveat: the improvement from ∼70% in v0.1.0 to >87–95% in v0.11.0 reflects the combined benefit of framework-level hardening (e.g., standardized schemas, curated system instructions, and simulator bug fixes) and general advancements in foundation model capabilities over that six-month period.

Averaged across five runs, our evaluation revealed a clear performance advantage for larger models, while differences between "thinking" levels within the same model family were minor and statistically non-significant (Table 2). Specifically, Gemini 3.1 Pro achieved higher overall runtime pass rates (High: 95.5% ± 2.2%, 95% CI [92.7%, 98.3%]; Low: 94.0% ± 2.6%, 95% CI [90.7%, 97.3%]) compared to Gemini 3 Flash (High: 87.8% ± 3.4%, 95% CI [83.6%, 91.9%]; Low: 87.3% ± 6.4%, 95% CI [79.4%, 95.3%]). Because the confidence intervals for High and Low thinking heavily overlap, thinking mode does not significantly alter zero-error runtime pass rates; rather, its primary trade-off is inference latency versus planning capacity for complex prompts. Notably, across all backends, every single test prompt achieved at least one successful execution across the five runs (100% at-least-one pass rate), indicating that multi-turn retry loops can effectively overcome non-deterministic generation errors. For simpler interactions (e.g., *"Create a beautiful dandelion that blows away when I pick it up"*), faster models like Gemini Flash often completed generation in under 20 seconds. For applications requiring complex animations and precise hand-interaction logic (e.g., Prompt 060 in Appendix A), however, Flash exhibited higher error rates compared to Gemini Pro.

Manual Review and Qualitative Observations. To evaluate interaction quality beyond automated headless checks, the authors manually reviewed and interacted with all generated demos across the 60 benchmark prompts in both the desktop simulator and on-device with an Android XR headset, selecting representative prototypes to showcase in the supplementary video. While all prompts achieved technical runtime viability in at least one run, not every creation was qualitatively successful due to non-determinism in LLM inference. For instance, in a two-player "ping-pong game" (Prompt 055 in Appendix A), the generated racket was occasionally not grabbable by the user's hands; even when grasping succeeded, the racket sometimes failed to orient correctly with the hand joints, or the ping-pong ball failed to register physical collisions. These

Table 2: Inference time and one-shot pass rate (pass@1) for XR Blocks Gem on VCXR60 across 5 runs (accessed March 19–22, 2026). Pass@1 reports the mean, standard deviation (SD), and 95% confidence interval (CI) across runs. Note that pass@1 is solely a runtime check (i.e., code executes without crashing or unhandled console errors), not a measure of semantic correctness or alignment with user intent.

Vibe Coding XR		Time (secs)		Pass Rate@1 (5 runs)	
Model Backend	Thinking	Median	IQR	Mean ± SD	95% CI
gemini-3.1-pro	High	86.02	32.12	**95.5%** ± 2.2%	[92.7%, 98.3%]
gemini-3.1-pro	Low	33.39	8.60	94.0% ± 2.6%	[90.7%, 97.3%]
gemini-3-flash	High	22.26	4.66	87.8% ± 3.4%	[83.6%, 91.9%]
gemini-3-flash	Low	**17.30**	4.00	87.3% ± 6.4%	[79.4%, 95.3%]

subtle spatial and behavioral discrepancies demonstrate that technical zero-error execution does not guarantee full interaction fidelity or user intent fulfillment, motivating future work on more rigorous spatial benchmarking platforms that systematically evaluate and align with human ratings (section 6).

Qualitative Feedback from Workshop Prototypers. To complement our technical benchmarking with user perspective, we documented qualitative observations across four prototyping workshops involving 20 participants (spanning research scientists, UX prototypers, and software engineers). Three consistent themes emerged: (1) **Rapid Scaffolding vs. Native Build Latency**: Participants highlighted that generating functional 3D scenes in under 90 seconds dramatically bypassed the traditional 5–10 minute compile-and-deploy cycles of native engines like Unity, making rapid concept exploration viable. (2) **Friction in Fine-Grained Spatial Tuning**: While coarse scaffolding was rapid, participants reported friction when making localized adjustments (e.g., nudging coordinates or styling UI cards), noting that conversational prompting often unintentionally perturbed existing scene elements and motivating hybrid direct-manipulation controls. (3) **Debugging Interaction and Session Drift**: Participants found debugging dynamic spatial behaviors (e.g., hand triggers and colliders) challenging because the LLM lacks runtime visual perception, emphasizing the need for multimodal simulator snapshots and MVC architectures to decouple application state from transient LLM context.

5 Deployment Case Studies and Reflections

Following its open-source release, XR Blocks was adopted by early practitioners and workshop developers, yielding emergent prototyping patterns that bridge declarative 2D tooling with spatial physics. We summarize three qualitative deployment scenarios below:

Vibe Coding XR for Designers. Bridging the gap between declarative 2D design and spatial interaction has historically been a significant bottleneck. User experience designers typically author interfaces in 2D tools (*e.g.*, Figma), leaving XR prototypers to manually recreate these layouts within Unity to achieve spatial interactivity—a process that often takes hours. Leveraging the latest coding agents alongside XR Blocks's UI modules, creators can now upload a flat mockup and simply prompt the system to "create a spatial UI with the UI blocks skills." As demonstrated in Figure 6 (W1), Vibe Coding XR ingests a 2D screenshot of a music player and translates it into a translucent, floating spatial interface, complete with physics-aware, pinch-interactive buttons.

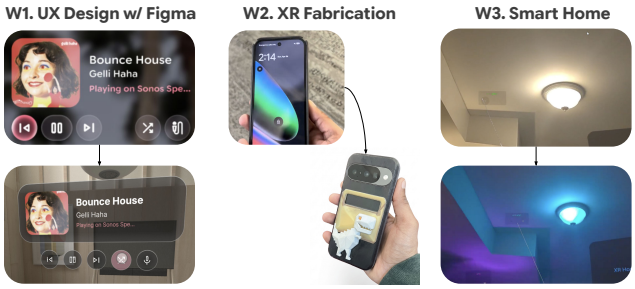

Figure 6: Emergent in-the-wild applications of Vibe Coding XR. (W1) Translating a flat 2D UI mockup (top) into an interactive, translucent spatial UI. (W2) Validating the scale and fit of a generative 3D-printed phone case against a physical device measured in XR with depth sensing. (W3) Opportunistic smart home control linking spatial interactions with physical IoT devices.

