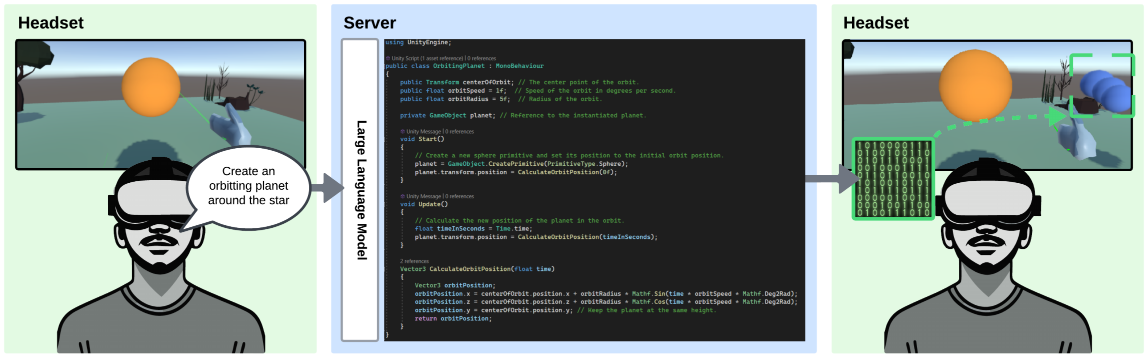
DreamCodeVR: Towards Democratizing Behavior Design in Virtual Reality with Speech-Driven Programming

Daniele Giunchi[*] Nels Numan[†] Elia Gatti[‡] Anthony Steed[§]

University College London, UK



```
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class OrbitingPlanet : MonoBehaviour
{
    public Transform centerOfOrbit; // The center point of the orbit.
    public float orbitSpeed = 1f;  // Speed of the orbit in degrees per second.
    public float orbitRadius = 5f;  // Radius of the orbit.

    private GameObject planet; // Reference to the instantiated planet.

    Unity Message | 0 references
    void Start()
    {
        // Create a new sphere primitive and set its position to the initial orbit position.
        planet = GameObject.CreatePrimitive(PrimitiveType.Sphere);
        planet.transform.position = CalculateOrbitPosition(0f);
    }

    Unity Message | 0 references
    void Update()
    {
        // Calculate the new position of the planet in the orbit.
        float timeInSeconds = Time.time;
        planet.transform.position = CalculateOrbitPosition(timeInSeconds);
    }

    2 references
    Vector3 CalculateOrbitPosition(float time)
    {
        Vector3 orbitPosition;
        orbitPosition.x = centerOfOrbit.position.x + orbitRadius * Mathf.Sin(time * orbitSpeed * Mathf.Deg2Rad);
        orbitPosition.z = centerOfOrbit.position.z + orbitRadius * Mathf.Cos(time * orbitSpeed * Mathf.Deg2Rad);
        orbitPosition.y = centerOfOrbit.position.y; // Keep the planet at the same height.
        return orbitPosition;
    }
}
```

Figure 1: DreamCodeVR: AI-code-based behavior generator triggered by user's speech that can change the appearance, behavior and state of a live running Unity VR application.

Abstract

Virtual Reality (VR) has revolutionized how we interact with digital worlds. However, programming for VR remains a complex and challenging task, requiring specialized skills and knowledge. Powered by large language models (LLMs), DreamCodeVR is designed to assist users, irrespective of their coding skills, in crafting basic object behavior in VR environments by translating spoken language into code within an active application. This approach seeks to simplify the process of defining behaviors visual changes through speech. Our preliminary user study indicated that the system's speech interface supports elementary programming tasks, highlighting its potential to improve accessibility for users with varying technical skills. However, it also uncovered a wide range of challenges and opportunities. In an extensive discussion, we detail the system's strengths, weaknesses, and areas for future research.

Keywords: large language model, VR, rapid prototyping, low code programming, speech programming

Index Terms: Human-centered computing—Virtual reality Human-centered computing—Natural language interfaces

1 Introduction

The interplay between Virtual Reality (VR) and Artificial Intelligence (AI) presents a unique conjuncture, providing a platform for innovative immersive experiences [59,5,62]. AI has been seamlessly woven into the fabric of VR applications, utilizing its generative ability to create captivating multimedia content, such as images, audio, text, and 3D meshes [52, 61, 4, 57]. AI has significantly influenced VR, driving the development of visually and auditorily impressive environments and characters through algorithms that manipulate static content generation [15, 28, 16, 36]. While this integration has undoubtedly increased VR's sensory and creative appeal, its potential remains partially untapped, particularly concerning the possibilities of crafting behaviors and procedures that are integrated with the virtual environment and respond to user interactions.

The increasing power of real-time systems and new data-driven capture pipelines enable artists and designers to build increasingly complex static content. However, they often lack the robust capabilities to intertwine these elements with responsive, dynamic behaviors. Development of behaviors is still stuck within integrated-development environments driven by off-line code editing or visual programming [65, 17]. While several works have been proposed to change behavior at runtime in VR [64, 63, 75], they tend to offer limited functionality and applicability, such as replacing a single object property or shape. We introduce DreamCodeVR (Figure), a VR system based on Unity and Ubiq [26, 57] that bridges the gap between static content creation and dynamic behavior generation in VR environments. Using LLMs [14, 66], our approach leverages the generation of procedural code, aligning with the principles of low-code programming [35]. Unlike conventional programming approaches, low-code platforms leverage alternative methodologies—like visual or natural language processing (NLP) techniques, enabling users to create applications without necessitating extensive coding expertise. Our adaptation of this methodology to VR explores a paradigm where users, prioritizing their creative intent, can articulate desired outcomes without delving into the intricacies of implementation or mastering programming languages. DreamCodeVR introduces two fundamental contributions to VR content creation. Firstly, it facilitates the generation of procedural code, simplifying the management of real-time behaviors, visual attributes, and application logic. Secondly, it provides an incremental integration of this code into an active application, allowing for in-the-loop testing and adjustment without requiring an application restart, thus streamlining the development process. This is enabled through its pipeline that combines text-to-speech (TTS), code generation, and real-time compilation and loading of executable components while the user is immersed in the virtual environment. DreamCodeVR represents a preliminary

[*]e-mail: d.giunchi@ucl.ac.uk
[†]e-mail: nels.numan@ucl.ac.uk
[‡]e-mail: elia.gatti@ucl.ac.uk
[§]e-mail: a.steed@ucl.ac.uk

exploration of behavior generation within VR applications, serving as a platform for further research and development into the goal of elevating each user to the status of citizen-developer [43,60], referring to individuals who create applications without an extensive software development background. This paper's contributions include:

- The introduction of a VR system that enables users without coding expertise to generate behavioral components of scene elements, highlighting a critical advancement in user-driven design.
- A detailed exposition of a modular and extensible system design, proficient in real-time injection of AI-generated code via speech within a running Unity VR application.
- The presentation of results derived from a preliminary user study, investigating the impact of AI-generated procedures on user experience and engagement.
- A discussion on potential issues, limitations, and recommendations for future work.

