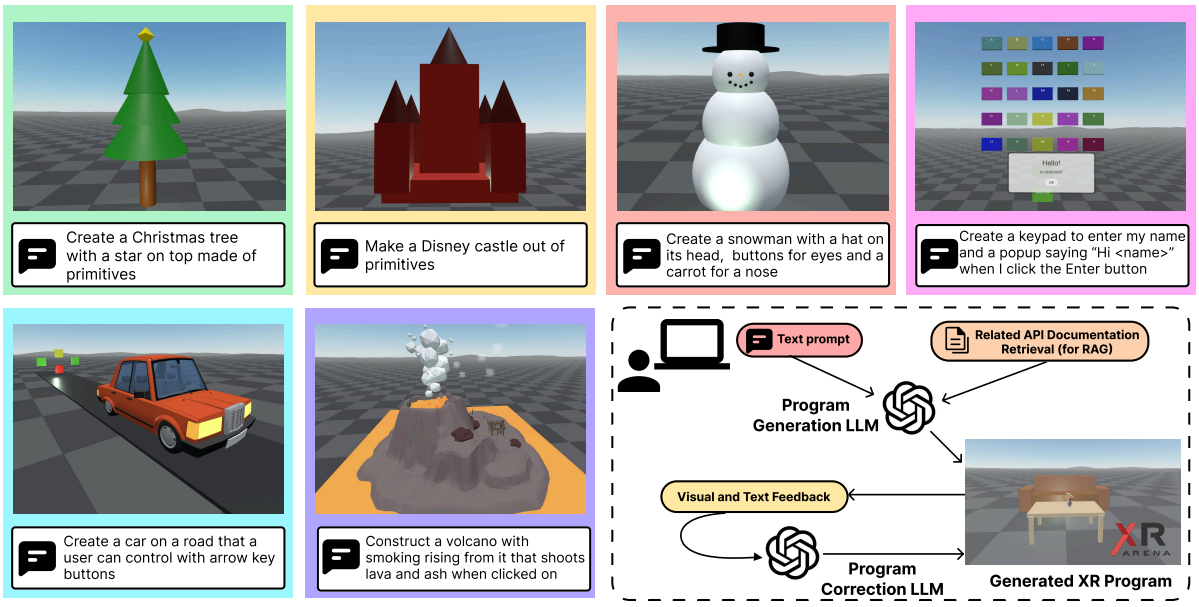
Demo Abstract: GenAssist: Interactive Prompt-Driven XR Program Generation

Sruti Srinidhi* Carnegie Mellon University
Akul Singh* Carnegie Mellon University
Edward Lu* Carnegie Mellon University
Anthony Rowe* Carnegie Mellon University, Bosch Research



(a) Example Programs Generated using GenAssist **(b) High Level System Diagram and Workflow for GenAssist**

Figure 1: **GenAssist overview and examples. (a)** Example XR programs generated from natural-language prompts: procedural modeling with a set of base objects ("Christmas tree with a star"; "castle"), compositional object creation ("snowman with a hat, button eyes, and a carrot nose"), GUI and event handling ("keypad to enter a name and a pop-up saying 'Hi <name>' on Enter"), continuous control ("car that the user drives with arrow keys"), and event-triggered effects ("volcano that smokes and erupts on click"). *Images are static renderings; many examples involve interactions (e.g., clicking, and animations) that are present in the generated programs but not visible in the stills.* **(b)** System workflow: a user writes a text prompt; a Program-Generation LLM (aided by retrieval-augmented API documentation) synthesizes code and generate an XR program; the system then captures visual and textual snapshots; a Program-Correction LLM inspects these artifacts and edits the code; the loop repeats until the requested scene and behavior are achieved, yielding the final XR program.

Abstract

This work demonstrates GenAssist, a system for generating interactive Extended Reality (XR) programs from natural language prompts. Given plain-text descriptions, our system utilizes Retrieval-Augmented Generation (RAG) to fetch relevant documentation and example code, enabling Large Language Models (LLMs) to synthesize and execute XR programs in real time. To ensure outputs align with user specifications, GenAssist incorporates a closed-loop feedback mechanism that uses virtual cameras to inspect the scene and iteratively refine the code, mimicking the manual development cycle of compiling and testing. Beyond static object placement, the system generates functional scripts that animate objects and enable complex user interactions. Our demo showcases the live generation of interactive XR content, allowing users to use voice or text prompts to iteratively build, view, and interact with 3D programs through a headset.

Index Terms: Virtual Reality, Large Language Models, Program Generation

*e-mail: {ssrinidh, akuls, elu2, agr}@andrew.cmu.edu

1 Introduction

With the rapid advancement of Extended Reality (XR) technologies, 3D applications are moving beyond niche domains toward broader everyday use, including immersive entertainment and interactive tools that blend digital and physical environments. Despite this growing adoption, XR development remains complex and time-consuming, often requiring expertise in programming, design, and multiple specialized tools. This high barrier limits accessibility for non-experts and makes rapid prototyping challenging.

At the same time, AI-powered chatbots and assistants have matured into promising tools for simplifying content creation. However, text-only interfaces are insufficient for XR authoring, which depends heavily on visual feedback and iterative refinement. Effective XR generation requires systems that can both produce executable programs and provide real-time, visual insight into the resulting 3D scene so developers can evaluate and adjust their output.

We present GenAssist, a system that enables users to create interactive XR programs in real time using natural language prompts. GenAssist leverages Large Language Models (LLMs) to generate XR code that runs on the ARENA platform [3], which provides a WebXR front-end for executing Python-based XR applications

in browsers or immersive headsets. To improve accuracy and robustness, GenAssist combines a self-correction feedback mechanism based on visual and program-state evaluation with Retrieval-Augmented Generation (RAG) that grounds code generation in platform-specific documentation and examples. Together, these techniques enable rapid, accessible XR prototyping for non-experts.