Vibe Coding XR for Fabrication. Vibe Coding XR also enables creators to transition virtual artifacts into the physical world. In one instance, an author prompted the system for a "Chrome Dino phone case with a leather-style wallet for a Pixel 10." Gemini generated a reference image, which was subsequently lifted into a 3D mesh utilizing the TRELLIS [63] generation model. Crucially, rather than judging proportions on a flat desktop monitor, the creator utilized the Android XR headset's passthrough to superimpose the generated 3D model directly onto a physical Pixel phone (Figure 6, W2). This *in-situ* registration allowed the creator to rigorously validate the shape, true physical scale, and wallet fitment prior to exporting the asset for 3D printing. By sharing a common spatial representation, Vibe Coding XR minimizes the costly trial-and-error typically associated with digital fabrication.

Vibe Coding XR for Home Controls. Extending spatial computing into the Internet of Things (IoT), an expert user deployed XR Blocks to interface with their Matter-enabled smart home ecosystem (Figure 6, W3). Using Vibe Coding XR, the user leveraged video see-through capabilities to authenticate the XR application's access to their local IoT network via a mobile companion device. By integrating real-time object recognition, the application identifies physical lighting fixtures within the room, assigns them to the existing smart home setup, and allows the user to manipulate colors and intensities through direct, *in-situ* XR interactions. This compellingly demonstrates Vibe Coding XR's capacity to facilitate opportunistic, context-aware control of the physical environment.

Engineering Reflections and Lessons Learned. Our design and deployment strategy for XR Blocks was inspired by open-web systems like Python Tutor [31]. However, navigating the intersection of generative AI and WebXR revealed non-obvious engineering pitfalls and design pivots:

The Cost of Monolithic Abstractions: Early framework iterations attempted to wrap Three.js in a monolithic engine class that

managed all lifecycles internally. In practice, this bloated context windows and caused severe API hallucinations. We pivoted to a lightweight `xb.Script` subclassing architecture where developers and models import only the specific primitives needed, drastically lowering token overhead.

The Simulation-to-Device Fidelity Gap: We initially assumed 2D mouse clicks would suffice as a testing proxy for hand gestures on desktop. However, upon deploying to physical headsets, subtle spatial bugs—such as inverted hand-joint orientation matrices, missing collision volumes, and incorrect depth occlusion—were immediately exposed. This necessitated re-architecting the desktop simulator to inject full 6-DoF simulated hand skeletons, synthetic depth maps, and camera streams into the WebXR polyfill.

Learning from Recurring Hallucinations: In generative code synthesis, LLMs frequently invent non-existent method signatures based on statistical priors from mature libraries (e.g., hallucinating `xb.hands.getPinchDistance()` or `xb.ui.addSlider()`). Rather than treating these solely as errors to be suppressed, we observed that when multiple models consistently hallucinated the exact same API shape across diverse prompts, it served as an unsupervised signal of an ergonomic API gap. We systematically cataloged these recurrent, syntactically coherent hallucinations and implemented them as first-class methods in subsequent framework releases.

Balancing Web Standards with Modern Tooling: While Guo [31] advocates for older, conservative technologies to maximize longevity, generative AI advances at a pace that demands strict type safety. We adopted TypeScript internally to generate formal declaration files, which proved essential for grounding coding agents, while preserving a zero-build vanilla JavaScript bundle for end-user HTML deployments.

6 Limitations and Discussion

This paper presents our initial step towards bridging the "idea to reality" gap in spatial computing. While our current implementation successfully prioritizes core abstractions for generative XR, its limitations highlight critical avenues for future research.

Absence of User Studies. While our weekly workshops provided rich qualitative feedback from 20 practitioners, this paper does not present a formal, controlled behavioral user study. Evaluating intent-driven spatial computing requires rigorous investigation into usability (e.g., System Usability Scale), learnability curves for non-programmers, cognitive workload (NASA-TLX), and task completion times relative to traditional game engines. Although technical systems research often focuses initially on architectural feasibility and reproducible benchmark metrics [22, 37], human-in-the-loop usability remains an essential requirement for validating end-user authoring workflows.

Narrow Scope of Automated Technical Evaluation. Our automated benchmark on VCXR60 captures baseline code viability via headless execution logs, verifying that generated applications compile, initialize, and run without unhandled runtime exceptions. However, this metric does not evaluate visual aesthetics, render-loop performance, physical plausibility, or semantic adherence to user intent. As demonstrated in our manual review, applets can achieve zero-error compilation while failing critical interaction heuristics (e.g., ungrabbable ping-pong rackets or misaligned joint transforms). Future benchmarks must incorporate multimodal autoraters and automated spatial unit tests.

Dependence on a Single Model Family. All technical evaluations in this work were conducted using the Gemini model family. While these models demonstrate high one-shot viability when grounded in our framework, instruction following, spatial math competence, and API adherence may vary significantly across other frontier foundation models. Evaluating XR Blocks across diverse LLMs is a critical next step to verify framework generality.

Performance and Platform Trade-offs. Our reliance on web technologies intentionally prioritizes accessibility and cross-platform shareability, though this introduces known trade-offs. The WebXR framework currently cannot match the raw rendering throughput of native engines like Unity or Unreal, and the dependency on cloud-hosted LLMs introduces network latency during the generation phase. To bridge this gap, a compelling longer-term objective is the development of an *LLM-driven cross-compiler* capable of translating high-level XR Blocks scripts into optimized, native code for target engines such as Godot [24]. We actively invite the community to contribute to the SDK to establish shared conventions for AI + XR interactions, which will also serve to optimize prompt efficiency and reduce LLM token costs.

Toward Rigorous Benchmarking. The VCXR60 dataset covers a valuable but limited slice of the vast AI + XR design space. Furthermore, our automated pass/fail metric captures baseline code viability but cannot assess usability, aesthetic quality, or interaction fidelity. As observed during our manual inspection of generated applets, subtle spatial failure modes—such as improper hand attachment, raycasting misalignments, or physics collision omissions—frequently escape automated runtime validation. A more rigorous future evaluation should decouple the contributions of the XR Blocks abstraction layer from the underlying `three.js` library, quantifying the exact token savings and error reduction our framework provides relative to raw API usage. Additionally, future benchmarks must address iterative, multi-turn usage scenarios—such as refining an existing prototype through conversational prompting—which introduces distinct challenges in context management and incremental code coherence. Just as ImageNet [15] galvanized progress in computer vision, the XR and HCI fields urgently need large-scale, open-source datasets of *prompt-to-interaction* pairs alongside automated spatial autoraters calibrated to human quality judgments to enable the systematic evaluation of generated spatial behaviors.

Expanding the Interaction Vocabulary. Currently, Vibe Coding XR accepts text as its primary input modality. However, spatial intent is often difficult to articulate through text alone. Hybrid approaches that blend natural language with spatial UI selections, gaze targeting [41, 50], micro-gestures [32], and cross-device interactions [70] could substantially improve the precision of user intent. Furthermore, XR Blocks is designed to be extensible beyond audio-visual outputs; integrating modules for haptics, olfactory interfaces, or other sensory modalities (e.g., via Arduino hardware through WebUSB) would empower creators to compose significantly richer, multi-sensory experiences.