2 Related Work

2.1 VR Programming Systems

Programming systems for VR applications must address a very wide variety of features, interfaces, and requirements [20]. Such systems should produce applications capable of generating high-fidelity output across multiple modalities while adhering to robust end-to-end timing guarantees to ensure a smooth and consistent user experience. Additionally, they must interface with various physical devices, potentially involving multi-process or distributed programming scenarios.

Although different pieces of equipment might have their own application programming interfaces (APIs) or server software, there have been substantial efforts aimed at providing standardized interfaces to devices. The enduring Virtual Reality Peripheral Network (VRPN) continues to be used for certain hardware [72]. Special-purpose commercial software has served this market for multiple decades [78], though concurrent open-source efforts are no longer as widely used (e.g. Diverse [42] and VRJuggler [11]). Additionally, efforts have been undertaken to establish content standards such as X3D [39], seeking to extend these to platforms that support diverse requirements [8]. More generalized support in web browsers is provided by the WebXR API [77].

Despite the various open initiatives, the majority of VR application development takes place within commercial game engine tools such as Unity or Unreal. Rather than a mere set of abstraction APIs, these tools provide visual editors that enable the assembly of assets, management of scenes, and structuring code within a visual editor. The resulting assets and code are built into a runtime engine and asset set that forms a packaged application. The prevailing model of distribution of VR applications entails downloading and installing these packages. Furthermore, updates of the runtime code are typically conducted through patching or updating.

2.2 Dynamic Code

In applications designed to cultivate creativity and those incorporating diverse sub-worlds, such as social platforms, the capacity to update code during runtime dynamically is beneficial for facilitating real-time user interaction, continuous content creation, and seamless implementation of system updates or modifications. Historically, numerous games incorporated their own scripting languages or environments. Early multi-user dungeon (MUD) systems provided players with the capabilities to edit and extend the system [7]. Furthermore, various early game engines included a scripting engine, facilitating designers to program in a language-oriented towards game content, as opposed to utilizing low-level abstractions, with the ability to interpret the language at runtime. Notable examples include *SCUMM* from LucasFilm/LucasArts [9] and *QuakeC* from iD Software [2]. This pattern was embraced by early VR systems (e.g., DIVE, which used the Tcl language [27]) and augmented reality (AR) systems (e.g., Dart AR [50]). In the context of social platforms, a seminal example is the Linden Scripting Language for Second Life (e.g., see critique [19]).

As VR platforms evolve and the interest in content creation increases, a relevant question arises concerning the immersed user's capability to alter and extend application code. Unity demonstrated extensive editing capabilities with their *EditorVR* framework [24] to allow users to create content directly in VR. Rec Room incorporated a built-in visual scripting language, *Circuits*, enabling constrained editing of object behavior [13], while VRChat employed a similar system called *Udon* [76]. Moreover, RealityFlow is an open-source implementation of an immersive collaborative visual programming environment [55]. To safeguard system integrity and user experience, such languages and frameworks are somewhat restricted in their domains and capabilities, typically only enabling asset creation, modification, and deletion.

Modern runtimes afford the capability to compile and execute code dynamically, enhancing the flexibility and interactive nature of VR environments. DreamCodeVR leverages the Roslyn compiler [23] for dynamic C# code compilation within Unity, enabling the integration of user-driven, real-time modifications into virtual environments. A comprehensive description of the utilized mechanism is provided in Section 3.3.

2.3 LLM for Code Generation

LLMs have recently emerged as a powerful tool for code generation, having been trained on extensively large datasets containing code in various programming languages [66,47]. However, LLMs can sometimes generate code that is neither idiomatic nor efficient, leading to potential difficulties in readability, maintainability, and performance when compared to human-written code [40].

Despite these challenges, LLMs can be a valuable tool for software developers and have sparked the development of various approaches to optimize their utility. Consequently, numerous commercial and open-source LLM-based code generation tools are now available [25, 32]. A notably prominent tool is GitHub Copilot, which integrates with Visual Studio Code and other popular code editors to suggest code completions, generate new code, and facilitate code translation between programming languages. It is built on the OpenAI Codex model, an LLM fine-tuned on a large code dataset. Another widely-recognized LLM-based tool is Tabnine, which provides code completions, refactoring suggestions, and code generation capabilities, along with features that allow training on personal code corpora. Other available LLM-based code generation tools include Amazon CodeWhisperer, CodeT5/T5+, and PolyCoder. However, despite the advancements of LLMs, they occasionally generate code that is incorrect, inefficient, or unexecutable. Furthermore, another challenge that arises from utilizing LLMs for code generation is the inherited bias from their training datasets, which reflect the biases of their creators [18].

Nonetheless, LLMs possess the potential to substantially enhance the productivity of software developers by automating various coding tasks. These tasks include generating boilerplate code, refactoring existing code, and translating natural language descriptions into new code functionalities [48]. Moreover, LLMs can adeptly create code for the implementation of new features, the correction of bugs, and the execution of other labor-intensive tasks that are commonly considered time-consuming. In our work, we utilize LLMs to derive code from natural language descriptions of compact functionalities, thereby facilitating code generation for users unfamiliar with programming or seeking to expedite their work.

2.4 LLMs in VR

LLMs have the potential to transform interactions within virtual worlds, establishing avenues for more immersive and interactive experiences, and fostering the development of innovative applications in VR [57]. For instance, the system *Ubiq-Genie* underscores the utility of employing an LLM in the backend of VR applications, by driving a virtual agent that orchestrates a quiz game among multiple users within a social VR application. LLMs in VR mainly demonstrate their utility to users in two distinct phases: the development phase and the execution phase, each presenting unique attributes and challenges. In the development phase, users interact with LLMs through an interface, utilizing its output in code or other textual forms for diverse development processes, including integrating the generated code into their programming environment.