We demonstrate GenAssist through an interactive XR authoring demo that allows users to create and modify 3D programs using natural language in real time. In this demo, users iteratively describe desired objects, animations, and interactions through natural language prompts, and immediately observe the resulting changes in the XR environment. By combining natural language input, visual feedback, and automatic self-correction, GenAssist enables rapid prototyping of interactive XR programs without requiring prior expertise in XR development. This demonstration highlights GenAssist's ability to lower the barrier to XR content creation while supporting intuitive, iterative program design. [1]

2 System Overview

GenAssist is a system that enables real-time XR program creation from natural language by combining LLM-based code generation, iterative visual feedback, and retrieval-augmented generation (RAG). Our implementation targets the ARENA WebXR platform, but the overall design can be generalized to other XR runtimes that support programmable scene manipulation and scene state access and has been successfully tested with a Unity backend.

2.1 XR Code Generation for ARENA

GenAssist generates Python programs using ARENA's scripting API, `arena-py` [4], which allows dynamic creation and manipulation of XR scenes through a distributed pub-sub architecture. Python is well suited for LLM-based code generation due to its widespread adoption and strong representation in the training data of modern models such as GPT-4o. ARENA's hot-pluggable execution model enables generated programs to be injected and updated in real time without interrupting connected viewers, supporting rapid iteration during interactive use.

2.2 Iterative Scene Correction via Visual Feedback

To improve reliability, GenAssist employs an iterative correction loop that uses visual and spatial feedback from the generated XR scene. After the generated code is executed, the system captures 2D screenshots rendered in a browser using Microsoft Playwright and extracts spatial metadata such as object positions and 3D bounding boxes. This information, together with execution logs, the current program state, and prompt history, is fed back to the LLM to detect errors, resolve spatial inconsistencies, and refine the program to better match the user's input prompts.

2.3 Retrieval-Augmented XR Program Generation

GenAssist uses retrieval-augmented generation to ground code synthesis in platform-specific knowledge and reduce hallucinations. ARENA documentation and example code are embedded in a vector database and retrieved based on semantic similarity to the user's prompt, then injected into the LLM context to leverage in-context learning [1]. A similar approach is used to retrieve relevant 3D assets from ARENA's model repository. We GPT-4o for code generation and Chroma-db for retrieval, enabling accurate, context-aware XR program generation suitable for live demonstration.

3 Evaluation

We evaluated GenAssist across three dimensions: user experience, generation accuracy and system overhead.

[1] Detailed system overview, evaluation, results and visuals can be found at https://www.srutisrinidhi.com/GenAssist/

User Experience: A user study with 18 participants showed an overall low NASA Task Load Index (TLX) of 36.6/100, indicating relatively low cognitive load , and a System Usability Score (SUS) of 69.6, suggesting good usability for non-experts.

Generation Accuracy: We curated a taxonomy of 50 prompts across five categories: (1) Object Placement, (2) Animations, (3) Interactivity, (4) Complex Programs, and (5) Iterative Generation. We recruited 15 raters to evaluate programs on a 10-point scale across four metrics: Prompt Match, Object Placement, Functionality, and Overall Quality. Using a linear mixed-effects model to control for rater bias and prompt difficulty, GenAssist achieved a predicted rating of 7.304, outperforming GPT-4o alone (5.377) and the prior state-of-the-art LLMR (3.479) [2]. We also verified functional validity through objective boolean checks (e.g., code execution and object existence), where GenAssist consistently satisfied more criteria than ablated variants.

System Overhead: The system is efficient for live generation, with an average initial generation time under 10 seconds, and each subsequent correction cycle being approximately 14 seconds, which is 3.7x faster than prior state-of-the-art systems.

4 Demonstration

We demonstrate GenAssist through a live XR authoring experience in which users create and modify XR programs using natural language while wearing a VR headset.

Users wear a Meta Quest 3 headset and enter an ARENA scene where they can interact with and view the XR content they author. They are given a small microphone they can clip on which they can speak into to provide natural language prompts to create the XR program, like "create a red car out of primitives" or "make a house whose door opens when I get close to it". GenAssist then generates and executes the corresponding XR program in real time. The resulting program is rendered immediately on the headset, allowing users to explore the generated content from an immersive first-person perspective. For audience engagement, the XR view is also mirrored on an external monitor so that observers can see the generated scene and interactions without wearing a headset.

During the demonstration, the system specifically highlights its recovery from common failure modes, such as spatial inconsistencies or incorrect event-handler logic. If a prompt results in a bug (e.g., an interaction that fails to trigger), the iterative feedback loop utilizes execution logs and visual snapshots to detect the discrepancy. The correction loop then re-synthesizes the faulty code block to align with the user's intent. This self-correction process allows users to iteratively refine their programs through additional spoken prompts without manually editing code, supporting intuitive prototyping while maintaining robustness through continuous correction.

References

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020. 2

[2] F. De La Torre, C. M. Fang, H. Huang, A. Banburski-Fahey, J. Amores Fernandez, and J. Lanier. Llmr: Real-time prompting of interactive worlds using large language models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pp. 1–22, 2024. 2

[3] N. Pereira, A. Rowe, M. W. Farb, I. Liang, E. Lu, and E. Riebling. Arena: The augmented reality edge networking architecture. In *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 479–488, 2021. doi: 10.1109/ISMAR52148.2021.00065 1

[4] A. XR. arena-py: Python library for accessing the arena. https://github.com/arenaxr/arena-py. 2