Accessibility and Responsible Deployment. As we evolve XR Blocks through open-source collaboration, we recognize that the framework currently lacks ready-to-use accessibility modules for motor or visual impairments—a critical gap given our goal of lowering technical development barriers for XR. Moreover, the responsible deployment of LLM-driven spatial authoring tools demands rigorous human-in-the-loop evaluation to understand how designers and novices co-create with AI agents. Ensuring that generative outputs safely augment human intent, rather than introducing harmful or disorienting spatial behaviors, remains a paramount challenge for the HCI, XR, and AI communities.

7 Conclusion

We presented Vibe Coding XR, an end-to-end workflow that translates high-level creative intent into functional XR prototypes. By coupling the generative reasoning of LLMs with XR Blocks, an LLM-native abstraction layer, we demonstrated an approach that substantially reduces the distance between a natural language concept and an interactive, physics-aware spatial application. Our preliminary evaluation on the VCXR60 dataset demonstrates that providing LLMs with a semantic "Reality Model" and concise, declarative component lifecycles yields high zero-error runtime execution pass rates (up to 95.5%) across diverse spatial interaction scenarios, bypassing *the compilation overhead* and *steep learning curves* of traditional native game engines. Furthermore, by integrating a simulated-to-extended reality loop, we mitigate the substantial friction of iterative on-device testing.

We release this framework, workflow, and benchmark dataset as an invitation. We welcome the HCI, XR, and AI communities to build upon this foundation, expand the capabilities of generative spatial computing, and collectively work toward a future where moving from "idea to reality" is as fluid as describing it.

Acknowledgments

We thank all of XR Blocks contributors in 2025: David Li and Ruofei Du (equal primary contributions), Nels Numan, Xun Qian, Yanhe Chen, and Zhongyi Zhou, (equal secondary contributions, sorted alphabetically), as well as Evgenii Alekseev, Geonsun Lee, Alex Cooper, Brandon Jones, Min Xia, Scott Chung, Jeremy Nelson, Xiuxiu Yuan, Jolica Dias, Tim Bettridge, Benjamin Hersh, Michelle Huynh, Konrad Piascik, Ricardo Cabello, and David Kim. We extended our thanks to all of our 2026 student researchers and GitHub contributors: William Huang, Xincheng Huang, Hyunsung Cho, Salman Chishti, Akilesh, Jamie; We further thank the Gemini Canvas and AI Studio teams for their support including, but not limited to: Tim Bettridge, Yan Li, Daniel Marques, Deven Tokuno, Levent Yilmaz, Saravana Rathinam, Samuel Petit, Mike Taylor-Cai, Ammaar Reshi, and Robert Berry, We would like to thank Mahdi Tayarani, Max Dzitsiuk, Jim Ratcliffe, Patrick Hackett, Seeyam Qiu, Coco Fatus, Alon Hetzroni, Aaron Kim, Yinghua Yang, Brian Collins, Eric Gonzalez, Keith Moon, Nicolás Peña Moreno, Yidang Zhang, Yuhao He, Yi-Fei Li, Ziyi Liu, Jing Jin for their feedback and discussion on our early-stage proposal and WebXR experiments, as well as all of DepthLab [18], Ad hoc UI [17], Rapsai (Visual Blocks) [16], InstructPipe [68], DialogueLab [34], AgentHands [44], and Sensible Agent [40] co-authors, which greatly inspired this project along the way. We appreciate all anonymous reviewers', Tim Herrmann and Andrew Helton's thoughtful reviews. We thank Maryam Sanglaji, Max Spear, Adarsh Kowdle, and Guru Somadder, Shahram Izadi for the directional feedback and contribution.

References

[1] 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org. https://www.tensorflow.org/
[2] A-Frame Authors. 2025. A-Frame. https://aframe.io.
[3] Anthropic. 2026. Claude Code. https://code.claude.com/docs/en/desktop
[4] Mixed Reality Toolkit Authors. 2025. MRTK3. https://github.com/MixedRealityToolkit/MixedRealityToolkit-Unity
[5] WebXR authors. 2022. WebXR. https://immersiveweb.dev/
[6] Bezi. [n. d.]. Bezi | AI Assistance for Unity Developers & Studios. https://www.bezi.com.
[7] Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira. 2001. VR Juggler: A Virtual Platform for Virtual Reality Application Development. In *Proceedings IEEE Virtual Reality 2001*. 89–96. doi:10.1109/VR.2001.913774
[8] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: Composable Transformations of Python+NumPy Programs. http://github.com/jax-ml/jax
[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language Models Are Few-Shot Learners. *Advances in Neural Information Processing Systems* 33 (2020), 1877–1901. doi:10.48550/arXiv.2005.14165
[10] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG] https://arxiv.org/abs/2107.03374
[11] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. arXiv:2403.04132 [cs.AI]
[12] Sébastien Crozet. 2020. *Rapier Physics Engine.* https://rapier.rs/
[13] Cursor. 5. Cursor - the AI Code Editor. https://cursor.com.
[14] Fernanda De La Torre, CathyMengying 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 CHI Conference on Human Factors in Computing Systems*. ACM. doi:10.1145/3613904.3642579
[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. doi:10.1109/CVPR.2009.5206848
[16] Ruofei Du, Na Li, Jing Jin, Michelle Carney, Scott Miles, Maria Kleiner, Xiuxiu Yuan, Yinda Zhang, Anuva Kulkarni, Xingyu Bruce Liu, Ahmed Sabie, Sergio Orts-Escolano, Abhishek Kar, Ping Yu, Ram Iyengar, Adarsh Kowdle, and Alex Olwal. 2023. Rapsai: Accelerating Machine Learning Prototyping of Multimedia Applications Through Visual Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 1–23. doi:10.1145/3544548.3581338
[17] Ruofei Du, Alex Olwal, MathieuLe Goc, Shengzhi Wu, Danhang Tang, Yinda Zhang, Jun Zhang, DavidJoseph Tan, Federico Tombari, and David Kim. 2022. Opportunistic Interfaces for Augmented Reality: Transforming Everyday Objects Into Tangible 6DoF Interfaces Using Ad Hoc UI. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems (CHI, 183)*. ACM, 1–4. doi:10.1145/3491101.3519911
[18] Ruofei Du, Eric Turner, Maksym Dzitsiuk, Luca Prasso, Ivo Duarte, Jason Dourgarian, Joao Afonso, Jose Pascoal, Josh Gladstone, Nuno Cruces, Shahram Izadi, Adarsh Kowdle, Konstantine Tsotsos, and David Kim. 2020. DepthLab: Real-Time 3D Interaction With Depth Maps for Mobile Augmented Reality. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (UIST)*. ACM, 829–843. doi:10.1145/3379337.3415881

[19] Sam Earle, Samyak Parajuli, and Andrzej Banburski-Fahey. 2025. DreamGarden: A Designer Assistant for Growing Games From a Single Prompt. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. ACM. doi:10.1145/3706598.3714233

[20] Benj Edwards. 2025. Will the Future of Software Development Run on Vibes. https://arstechnica.com/ai/2025/03/is-vibe-coding-with-ai-gnarly-or-reckless-maybe-some-of-both.