For this purpose, several LLM-based tools were developed to streamline such tasks [37, 49]. Conversely, during the execution phase, LLMs are directly integrated into running applications, relieving users from additional integration or interpretation steps. Despite inherent challenges, this introduces the potential to facilitate the coherent creation and orchestration of program logic while immersed in a running VR application. Roberts et al. [63] utilized LLMs as a context engine for object-to-object interaction to facilitate prompt-based creation of VR content, using the example of "Codex VR Pong". This game showcased non-deterministic game mechanics, where generative processes create both static content and non-trivial interactions between 3D objects. In very recent parallel work, researchers introduced LLMR [21], a framework that integrates advanced scene understanding and task planning within Unity in MR supported by an LLM and runtime compilation, showing its effectiveness in modifying diverse virtual elements. One particularly promising application of LLMs in VR lies in games, where LLMs can be used to control game logic dynamically, generate realistic dialogue and behaviors for characters, and provide players with contextually relevant information and assistance, enhancing both the interactivity and immersive experience of games. For example, Van Stegeren et al. fine-tuned an LLM to generate dialogues of non-player characters (NPCs) that assign users tasks (i.e., "quests") in a role-playing game [74]. Similarly, Volum et al. employed LLMs to author scripts that drive interactive NPCs capable of assisting Minecraft players by answering questions, performing actions (e.g., crafting an item), and engaging in multi-turn conversations with the user [75]. When two objects (e.g., fire and ice) interact in their system, an LLM is used to infer the interaction outcome. Based on the inferred result, a 3D mesh model is retrieved from an existing database and introduced into the environment.

Applications utilizing LLMs, including the aforementioned examples and DreamCodeVR, which require real-time interaction, need LLMs to generate high-quality code with minimal latency to assure a seamless user experience. Specifically, in the context of DreamCodeVR, code generation and compilation can be identified as two primary tasks that can potentially impede user experience if not conducted promptly. While open-source LLMs offer the flexibility to operate on local or low-end hardware, server-based or cloud-based LLMs are often preferable for ensuring optimal quality and minimized latency, particularly in resource-intensive VR applications. The substantial computational power provided by server-based and cloud-based LLMs ensures fast and reliable processing, essential for sustaining immersive and interactive VR experiences. Consequently, DreamCodeVR adopts a server-based application architecture to facilitate the utilization of any LLM that can be run on local server hardware. Notably, we deploy a cloud-based LLM in our user study, prioritizing high-quality and low-latency model responses.

2.5 Speech Interaction in VR

Speech interfaces offer significant utility in scenarios where keyboard support is unavailable or unreliable, particularly when integrated with embodied VR to enable body-centric commands [79, 41, 46]. Speech input can offer notable advantages over traditional keyboard or gesture inputs [69], particularly since the latter often requires extensive practice. However, speech-based interaction encompasses multiple challenges, including speech recognition, phrase interpretation, and user interaction dynamics [6, 70]. The implementation of speech interaction varies significantly across different studies, with user engagement closely linked to the specific task at hand and the capabilities of the system components [31, 30]. The intersection of speech interaction and VR dates back several decades, as evidenced by pioneering works [53, 54] leading to the development of multi-modal systems with two distinct approaches: *fully interactive speech* or *command-and-control*. The fully interactive approach was commonly speaker-dependent in early work, necessitating a training phase known as enrollment due to the vast array of words and sentences involved [56]. Before the recent advancements in NLP, researchers often relied on the 'Wizard of Oz' methodology [44, 29] to circumvent the technical constraints of speech interfaces in the recognition and processing of phrases. In contrast, command-and-control systems were designed to function effectively with a limited set of predefined commands. This approach sacrificed conversational flexibility for reliability and ease of use, as it more often did not necessitate user-specific training.

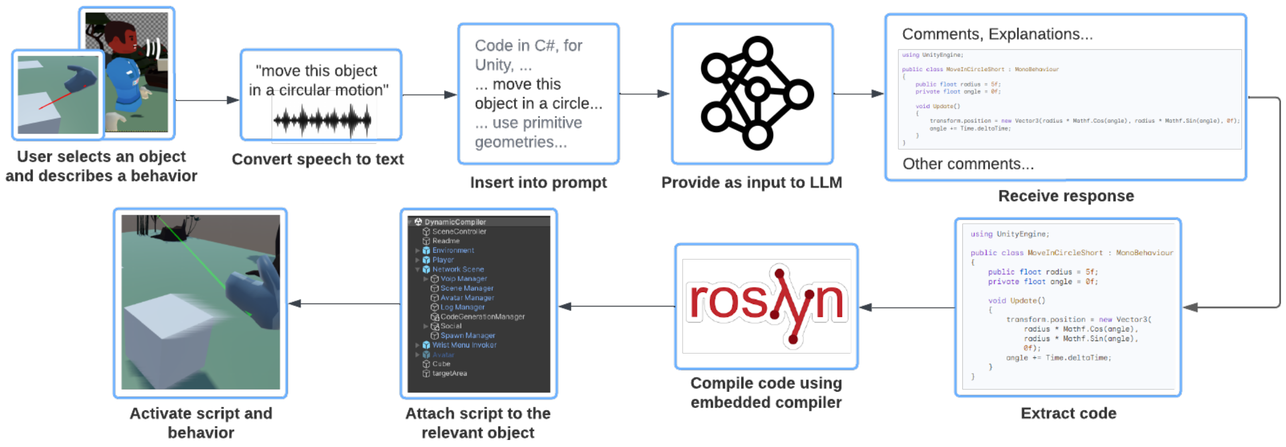
Leveraging its benefits, speech interaction has been explored in various fields such as medicine for treating social phobia [73], collaborative data analysis [12], and managing digital twins of complex machinery [68]. With the rise of advanced deep learning NLP models, speaker-dependent systems have become less relevant. Present-day services like Google Speech-to-Text, IBM Watson, Amazon Transcribe, and Microsoft Cognitive Services can receive and process audio streams, providing real-time transcription. Our system harnesses these capabilities to generate input to an LLM tasked with generating functional code in C#.

3 System Design and Implementation

We adopted the Unity framework and utilized the open-source social VR platform, Ubiq [26], which provides functionalities dedicated to the rapid development of networked VR applications. Ubiq has demonstrated efficacy in seamlessly integrating third-party services into VR solutions [57], rendering it a logical choice for harnessing external AI resources.

Ubiq-Genie comprises three primary entities: "Unity Scene," "Application," and "Services." While the Application and Services operate on a backend server, the Unity Scene contains the 3D environment with which the user interacts. Each Application is connected to its unique scene in this configuration, thereby serving as an orchestrator of backend services. Services are defined as Node.js components (e.g., text-to-speech service, LLM service) and act as modular entities capable of invoking processes written in other languages (e.g., Python). Consequently, each Application establishes a connection with a server-side pipeline, facilitating interaction with various services and enabling data exchange among them.

DreamCodeVR defines a two-service pipeline: the first service transcribes the audio speech via Azure speech-to-text, and the resultant string is supplemented with a predetermined prompt to guide the subsequent text generation process. The second service manages calls to an LLM API or model, passing the input string received from the speech-to-text service and responding with an answer that comprises both an explanation and code. In our configuration, we primarily utilize the cloud-based API of GPT-4 (see Section 3.2); however, this service provides flexibility in utilizing any LLM that can be run through any programming language capable of communicating over the I/O stream (e.g., Java, Python, C++). We process the output of the LLM by extracting the code. Subsequently, the code, in the form of a string, is sent back to the user's device, where it is compiled and executed with the Roslyn C# compiler Unity



```
using UnityEngine;

public class MoveInCircleShort : MonoBehaviour
{
    public float radius = 5f;
    private float angle = 0f;

    void Update()
    {
        transform.position = new Vector3(radius * Mathf.Cos(angle), radius * Mathf.Sin(angle), 0f);
        angle += Time.deltaTime;
    }
}
```

```
using UnityEngine;

public class MoveInCircleShort : MonoBehaviour
{
    public float radius = 5f;
    private float angle = 0f;

    void Update()
    {
        transform.position = new Vector3(
            radius * Mathf.Cos(angle),
            radius * Mathf.Sin(angle),
            0f);
        angle += Time.deltaTime;
    }
}
```

Figure 2: Pipeline architecture that shows the following sequence of steps. After the user describes the behavior or the procedure he wants to attach to the scene, we convert the audio into text by Azure speech transcription service and enrich the input text via prompt engineering. LLM uses such text and produces an answer comprising explanations and code. After extracting the code, we pass the string back to the VR application that includes a C# compiler that generates the bytecode in memory and provides our controller with a way to attach it as a component to the selected object. The selected object will promptly be provided with such additional behavior.

asset [38], distributed as a dynamic library in the VR App. The assembly now present in memory is then attached by an orchestrator class to a scene's object, manually chosen by the user beforehand, or to a default controller. The functionality is then immediately available as following the Unity script lifecycle flowchart [22]. The entire pipeline is depicted in Figure 2.

Two distinct scenes were designed for the usage of DreamCodeVR: the first, equipped with rudimentary objects to facilitate behavioral testing, and the second, an empty canvas for creating objects along with their corresponding behaviors.

3.1 Prompt Engineering

Utilizing LLMs to generate code is not a straightforward task. The output from an LLM can significantly vary depending on the formulation of the prompt. To minimize output fluctuations and ensure alignment of the generated code with the desired task and requisite framework, meticulous engineering of the prompt is essential. One approach to prompt engineering, particularly for code generation, involves utilising a system prompt template. This system prompt should outline the desired code structure, style, and additional rules to constrain the output to a specific language or framework.

We refined our prompt through several iterations to address various issues: absence of code in the answer, improper separation of code from the answer, code that failed to compile, code that compiled but was not adequately tailored for the Unity framework, and code that did not function correctly even when active in the scene. We incorporated additional visual representation, positioning, and shape crafting rules. Therefore, we constructed our prompt by considering the following criteria:

- C# language
- Code always present
- Avoid Unicode characters that can tamper the building process
- Be compliant with the Unity framework
- The behavior will be embedded as a single and unique Component (inheriting from the `MonoBehaviour` class)
- Avoid conflicts such as name clashing for the components
- Potential new object needs to be visible with a MeshRenderer
- Spatial relationships are allowed for the selected object or, if no object is selected to the Controller Object
- Shapes have to be simplified to primitive Geometries present in Unity
- A common tag for created objects needs to be present (to ensure selection)
- Avoid classic input mechanisms such as a keyboard or mouse.

We achieved the best results with the following prompt:

```
  Write a script in C# for Unity, so a class that
inherits from MonoBehaviour.  In your answer always
write code between tag “csharp and tag“<USER_INPUT>.  Use
a Component name that does not conflict with previous
used or default Component names from Unity.  Always set
the tag to game.  Do not use interaction from devices
such as keyboard or mouse.  Use ASCII code and no Unicode
in the answer.  Create only one MonoBehaviour.  If 'this'
or 'that' is used, assume the object is still present and
do not create other instances.  If a new instance of an
object is created it should have parent 'transform', and
infer the shape to primitive geometries present in Unity.
Attach a MeshCollider.
```

This prompt has been tailored for the basic 3D scenario in DreamCodeVR, signifying that additional development is necessary for handling more complex contexts or tasks. In future work, a dynamically adaptive prompt could be developed, incorporating information from the scene (e.g., the Unity scene graph, component lists, scene constraints) that can change during the user experience.

3.2 Code Generation Comparison

To determine which available LLM is most suitable for application within DreamCodeVR, we qualitatively compared five LLM models: a local LLM implementation based on the GPT4All [3] Python package, with two models running on a CPU with high performances [1] (Falcon with 7B parameters and Nous-Hermes2 with 13B parameters); a local GPU version of CodeLlama (CodeLlama-7b-Instruct-hf with 7B parameters); GPT-3.5-turbo cloud-based API; and the GPT-4 cloud-based API. The local models were deployed on a PC with an RTX 4060 GPU, an Intel i9 processor, and 64 GB of RAM.

Table 1: Comparison of five different models related to accuracy and time to answer. (l) denotes local LLM, (r) is a remote-serviced LLM

Model	Accuracy (%)	Time to answer (sec)
Falcon (l)	20%	143.0
Nous-Hermes 2 (l)	20%	89.7
CodeLLama (l)	40%	453.8
GPT 3.5 API (r)	70%	**15.1**
GPT 4 API (r)	**90%**	29.5

For each model, we constrained the maximum number of tokens to 1000 and set the temperature to 0.7. Notably, the token limit for CodeLlama was reduced to 500 due to significant latency experienced with 1000 characters (exceeding 1400 seconds). We executed ten predefined descriptions of object behaviors involving alterations in visual appearance, object creation (singular and multiple), dynamic changes, inter-object relationships, destruction, temporal requirements, and task combinations. The generated code was evaluated for each description, considering a task "passed" if all prompt requirements and conditions were achieved with correct functionality in the resultant code. This evaluation encompassed visual inspection of the code and examination of the prompt generation rules and the ensuing behavior in the Unity scene. The average accuracy per model and the average time (in seconds) needed to generate the script are reported in Table 1.