[21] Cathy Fang, Yang Zhang, Matthew Dworman, and Chris Harrison. 2020. Wireality: Enabling Complex Tangible Geometries in Virtual Reality With Worn Multi-String Haptics. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–10. doi:10.1145/3313831.3376470

[22] James Fogarty. 2017. Code and contribution in interactive systems research. In *Workshop HCITools: Strategies and Best Practices for Designing, Evaluating and Sharing Technical HCI Toolkits at CHI*. 1–4.

[23] Daniele Giunchi, Nels Numan, Elia Gatti, and Anthony Steed. 4. DreamCodeVR: Towards Democratizing Behavior Design in Virtual Reality With Speech-Driven Programming. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)* (2024-03). 579–589. doi:10.1109/VR58804.2024.00078

[24] Godot. 2022. Godot Engine. https://godotengine.org/

[25] Google. 2025. Android XR. https://android.com/xr.

[26] Google. 2025. Gemini CLI. https://github.com/google-gemini/gemini-cli.

[27] Google. 2025. Google Gemini. https://gemini.google.com/canvas.

[28] Google. 2025. TensorFlow Hub. https://www.tensorflow.org/hub.

[29] Google. 2026. Google Antigravity. https://antigravity.google.

[30] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. 2025. DeepSeek-R1 Incentivizes Reasoning in LLMs Through Reinforcement Learning. *Nature* 645, 8081 (2025), 633–638. doi:10.48550/arXiv.2506.14245

[31] Philip Guo. 2021. Ten Million Users and Ten Years Later: Python Tutor's Design Guidelines for Building Scalable and Sustainable Research Software in Academia. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. ACM. doi:10.1145/3472749.3474819

[32] Matilda Haneling, Jesper Lundqvist, and Andrii Matviienko. 2025. Beyond Controllers: Exploring On-Body Micro-Gestures for Mixed Reality Adventure Gameplay. In *Proceedings of the 2025 31st ACM Symposium on Virtual Reality Software and Technology*. 1–11.

[33] Fengming He, Xiyun Hu, Jingyu Shi, Xun Qian, Tianyi Wang, and Karthik Ramani. 2023. Ubi Edge: Authoring Edge-Based Opportunistic Tangible User Interfaces in Augmented Reality. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–14. doi:10.1145/3544548.3580704

[34] Erzhen Hu, Yanhe Chen, Mingyi Li, Vrushank Phadnis, Pingmei Xu, Xun Qian, Alex Olwal, David Kim, Seongkook Heo, and Ruofei Du. 2025. DialogLab: Authoring, Simulating, and Testing Dynamic Group Conversations in Hybrid Human-AI Conversations. In *Proceedings of the 39th Annual ACM Symposium on User Interface Software and Technology (UIST)*. ACM. doi:10.1145/3746059.3747696

[35] Erzhen Hu, Mingyi Li, Andrew Hong, Xun Qian, Alex Olwal, David Kim, Seongkook Heo, and Ruofei Du. 2025. Thing2Reality: Enabling Spontaneous Creation of 3D Objects From 2D Content Using Generative AI in XR Meetings. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology* (Busan, Republic of Korea). Association for Computing Machinery. doi:10.1145/3746059.3747621

[36] Yushen Hu, Keru Wang, Yuli Shao, Jan Plass, Zhu Wang, and Ken Perlin. 2024. Generative Terrain Authoring with Mid-air Hand Sketching in Virtual Reality. In *Proceedings of the 30th ACM Symposium on Virtual Reality Software and Technology*. 1–10.

[37] Scott E Hudson and Jennifer Mankoff. 2014. Concepts, values, and methods for technical human–computer interaction research. In *Ways of Knowing in HCI*. Springer, 69–93.

[38] Hugging Face. 2025. Hugging Face – the AI Community Building the Future. https://huggingface.co.

[39] Hirokazu Kato and Mark Billinghurst. 1999. Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System. In *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*. 85–94. doi:10.1109/IWAR.1999.803809

[40] Geonsun Lee, Min Xia, Nels Numan, Xun Qian, David Li, Yanhe Chen, Achin Kulshrestha, Ishan Chatterjee, Yinda Zhang, Dinesh Manocha, David Kim, and Ruofei Du. 2025. Sensible Agent: A Framework for Unobtrusive Interaction With Proactive AR Agent. In *Proceedings of the 39th Annual ACM Symposium on User Interface Software and Technology (UIST)*. ACM. doi:10.1145/3746059.3747748

[41] Jaewook Lee, Jun Wang, Elizabeth Brown, Liam Chu, Sebastian S Rodriguez, and Jon E Froehlich. 2024. GazePointAR: A Context-Aware Multimodal Voice Assistant for Pronoun Disambiguation in Wearable Augmented Reality. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–20. doi:10.1145/3613904.3642230

[42] 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. In *Arxiv*. 9 pages. doi:10.48550/arXiv.2509.25504

[43] Jingyu Li, Qingwen Yang, Kenuo Xu, Yang Zhang, and Chenren Xu. 2025. EchoSight: Streamlining Bidirectional Virtual-Physical Interaction With In-Situ Optical Tethering. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–18. doi:10.1145/3706598.3713925

[44] Ziyi Liu, David Li, Zhongyi Zhou, David Kim, Ruofei Du, and Xun Qian. 2026. AgentHands: Generating Interactive Hands Gestures for Spatially Grounded Agent Conversations in XR. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 24 pages. doi:10.1145/3772318.3790938

[45] Meta. 2025. Immersive Web SDK. https://github.com/meta-quest/immersive-web-sdk.

[46] Nels Numan, Daniele Giunchi, Benjamin Congdon, and Anthony Steed. 2023. Ubiq-Genie: Leveraging External Frameworks for Enhanced Social VR Experiences. In *2023 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. IEEE, Shanghai, China, 497–501. doi:10.1109/VRW58643.2023.00108

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. 8026 – 8037 pages. doi:10.48550/arXiv.1912.01703

[48] Playwright. 2026. Playwright: Fast and Reliable End-To-End Testing for Modern Web Apps. https://playwright.dev

[49] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. 2022. DreamFusion: Text-To-3D Using 2D Diffusion. In *International Conference on Learning Representations*. doi:10.48550/arXiv.2209.14988

[50] George Ramiotis and Katerina Mania. 2025. CONTEXT-GAD: A Context-Aware Gaze Adaptive Dwell model for Gaze-based Selections in XR Environments. In *Proceedings of the 2025 31st ACM Symposium on Virtual Reality Software and Technology*. 1–11.