In VR applications, latency is often a pivotal factor as users can perceive minor delays in feedback. The cloud-based APIs of GPT-3.5 and GPT-4 showed fast response times and accuracy, positioning them as viable options for VR applications. Considering the paramount importance of precise behavior, the DreamCodeVR implementation uses the GPT-4 API model, demonstrating the highest precision and the second-lowest response time. Conversely, CodeLlama had the poorest response time and showed inadequate accuracy in generating valid code for the Unity environment.

3.3 Real-Time Code Building on Untethered Devices

The .NET Compile Platform SDK allows a runtime to include functionality to analyze and compile C# and Visual Basic code. This allows for creating an assembly, crucial for tasks such as code generation from templates, transpiling code from one language to another, refactoring, and performing code analysis. On the other hand, the compiler pipeline consists of four phases:

1. **Parsing**: The code is tokenized and parsed into syntax that conforms to the language's grammar.
2. **Declaration**: The code and imported information are parsed to generate named symbols.
3. **Binding**: In the code, identifiers are mapped to symbols.
4. **Emit**: An assembly with all the information the compiler generates is emitted.

Each phase of the SDK exposes a specific object model, offering various access levels. For instance, the parsing phase reveals a syntax tree, the declaration phase unveils a hierarchical symbol table, the binding phase presents the compiler's semantic analysis results, and the emit phase provides an API that yields Intermediate Language (IL) byte codes. To facilitate code compilation at runtime, we employ the compiler API to instantiate a compiler object, which is then used to compile the code. We directly access the in-memory generated assembly, ensuring it exposes an interface compatible with the Unity environment. Notably, the latency incurred during code building and loading within our environment is marginal compared to the time required for our employed LLM to generate a result.

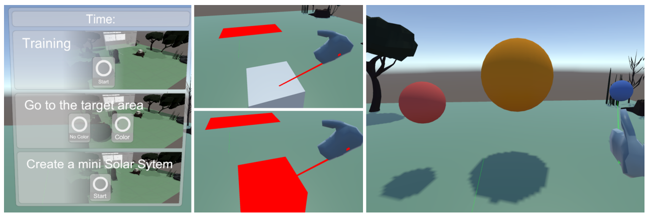


Figure 3: VR Scenario presented to the users. After a training session selected from the main menu (on the left), the user can provide behaviors on a cube to reach the target (without color or with color) as displayed in the pictures in the middle; on the right, the third session, where the user has to create a small solar system with a planet orbiting around the star.

By using the compiler pipeline, we can construct a component's functionality in memory, which can be attached to a scene-defined object within the Unity environment. We perform the code compilation and assembly loading stages directly on the untethered client device, tasking the server solely with the LLM-driven code generation. Utilizing the compiler API, we generate a C# class representing the component functionality by inheriting from the `MonoBehaviour` class (the base class for all Unity components), which is then added to the Unity scene by attaching it to an object. This automated method replicates results akin to a developer manually adding a Component script in the Unity Editor, connected to an object via menu or through drag-and-drop in the Inspection panel. Once a component is attached to an object, its execution adheres to the same policies as a newly added scene component. If code is defined inside the `Awake` or `Start` function, it executes once; if embedded in the `Update` function provided by the `MonoBehaviour` interface, it executes each frame. Our method uniquely enables runtime incorporation and direct formulation within a standalone client device, distinct from traditional manual script addition during development. Moving forward, the compiler pipeline offers a pathway to enhance component functionalities and create a robust environment adept at managing error handling, object communications, functionality sequences, complex interactions, and intricate object generation.

3.4 User-System Interaction

The quality of user interaction with DreamCodeVR is highly dependent on its feedback mechanisms for communicating errors and incorrect behaviors to the user. Leveraging its pipeline, DreamCodeVR transforms user inputs into executable scripts with a latency tolerable for real-time interaction (Section 3.2). However, the system is not immune to errors in behavior execution, as classified in Section 6.1. Error handling in DreamCodeVR occurs across multiple layers of the system. Server response errors or compiler errors immediately prompt a halt in the pipeline process, preventing further complications. As a result, users will not observe any unintended effects within the VR environment related to dysfunctional scripts or assemblies, thus maintaining the scene's integrity and readiness for subsequent requests. Furthermore, DreamCodeVR incorporates an interactive debug display, currently utilized by experimenters to monitor and troubleshoot the creation process as driven by user input. This feature systematically captures and displays exceptions, particularly those arising from assembly building and loading operations. Recognizing the necessity for a more transparent system, future work should investigate strategies to tailor debug output to be intuitive for users without programming expertise and to provide varying levels of detail according to user skill. Moreover, the system necessitates an undo operation to remove unwanted behaviors that have passed the code generation, building, and loading stages.

4 Preliminary User Study

We conducted a user study with seven participants (four male, three female, mean age 32.6 years), adhering to recommendations for "debugging" tests outlined by Bevan et al. [10]. All users had previous experience with VR consumer gaming but no experience with programming. This specific profile of users was selected as it best represented the type of users we intend to target with our exploration: VR enthusiasts who do not possess the specialized knowledge needed to create VR environments through programming. The primary objective of the study was not to validate every facet of the system fully. The study focused on assessing our specific approach to immersive speech programming, exploring its potential and limitations, especially for novice users unfamiliar with VR software development. The insights gathered are intended to inform further iterations of our application and provide valuable insights for researchers aiming to build upon this work.

During the study, participants were instructed to sit comfortably on a chair and wear a Meta Quest 2 HMD. Upon doing so, they entered a virtual environment that simulated an outdoor setting, which included green grass, a tree, and a wooden log on the ground. A menu was displayed in front of the participants, allowing them to access a familiarization session and one of the three experimental sessions outlined below (see Figure 3).

1. *Cube movement*: Maneuver a gray cube into a specified distant target area.
2. *Cube colorization and movement*: Alter a cube's color before relocating it to the target area.
3. *Simple solar system*: Generate a stationary object (symbolizing the sun) and a moving object (symbolizing a planet), adhering to specific color instructions.

At the start of the study, participants were provided with a concise system overview. This was followed by a 5-minute familiarization session, allowing them to experience the system's accuracy and responsiveness. Subsequently, participants were requested to complete each experimental session in the provided order, involving various tasks related to object manipulation and creation within the virtual environment. We evaluated our system through a combination of methods:

- **Preliminary usability assessment**: We relied on the *task completion time* (TCT) and *error rate* to obtain a preliminary estimate of the usability of the system. While our sample size was too small to evaluate DreamCodeVR's usability fully, the results of these metrics could still be valuable in providing an initial indication of the weaknesses and strengths of the system.
- **Perceived workload**: We used the NASA-TLX [34] questionnaire to measure the participants' perceived workload during each session.
- **Structured interview**: We investigated the experience of users using immersive programming with a structured interview. The interview was carried out at the end of all three sessions. In the interview, we asked the participants to complete a questionnaire while reasoning aloud, motivating their answers. We collected their comments and reported their ratings for completeness (as the sample size was too small for these ratings to be statistically relevant) as shown in Figure 4. The structure and phrasing of the questionnaire items were based on existing surveys [67, 45], where a positive statement about the system is endorsed or refuted by users on a 7-point Likert scale, from 1 (strongly disagree) to 7 (strongly agree). The questionnaire included the following statements:

(Q1) I liked the possibilities given by the system.

(Q2) I felt immersed in the environment.

(Q3) It was simple attaching a behavior to an object.

(Q4) It was simple to create a system composed of multiple objects.

(Q5) Visual appearance properties are simpler to add than changing the kinematics of the object.

(Q6) The behaviors I added agreed with my description.

(Q7) The system was responsive, and the behavior was added in an acceptable time.

(Q8) I liked the overall experience.

5 Results and Analysis

In this section, we present a comprehensive analysis of the results obtained from the preliminary user study to evaluate the usability, perceived workload, and qualitative user experience assessment of DreamCodeVR.

5.1 Usability and Error Classification

TCT spanned between 70 and 499 seconds across all participants and tasks. Error occurrences varied from 0 to 6 across all contexts. Tasks 1 and 3 proved the most challenging, evidenced by the highest completion times and error rates. Intriguingly, despite Task 2 adding a color complexity layer to Task 1, it consistently resulted in lower TCTs and error rates for all participants. We classified observed errors into three categories:

Instruction Errors In this case, the instructions provided by the user were not accurate or correct for the program to accomplish the task, leading to either "no behavior" (6/37 of the generated instruction errors) or "wrong behavior" (7/37 the generated instruction errors), see Figure 5. Some examples of wrong behavior included the creation of an object in the wrong location and the movement of an object in an undesired pattern.

Non-Instruction Errors These errors (16/37) originated when users spoke while the system was waiting for instructions but without delivering instructions to the system (i.e., delivering a "non-instruction"). Non-instructions mostly included comments on the outcome of a previous interaction. For example, a user could comment *"wow, it really became red"* on changing the color of a sphere, while still pointing towards the object. This prompted the system to record the user's utterance and try to execute it. These errors gave origin to "no behavior". However, the occurrence of these errors was usually followed by increased frustration from the users. In fact, in these instances, users tended to deliver their next instruction while the system was still trying to compile the "non-instruction", giving the impression that the system was unresponsive.

System Errors Systems errors occurred when the system behavior did not reflect what was asked by the user. System errors accounted for 8 out of 37 errors. This includes three instances where the connection between the server and the services dropped. The number of system errors also includes five instances where the speech-to-text service could not correctly transcribe the users' instructions, likely due to some of the users' accents.

Item	Task 1	Task 2	Task 3
Completion time (s)	167.7 ± 78.1	79.5 ± 7.5	271.4 ± 134.5
Number of errors	2.1 ± 1.3	0.3 ± 0.4	2.9 ± 1.6

Table 2: Average TCT and Error number for all 3 tasks

5.2 Perceived Workload

Perceived workload ratings were generally low for the first two tasks and became higher for the third task. Specifically, in the third task, while the mental and physical load remained constant, the temporal demand, effort, performance, and frustration increased in value, although they remained of relatively low value (5.2).

Item	Task 1	Task 2	Task 3
Mental demand	13.4 ± 3.2	10.3 ± 2.0	15 ± 4.2
Physical demand	3.0 ± 1.2	2.3 ± 2.3	2.2 ± 3.1
Temporal demand	10.8 ± 4.3	7.5 ± 3.7	15 ± 4.4
Performance	11.2 ± 3.2	2.1 ± 5.0	14.4 ± 6.3
Effort	10.8 ± 4.5	3.1 ± 2.4	13.0 ± 4.6
Frustration	13.3 ± 4.0	6.5 ± 2.0	17.3 ± 5.2

Table 3: NASA-TLX results for all three tasks

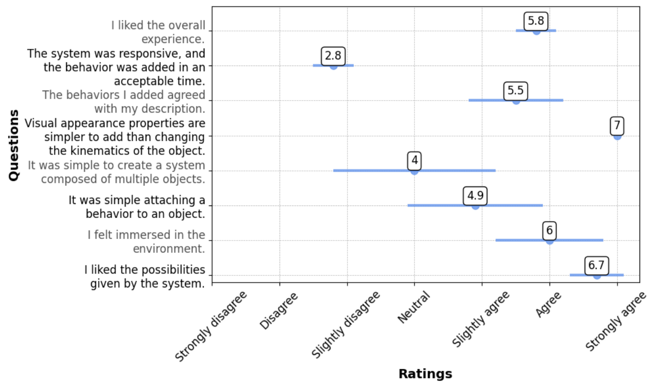


Figure 4: Average responses for the eight questions to drive the qualitative study. Although not statistically relevant due to the reduced sample size, these data provide an interesting summary of our users' impressions of the system.

5.3 Structured Interview Responses

We collected 216 comments across seven users and eight interview questions. Because of the structured nature of our interview, the focus of these comments was consistent across users. We categorized them into four categories: usability, potential use, immersion, and overall experience. Users commented positively about being immersed in the virtual environment and their overall experience (95% of comments on immersion and 83% on overall experience were positive). Most user comments referred to the system's potential use (20% of the overall comments) and usability (70% of the overall comments). Users largely expressed enthusiasm regarding the system's capabilities. One user commented *"This is fantastic. It would be amazing to create a game where you can conjure object* [to be used in the game] *by simply describing them."*. Another user pointed out how, despite its current limitation, the system had the potential to be used as a design tool: *"I feel like creating while in VR lets me estimate better what would be the experience of the users interacting with my design. I wonder whether there could be a hybrid solution where I can use common design tools as well as voice description to create the objects I want"*. Critiques primarily focused on interaction quality and the challenges faced in obtaining desired system outputs (mentioned by 7/7 users). For Q6, most users (6/7) concurred that DreamCodeVR outperformed their expectations, with one user noting that *it gets easier the more you use it*. All users found modifying existing objects (e.g., changing an object's color) easier than attaching behaviors (especially movement) or creating new objects. The complexity perceived by users amplified as more objects populated the scene, with 6/7 referring to the third task as the "hardest one." 5/7 expressed a desire for the system to generate more intricate objects. One user suggested, *"It would be cool to be able to create complex objects just with a sentence. Like: make a cow, and* [the system] *takes care of it for you"*.