[51] Jingyu Shi, Rahul Jain, Seunggeun Chi, Hyungjun Doh, Hyung-gun Chi, Alexander J Quinn, and Karthik Ramani. 2025. Caring-Ai: Towards Authoring Context-Aware Augmented Reality Instruction Through Generative Artificial Intelligence. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–23. doi:10.48550/arXiv.2501.16557

[52] Jiaxiang Tang, Zhaoxi Chen, Xiaokang Chen, Tengfei Wang, Gang Zeng, and Ziwei Liu. 2024. LGM: Large Multi-View Gaussian Model for High-Resolution 3D Content Creation. *ArXiv Preprint ArXiv:2402.05054* (2024). doi:10.48550/arXiv.2402.05054

[53] Song Tang, Kaiyong Zhao, Lei Wang, Yuliang Li, Xuebo Liu, Junyi Zou, Qiang Wang, and Xiaowen Chu. 2025. UnrealLLM: Towards Highly Controllable and Interactable 3D Scene Generation by LLM-Powered Procedural Content Generation. In *Findings of the Association for Computational Linguistics: ACL 2025*. 19417–19435.

[54] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: A Family of Highly Capable Multimodal Models. *ArXiv Preprint ArXiv:2312.11805* (2023). doi:10.48550/arXiv.2312.11805

[55] three.js authors. 2022. Three.js. https://threejs.org

[56] Unity. 2022. Unity Game Engine. https://unity.com/products/unity-platform

[57] Unity. 2025. XR Interaction Toolkit. https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/index.html.

[58] Unreal. 2022. Unreal Engine. https://www.unrealengine.com

[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *Advances in Neural Information Processing Systems* 30 (2017). doi:10.5555/3295222.3295349

[60] Aryan Vichare, Anastasios N. Angelopoulos, Wei-Lin Chiang, Kelly Tang, and Luca Manolache. 2025. WebDev Arena: A Live LLM Leaderboard for Web App Development. https://arena.ai/blog/webdev-arena

[61] VRTK Authors. 2025. VRTK. https://www.vrtk.io.

[62] Wonderland GmbH. 2026. Wonderland Engine. https://wonderlandengine.com/.

[63] Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. 2025. Structured 3D Latents for Scalable and Versatile 3D Generation. arXiv:2412.01506 [cs.CV] https://arxiv.org/abs/2412.01506

[64] Yinghao Xu, Zifan Shi, Wang Yifan, Hansheng Chen, Ceyuan Yang, Sida Peng, Yujun Shen, and Gordon Wetzstein. 2024. Grm: Large Gaussian Reconstruction Model for Efficient 3d Reconstruction and Generation. *ArXiv Preprint ArXiv:2403.14621* (2024). doi:10.48550/arXiv.2403.14621

[65] Runxin Yang, Yuxuan Wan, Shuqing Li, and Michael R. Lyu. 2025. 90% Faster, 100% Code-Free: MLLM-Driven Zero-Code 3D Game Development. arXiv:2509.26161 [cs.AI] https://arxiv.org/abs/2509.26161

[66] Lei Yin, Wentao Cheng, Zhida Qin, Tianyu Huang, Yidong Li, and Gangyi Ding. 2026. AutoUE: Automated Generation of 3D Games in Unreal Engine via Multi-Agent Systems. *arXiv preprint arXiv:2603.07106* (2026).

[67] Zhongyuan Yu, Daniel Zeidler, Victor Victor, and Matthew Mcginity. 2023. Dynascape: Immersive authoring of real-world dynamic scenes with spatially tracked rgb-d videos. In *Proceedings of the 29th ACM Symposium on Virtual Reality Software and Technology*. 1–12.

[68] Zhongyi Zhou, Jing Jin, Vrushank Phadnis, Xiuxiu Yuan, Jun Jiang, Xun Qian, Jingtao Zhou, Yiyi Huang, Zheng Xu, Yinda Zhang, Kristen Wright, Jason Mayes, Mark Sherwood, Johnny Lee, Alex Olwal, David Kim, Ram Iyengar, Na Li, and Ruofei Du. 2025. InstructPipe: Building Visual Programming Pipelines in Visual Blocks With Human Instructions Using LLMs. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. ACM, 1–22. doi:10.1145/3706598.3713905

[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*. 1–23. doi:10.1145/3746059.3747676

[70] Fengyuan Zhu, Xun Qian, Daniel Kalmar, Mahdi Tayarani, Eric Gonzalez, Mar Gonzalez-Franco, David Kim, and Ruofei Du. 2025. Beyond the Phone: Exploring Context-Aware Interaction Between Mobile and Mixed Reality Devices. In *2025 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE. doi:10.1109/VR59515.2025.00099

Appendix for Vibe Coding XR

A VCXR60 Dataset Prompts

001: blowing_dandelion Create a beautiful dandelion. when I pick it up and hold with my hands, make the seeds blow away.

002: rainbow_pen Make a pen that draws rainbows in 3D.

003: vine_growth Create a vine growth animation with tendrill curling, leaf, sprouting, wall climbing, flower blooming, and organic spreading pattern.

004: bubble_pop Make a bunch of bubbles that pop when I touch them.

005: origami_cherry_blossoms Origami cherry blossoms that fall when you pick the first petal :cherry-blossom-cowboy: Make pedals collide with physical environment using depth mesh.

006: wall_aquarium Create a new app, When I click a detected mesh of the wall, a virtual fish tank should be created and carved into that detective mesh. The final result should looks like there is a fish tank which inside the wall.

007: cat_platformer Create a cat-themed super mario alike game.

008: planetary_gearbox Create a model of a planetary gearbox and display it in the app. Animate the model to demonstrate how the device works. Add teeth to the gears in the model to make the rotation of the gears easier to see.

009: guitar_tab_tutor Create an app with a model of a guitar that shows dots on strings and frets that correlate to tablature and show you how to play a song. Use Jimi Hendrix little wing as an example.

010: hand_fire Create a WebXR application called "Hand Fire" using three.js and xrblocks that tracks both hands to emit distinct particle-based fire effects, blue fire for the left hand and orange fire for the right. Implement a custom CPU-driven particle system with a shader for the visual effect, and include gesture logic where pinching makes the fire follow the fingertips, while a "thumbs up" gesture freezes the fire in mid-air at its current location. Additionally, provide a spatial UI panel with an "Ignite Hands" toggle button that forces the fire to appear immediately (floating in front of the camera if hands are not detected, or snapping to the palm center if they are), allowing for both manual and gesture-based control of the effects.