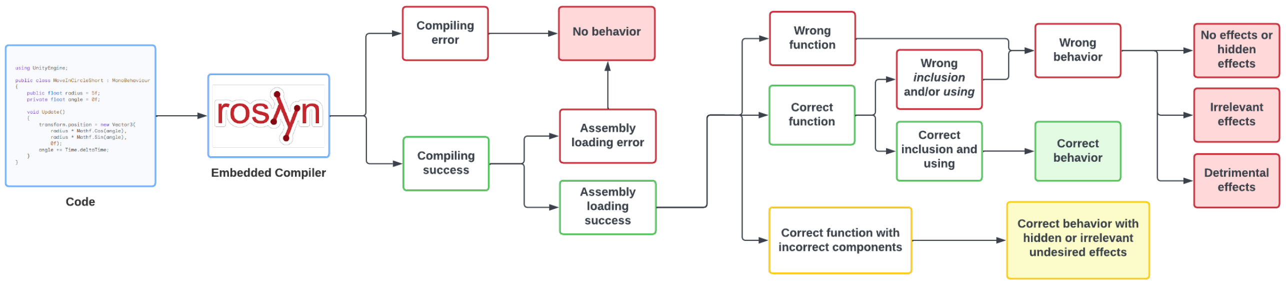
Regarding interaction, primary criticisms pertained to the system's object generation/modification latency. Beyond an evident desire for immediate request fulfilment (highlighted by 7/7 users), a common complaint was the disconnect between system operations and VR display: *"I wish I knew that my command has been recorded and that* [the system] *is working on it"*. One user also highlighted interaction deficits related to system trust, seeking a feedback mechanism to confirm object properties: *See, when I created a sphere, for example, I had no way to check whether it really had the radius of the length I asked for*.

6 Discussion

Our preliminary user study evaluated the potential of our system in an immersive scenario. Although the insights were derived from a small number of users, they provided notable preliminary insights on system learnability, user preferences, and the relationship between task complexity and user strategies. An intriguing observation arose from the transition from Task 1 (cube movement) to Task 2 (cube color change and movement). Despite Task 2's apparent complexity, participants exhibited lower error rates and shorter TCT than Task 1. This unexpected outcome suggests that users who mastered cube manipulation in Task 1 effortlessly applied this knowledge to Task 2. Additionally, modifying an object's appearance (color) did not introduce errors, implying that certain aspects of VR interaction, like altering visual properties, might be more intuitive than orchestrating object movement (Q5). This underscores the importance of foundational training before introducing complexity.

User performance across tasks revealed a direct link between task complexity, interaction strategies, and error occurrence. For instance, when creating a miniature solar system, users employed various techniques. Task complexity affected interaction depth and increased the likelihood of system errors, potentially leading to frustrating "no behavior" and "non-instruction" errors. This impacted Q6 and Q7, where longer prompts elevated the risk of misunderstandings by the system, resulting in higher error rates and delays due to the necessity to resubmit queries. These findings emphasize the importance of meticulously crafting interactions and prompts for specific tasks, ensuring a balance between the accuracy of the output and the system's response time. Participants' experiences underscored the need for refined instructions to minimize errors while maintaining efficiency. Combining multiple instructions into a single prompt elevated sentence complexity and error risk but reduced TCT when executed correctly. Addressing "non-instruction" errors might be achieved by implementing a controller trigger or similar mechanisms to filter out unintended speech interactions while awaiting instructions.

Users emphasized the need for enhanced interaction feedback as the display panel was only used for debugging purposes. They expressed a desire for notifications indicating when the system was generating a response and the ability to inspect objects in the VR environment, visualizing their attributes and relationships. These suggestions align with established principles of effective human-computer interaction, emphasizing the importance of clear feedback about system actions and status. Incorporating a user-facing display panel with comprehensive debugging feedback could significantly enhance user satisfaction, reduce frustration, and improve the transparency of AI-driven code generation, potentially increasing user confidence. Participants displayed genuine enthusiasm for DreamCodeVR's potential despite system limitations and user suggestions,



```
using UnityEngine;

public class MoveInCircleShort : MonoBehaviour
{
    public float radius = 5f;
    private float angle = 0f;

    void Update()
    {
        transform.position = new Vector3(
            radius * Mathf.Cos(angle),
            radius * Mathf.Sin(angle),
            0f);
        angle += Time.deltaTime;
    }
}
```

Figure 5: Diagram showing the possible outcomes of the different stages of a compiler in Unity environment. Red-filled boxes represent non-functional behaviors, green-filled boxes represent correct behaviors, and yellow-filled boxes represent correct behaviors with possible irrelevant or not perceived behaviors.

as suggested by the results of Q1 and Q8. They envisioned two primary use cases: (1) a design and creation tool and (2) an integrated game mechanism. Users valued the opportunity for VR-based content creation, facilitating more precise user experience estimations. However, they also highlighted the need for a hybrid approach that combines traditional design tools with voice-based object creation. It is crucial to recognize that while receiving encouraging feedback, the study mainly focused on identifying user interaction challenges and insights for future solution development and research directions. The potential for novelty effects to inflate usability perceptions among inexperienced users warrants caution in interpreting the results. Hence, although participants showed enthusiasm for DreamCodeVR, further investigation is necessary to fully understand its strengths and potential. Future research should engage a more extensive participant pool to increase the reliability of results.