011: neon_dodge_arena Create a high-intensity neon dodge arena. The visuals should be dominated by glowing primitives, particle trails, and chromatic aberration. The Hero: A soft-glow orb that leaves a light trail. The Threat: Procedural bullet patterns that pulse to a rhythmic tempo. The Flow: Add a "Bullet Time" meter for tactical slowing and a combo counter that ticks up when players "near-miss" projectiles. The Experience: The game should launch into a self-playing cinematic demo (Attract Mode) that seamlessly transitions to gameplay upon player input. Decoration a variety of spring festival lanterns in my physical environment, when I pinch, spawn a new lantern

012: origami_bird Make an origami bird that flies around the room for a few seconds and then lands on my hand. When I move my hand, it flies away and repeats.

013: voxel_parthenon generate a voxel Parthenon building

014: bird_quiz_game create a bird quiz game in 3D

015: furniture_placement Build a simple 3D scene using basic geometry and orbit controls that features a furniture placement system and a dynamic day-night lighting cycle.

016: alpine_shiba_cabin A photorealistic alpine meadow with wildflowers. Among the evergreen pine trees is a rustic log cabin with a front porch, with a shiba inu outside the cabin in front of the user.

017: stickman_sketch Develop an interactive AR/3D application where users can sketch a stickman in three-dimensional space, featuring a "Finish" toggle that converts the static drawing into a rigged, physics-based character capable of procedural animation and reactive haptics (such as recoiling or stepping back) upon collision with the user's hand.

018: underwater_fish_scene Create a photorealistic underwater scene featuring schools of fish swimming naturally through a complex environment of coral and rocks, utilizing advanced depth rendering for realistic environmental occlusion.

019: procedural_vine_growth Generate a procedural vine growth simulation featuring recursive branching, dynamic tendril curling with physics-based wall-climbing, and time-remapped triggers for leaf sprouting and blooming flowers along an organic spreading path.

020: voxel_garden Create an app with XR Blocks that is delightful and fun. Users can aim with either hand to raycast their cursor against the environment, using depth perception. When they do a pinch gesture, a plant starts growing from the place they aimed at. They can do it many times to create many plants all around their environment. Every time they pinch, a different plant should appear. It can be a palm tree, and oak tree, a cactus, a baobab, a rose. All the plants should be made out of voxels. Their growth should be animated, and it should take 5 seconds for the plant to reach its full size. Plants at their full size should be no more than 2 meters tall so that they fit in indoor environments.

021: waterfall_simulation Develop a waterfall simulation featuring cascading particles, mist spray against rugged rock formations, realistic pool splashing, and a refractive rainbow effect within the mist.

022: weather_prototype Make an immersive weather prototype that can switch between sunny, rainy, thunderstorm, and fog with two buttons

023: edm_concert edm concert environment with blurry lights around

024: flower_watering Create an interactive scene where user can pour the water and water the flower when the hand pinch gesture is detected.

025: four_bar_linkage Create a scene to show a basic four-bar linkage

026: spatial_invaders Create a space invaders game. Spatial Playfield: A grid of animated "alien" meshes that descend toward the user. Controller Interaction: Shooting projectiles from your hands or mouse click

027: spatial_pacman Spatial Pacman. There should be a spatial grid based on depth of the space the user is in, and the movement should happen with user gestures.

028: voxel_tower_defense Create an XR tower defense game where I can place voxel towers on a real table and the towers will attack randomly created voxel monsters.

029: fiery_origami_horse build a fiery horse out of origami that dances to audio from my microphone

030: chill_cats Create a relaxing experience where cats are wandering around in my space around me and just hanging out chill

031: toilet_paper_rain Tons of toilet papers falling from your room ceiling, whenever you look up. Toilet paper rolls should land on the floor. The floor and furnitures should have colliders

032: chunky_holographic_horse Create a 3D scene loading a standard horse GLB model. Set the horse to a continuous gallop loop. Make the horse look slightly fat/chunky using vertex manipulation and apply a glowing holographic shader skin. Add motion trail particles behind the horse to emphasize speed.

033: snowboard_simulator make a mini game that can help me practice snowboarding, simulate an indoor snow resort

034: schrodingers_cat An aesthetically pleasing depiction of Schrödinger's cat in XR. Finger pinch makes a cat (detailed 3D model) go into the box. Approaching the box within 50cm makes the box become two that move to the left and right and the boxes front wall becomes transparent. You see both versions of the cat inside (dead and alive), demonstrating the Quantum mechanics. When you pinch again, one of the states becomes reality. The box opens and you see it either alive or dead. With another pinch you can start again.

035: yellowstone_terrain render and visualize the terrain of the yellowstone national park and add educational markers to famous spot, when clicking on them, show panels to introduce the knowledge for that terrain / feature

036: xr_pomodoro make a productivity app of a tomato clock in xr that I can place on my desk

037: monster_escape_room create an app where user can play escape room, and user can choose to generate the monster he wants. The monster can be a horse gradually evolving into a horse man.

038: snow_white Create a storytelling scene of snow white.

039: butterfly_catching Create an XR butterfly catching game where I hold a net and can use it to catch butterflies

040: tool_shopping Build an XR shopping game where I can pick up from tools like a drill, hammer, and knife. Each item should have a price tag and have physics so when I let go they drop to the floor.

041: marble_run Create a Mixed-Reality Marble Run. The user places tracks to lead a marble from a start point to a goal. Use the depth reticle to anchor "start" and "end" points on different parts. Make the marble bounce off your actual palms to keep it from falling.

042: cat_fireplace_fireflies Create a fireplace with a furry cat playing with fireflies nearby

043: ar_math_graph Open camera, when I give Gemini a math problem, I can watch it visualizes everything in a 3D AR graph

044: spatial_fish Fish swimming across the space

045: voxel_garden Create an app with XR Blocks that is delightful and fun.

Users can aim with either hand to raycast their cursor against the environment, using depth perception. When they do a pinch gesture, a plant starts growing from the place they aimed at. They can do it many times to create many plants all around their environment.

Every time they pinch, a group of plants should appear. In each group, there should always be a big plant in the middle, surrounded by between 3 and 5 smaller flowers in a 30cm radius around the big plant. The big plant can be either a sunflower, a fiddle-leaf fig, or a cactus. These big plants should be 1 meter tall. The other flowers should be smaller, no taller than 20cm when at their full size. They can be flowers of different colors, like roses (pink or red), tulips (reds, yellows, whites, pinks, oranges or deep purples), or dandelions. There should also be a little bit of grass around the flowers. All the plants should be made out of voxels. Their growth should be animated, and it should take 5 seconds for the plant to reach its full size. It is important that the animations work properly to create delight. Each plant within a group should grow individually. All plants should always be oriented upwards, and not based on the normal of the depth map.

If they maintain their pinch and drag, new small plants should be planted along the path they they are drawing with their cursor.

Every time a group of plants is added, there should be a few beautiful piano notes being played (one per new plant, creating a beautiful piano chord) as the plants grow to illustrate the beauty of growing nature. If pinching and dragging to plant flowers along a path, there should only be a note every 3 flowers to not make the sound too overwhelming.

Once plants have been added, there should be some voxel insects (bees and ladybugs) flying around the area. There should be as many insects as there are groups of plants. One new insect should be spawn in the area every time there is a new group of plants. They should start being attracted by the plants and land on them. After a bit, they should take off and fly to a different plant. They should stay keep doing this and visit random plants this way forever.

Do not show a laser pointer coming out of my hands. Make sure you import BufferGeometryUtils as a namespace to avoid import errors.

046: plant_doctor Plant doctor: AI expert can help with care, routine, mock watering tracking with plants you point at.

047: xr_home_decorator XR home decorator: add virtual picture frames, 3D assets, mini-games around your home. Use depth + AI to suggest locations and widgets.

048: social_home_cinema use your living room geometry and digitize/retexture it and decorate it. Invite people to hang out and watch movie together.

049: breathing_exercise breathing exercise with 3D visuals

050: matrix_mesh Matrix Effect with meshes

051: art_exhibition Curate and arrange an art exhibition in the XR world

052: object_capture Scan and copy real world objects into xr world

053: gemini_tutorial Gemini becomes a flying light ball, teach users how to interact with the app.
054: drawing_guess let gemini guess what I'm drawing
055: ping_pong Two player ping pong
056: sticky_notes Sticky Notes around the house
057: yoga_instructor Real time yoga instructor in your room
058: ar_weather "What will the weather be like tomorrow?" - show visual effects in AR that show rain, clouds, etc.
059: sonar_vision "Daredevil" sonar vision simulator
060: superpower_hands Create a delightful app that gives me super powers.
The room should be filled with pixie dust, and pinching with either of my hands should attract particles toward it, and make each particle individually disappears once it reaches it. Particles that have not reached the hand yet should remain visible. The goal is to clear the room from all the pixie dust. The dust should be made of very tiny shiny yellow dots, made from a billboarded quad with a radial gradient to make it look like a little sparkle. The particles are floating in mid air with a slow floating motion, and there should be thousands of them in the room. There should be some physics applied to the particles, so if I start pinching, they are attracted by my hand, and if I release the pinch, the still carry some momentum and slow down gradually.
Pinching should attract the particles that are within a cone that has its apex at the position of my hand, and its main axis going toward the opposite direction of the camera (my head). This allows me to attract particles that are far away by aiming toward them with me hand.
For each particle that reaches the hand and disappears, there should be a large flash a the position of my pinch. This will make clearing the particles more gratifying. The flash is made of a quad, with a radial gradient, the same color as the particle that just got cleared.
The particles should be occluded by the environment and by my hand using the depth sensing capabilities.
There should be a floating text 1.5m away from the user at start time that says "Aim and pinch with your hand to attract and clear the dust." Below, there should be a big percentage that shows the percentage of particles that have already been cleared.
Once all the particles have been cleared, there should be fireworks exploding all around the room. They should keep exploding in random locations forever so that the celebration never stops, until the user closes the app.

B Application Scenarios Prompts

AS1: euler_visualization Visualize Euler's theorem in geometry. Explain vertices, edges, and facets concepts with highlighting using different examples.
AS2: physics_lab create an interactive physics experiment: given different objects on each side of the scale, use different weights (with labels on them) to balance the scale.
AS3: immersive_chemistry create an interactive chemistry lab that users can use hands pinch to ignite and observe three experiments: Ignite methane in air and place a dry, cold beaker over the flame: the flame is pale blue, and liquid droplets form on the inner wall of the beaker. Ignite ethylene in air: the flame is bright, black smoke is produced, and heat is released. Ignite acetylene in air: the flame is bright, thick smoke is produced, and heat is released.
AS4: xr_sports Let me play volleyball with hands and collide with my environment. Volleyballs are textured and launched from a red ring slowly and easier to bounce with the hand.
AS5: chrome_dino create the Chrome Dino game in xr. Dino is voxelized in front of the user, letting every cactus rushing towards the user on a semi transparent lane. Add audio.

C System Prompts of Vibe Coding XR

```
# Role
Act as an expert Creative Technologist specialized in
    three.js, WebXR, and the XR Blocks (xrblocks)
    library.

# Context
You are authoring single-file WebXR experiences. You
    must strictly adhere to the XR Blocks framework
    architecture.

# User Request
Create the app following user request:
    <my_xr_experience>, following the XR Blocks
    examples.

# Engineering Guidelines (Strict)

1. Architecture (Single File):
   - Output a SINGLE index.html file.
   - Logic must be inside a class extending xb.Script
     within <script type="module">.
   - XR Blocks handles XR sessions, window resizing, and
     rendering loop.

2. Dependency Management (Critical):
   - Use the specific versions below. Do NOT hallucinate
     newer versions.
   - Only include imports required for the specific
     request (e.g., do not import TensorFlow unless
     using gestures).
   - Reference Map:
     // omitted

3. Coding Standards:
   - Class Structure: logic must be inside class
     MyScript extends xb.Script.
   - Lifecycle: Use init() for setup and update() for
     the loop. Do NOT use requestAnimationFrame
     manually; xb.Script handles this.
   - Interactions: Use onSelectStart, onSelectEnd,
     onSelecting methods within the class.
   - Coordinates: y is up. z is forward/backward.
     Initialize objects at z = -this.user.objectDistance.
   - Spatial UI: Use XR Blocks 3D UI instead of 2D
     overlay divs.

4. Specific XR Features:
   - Text: Use troika-three-text (Pattern: 1_ui).
   - Models: Wrap 3D models in xb.ModelViewer.
   - AI: If using Gemini/AI, use const API_KEY = "", and
     link the API key to the ai module so Gemini Canvas
     auto-populates it.
```

```
5. Planning:
   - Before generating code, briefly outline the
     xb.Script class structure, member variables needed,
     and the init() vs update() logic flow.

# Reference Examples
Here are some XRBlocks templates, samples, demos, and
     gallery examples...
```

This shows the additional parts other than XR Blocks examples. The latest full prompts with XR Blocks examples are shared in http://xrblocks.github.io/prompts, where the prompt for this work is shared in this Github commit.

D Example snippets of XR Blocks

D.1 Basic Pinch

```
import * as THREE from 'three';
import * as xb from 'xrblocks';

/**
 * A basic example of XRBlocks to render a cylinder and pinch to
 ↪  change color.
 */
class MainScript extends xb.Script {
  init() {
    this.add(new THREE.HemisphereLight(0xffffff, 0x666666,
    ↪  /*intensity=*/ 3));

    const geometry = new THREE.CylinderGeometry(
      0.2,
      0.2,
      0.4,
      /*segments=*/ 32
    );
    const material = new THREE.MeshPhongMaterial({
      color: 0xffffff,
      transparent: true,
      opacity: 0.8,
    });
    this.player = new THREE.Mesh(geometry, material);
    this.player.position.set(0, xb.user.height - 0.5,
    ↪  -xb.user.objectDistance);
    this.add(this.player);
  }

  /**
   * Changes the color of the mesh on a pinch in XR.
   * @param {SelectEvent} event event.target holds controler / hand
   ↪  data.
   */
  onSelectEnd(event) {
    this.player.material.color.set(Math.random() * 0xffffff);
  }

  /**
   * Changes the color of the mesh to blue during pinching in XR.
   * @param {SelectEvent} event The controller / hand event.
   */
  onSelecting(event) {
    this.player.material.color.set(0x66ccff);
  }

  dispose() {
    this.player.geometry.dispose();
    this.player.material.dispose();
  }
}

document.addEventListener('DOMContentLoaded', function () {
  xb.add(new MainScript());
  xb.init(new xb.Options());
});
```

D.2 Basic Camera + VLM

```
export class PoemGenerator extends xb.Script {
  constructor() {
    super();
    this.panel = null;
    this.isProcessing = false;
    this.responseDisplay = null;
    this.button = null;
    this.loadingSpinner = null;
    this.poemGenerated = false;
    this.hasError = false;
  }

  init() {
    this.ai = xb.core.ai;
    this.deviceCamera = xb.core.deviceCamera;

    this.createPanel();
  }

  createPanel() {
    this.panel = new xb.SpatialPanel({
      width: 2.0,
      height: 1.25,
      backgroundColor: '#1a1a1abb',
    });
    this.panel.position.set(0, 1.6, -2);
    this.add(this.panel);

    const grid = this.panel.addGrid();

    const responseRow = grid.addRow({weight: 0.9});
    this.responseDisplay = new xb.ScrollingTroikaTextView({
      text: '',
      fontSize: 0.03,
    });
    responseRow.add(this.responseDisplay);

    const buttonRow = grid.addRow({weight: 0.2});

    const videoRow = grid.addRow({weight: 0.7});
    this.videoView = new xb.VideoView({
      width: 1.0,
      height: 1.0,
      mode: 'stretch',
    });
    videoRow.add(this.videoView);

    const buttonPanel = buttonRow.addPanel({
      backgroundColor: '#00000000',
      showEdge: false,
    });
    this.button = buttonPanel.addGrid().addIconButton({
      text: 'photo_camera',
      fontSize: 0.6,
      backgroundColor: '#FFFFFF',
      defaultOpacity: 0.2,
      hoverOpacity: 0.8,
      selectedOpacity: 1.0,
    });
    this.button.onTriggered = () => this.captureAndGeneratePoem();

    if (this.deviceCamera) {
      this.videoView.load(this.deviceCamera);
    }

    // Create 3D loading spinner
    this.loadingSpinner = new TorusKnotLoadingAnimation();
    this.panel.add(this.loadingSpinner);
  }

  async captureAndGeneratePoem() {
    if (this.isProcessing || !this.ai?.isAvailable()) return;
    this.isProcessing = true;
    this.hasError = false;
    if (this.button) this.button.disabled = true;
```

```
    if (this.loadingSpinner) this.loadingSpinner.show();

    try {
      const snapshot = await this.deviceCamera.getSnapshot({
        outputFormat: 'base64',
      });
      if (!snapshot) {
        throw new Error('Failed to capture video snapshot.');
      }
      const {strippedBase64, mimeType} =
      ↪  xb.parseBase64DataURL(snapshot);
      const image = {inlineData: {mimeType: mimeType, data:
      ↪  strippedBase64}};
      const question =
        'Can you write a 12 lined, lighthearted poem about what you
        ↪  see?';
      const parts = [image, {text: question}];

      const response = await this.ai.query({
        type: 'multiPart',
        parts: parts,
      });
      this.responseDisplay.setText(`${response.text}\n\n`);
      this.poemGenerated = true;
    } catch (error) {
      this.responseDisplay.setText(`Error: ${error.message}\n\n`);
      this.hasError = true;
    } finally {
      this.isProcessing = false;
      // Re-enable button
      if (this.button) {
        this.button.disabled = false;
        this.button.update();
      }
    }
  }

  update() {
    if (this.loadingSpinner) {
      this.loadingSpinner.update(this.isProcessing);
    }
  }
}
```

D.3 XR Object Recognition

```
/**
 * A basic example of using the XR Blocks SDK's world component to
 ↪  detect
 * objects in the real world using Gemini.
 */
class MainScript extends xb.Script {
  init() {
    this.add(new THREE.HemisphereLight(0xffffff, 0x666666,
    ↪  /*intensity=*/ 3));
  }

  /**
   * Runs object detection on select (click in simulator, pinch in XR
   ↪  device).
   * The results of the detection are automatically handled by the
   * ObjectDetector, which will create debug visuals for each
   ↪  detected object.
   * This behavior is enabled when `showDebugVisualizations` is set
   ↪  to true.
   * @param {XRInputSourceEvent} event event.target holds controller
   ↪  or hand
   * data.
   */
  async onSelectEnd() {
    console.log('Running object detection...');
    const detectedObjects = await xb.world.objects.runDetection();

    // `detectedObjects` is an array of THREE.Object3D instances, each
    // representing a detected object. These objects contain the 3D
    ↪  world
    // position and other metadata returned by the detection model.
    if (detectedObjects.length > 0) {
      console.log('Detected objects:', detectedObjects);
    } else {
      console.log('No objects detected.');
    }
  }
}

document.addEventListener('DOMContentLoaded', function () {
  const options = new xb.Options();

  // AI is required for the object detection backend. Please set your
  options.enableAI();

  // Enable the environment camera to provide the video feed to the
  ↪  AI module.
  options.enableCamera('environment');

  // Depth is required to project the 2D detections from the AI
  ↪  module into 3D.
  options.enableDepth();

  // Enable the object detection feature and its debug visualizations.
  options.world.enableObjectDetection();
  options.world.objects.showDebugVisualizations = true;

  xb.add(new MainScript());
  xb.init(options);
});
```