6.1 System Outputs Classification

Driven by the user study analysis, this section discusses DreamCodeVR's systemic errors and deviations from expected outcomes in human-machine interaction. These errors are mostly brought on by two distinct systemic sources, which we explore extensively. The accuracy and consistency of the speech-to-text service play a vital role in the initial errors that arise from incorrect transcriptions. Such inaccuracies directly affect the accuracy of the LLM output. Specifically, if the transcription does not accurately reflect the user's spoken input, the resulting answer may be flawed. These flaws range from the total absence of code —mitigated by prompt-engineering techniques— to sound architecturally but functionally flawed code. System-based error effects are portrayed in Figure 5. We categorize the diverse behaviors exhibited by the system, contingent upon the output generated by the compiler. We list the following effects:

- **Absence of code**: Though our prompt engineering efforts have substantially minimized this error, it can manifest in component absence and provide a log detailing the encountered error when it does occur.
- **Code present but unable to compile**: An exceedingly infrequent error, resulting in the non-creation of a component and providing a pertinent log message.
- **Code compiled but with an assembly loading error**: Arising possibly from component name conflicts, our prompt engineering strategies typically preclude such issues. When they do occur, they are addressed through meticulous management.

Upon successful assembly loading, one of three scenarios may occur:

- **Incorrect function**: This scenario involves the system generating a flawed function, yielding results that can be imperceptible, irrelevant, or, in extreme cases, detrimental to the user. Imperceptible effects might encompass components creating non-visible logs or exhibiting no behavior. Irrelevant effects might involve minor alterations, like subtle color changes or minimal displacements of background objects. Detrimental effects have the potential to severely compromise user experience, spanning from object destruction to application crashes.
- **Correct function not used adequately**: Proper component attachment is vital. Even when the system generates the correct function, improper attachment can yield outcomes similar to those witnessed in the "Incorrect function" scenario.
- **Correct function with incorrect code sections**: The presence of the correct function with erroneous code sections can introduce irrelevant or hidden effects.
- **Correct function**: This scenario involves the correct function generation and usage, displaying the intended behavior.

This classification underscores the intricate and often convoluted path toward achieving accurate behavior in human-machine interaction. Nonetheless, these challenges are navigable, as the Unity Error Handling System dampens the impact of incorrect behavior, safeguarding both system stability and user experience. Moreover, comprehending and optimizing the LLMs' code performance represents an unresolved issue [51]. The efficacy of the generated code, especially concerning its memory and time complexity during execution, is subject to critical examination. Furthermore, it is worth noting that the efficiency of the LLM code can experience substantial variation, influenced by the LLM's stochasticity and the granularity of detail contained in the user's prompt.

6.2 Generalizability and Scalability

The system's architecture is modular, a design choice that facilitates the integration of future models and frameworks and enables straightforward customization for various contexts. The current design and functionality of the system are focused on the requirements for rapid development of basic object behavior in the context of game development, where the ability to attach dynamic objects or interactions is important. Expanding upon the current system's capabilities, particularly for the points mentioned in Section 7, this approach may eventually enable 3D artists to create dynamic environments without relying on developers or possessing extensive programming skills. While the system's primary use case and exploratory nature and the user study did not necessitate high accuracy, efficient code quality, or the integration of an existing code base, the system architecture was deliberately structured to support more extensions and more advanced language models in the future. These new models could enhance the quality and contextual relevance of the generated code. They may also facilitate more nuanced interactions, potentially enabling the development of fine-grained and complex behavior creation.

7 Limitations and Future Work

Human-AI collaborative programming has substantially influenced software development. DreamCodeVR represents an initial exploration into speech-based VR development, providing early insights into how LLMs may contribute to the evolution of VR application and content creation processes. However, there are areas for improvement and refinement to enhance its practical utility for developers.

Error Handling and User Feedback Utilizing LLMs to generate code introduces challenges, such as occasionally producing incorrect or incomplete code. Establishing a system to manage errors and furnish user feedback in an instructive and actionable manner is essential as has been shown in our preliminary user study. This includes developing a mechanism for categorizing errors, ranging from instruction conflicts to malfunctioning code. Moreover, the system should provide precise feedback, assisting users in rectifying errors or exploring alternative implementation methods. Tailoring feedback according to a user's expertise is also crucial, ensuring that guidance is both accessible and useful to all users.

Storing and Restoring the Edited Scene Exemplified by the frequency of systemic errors, introducing a feature that enables storing and restoring of the edited scene in DreamCodeVR is vital. This functionality would allow users to experiment with various code modifications without the risk of losing or damaging their existing work. Implementing a version control system could provide the mechanisms to empower users to manage their changes and revert to previous versions.

Security Issues Using LLMs for code generation brings potential security risks, such as generating malicious code. Thus, it is imperative to implement security measures within DreamCodeVR. Approaches might include the deployment of a sandbox to isolate assemblies, utilizing a code analysis tool to scrutinize generated code for potential security vulnerabilities, and employing an authentication mechanism to safeguard against unauthorized code injection.

Collaborative Speech Programming DreamCodeVR presents the compelling possibility of collaborative speech programming, where multiple users could concurrently contribute to a software project using voice commands. This concept, which can be viewed as an evolution of classical pair programming, has been shown to enhance code quality and diminish errors while positively influencing individual skills [33]. Presently, DreamCodeVR is constrained by limitations such as single-user code generation and binding the generated code to the speaking client's software environment. Achieving synchronization and coherence during code creation among spatially dispersed users within a VR programming session is a substantial hurdle, albeit one that might be overcome by utilizing template code from existing Ubiq examples tailored for social VR applications [58, 71].

Scenario Understanding for LLM Output Generation Another complication associated with employing LLMs for code generation is ensuring the produced code is appropriate for specific scenarios and contexts. DreamCodeVR could be enhanced by integrating an advanced scenario interpretation module. This module could evaluate the present scene and the user's objectives, subsequently generating code tailored to the situation. Especially within the VR data visualization domain, a nuanced understanding of how to optimally represent data graphically is essential. For these scenarios, the system could be designed to deduce optimal attributes for graphical representation from datasets, determine suitable chart type, layout, color scheme, and scale, and subsequently generate the requisite code to instantiate the graphic elements.

Dynamic Prompt Engineering Enhancing the quality of code generated by DreamCodeVR may also be achieved through dynamic prompt engineering, wherein the prompt provided to the LLM is modified in alignment with the scene's evolution. For instance, if the user defines a variable utilized across numerous components, the prompt could be adjusted to incorporate this information. Future endeavors might explore enabling the system to adapt prompt engineering as scenes evolve autonomously, minimizing manual intervention and aligning with the ethos of "learning to learn" whereby the system augments its performance through adaptation.

Accessibility As a voice-based programming interface, DreamCodeVR has potential to become an effective accessibility tool. For example, specific populations, such as individuals with impaired fine motor skills, could use it as an alternative to conventional speech-to-text software. With adaptations to the current interaction modality, which primarily relies on manual object selection, the system can be tailored to suit individuals encountering difficulties with precise hand movements. By incorporating voice commands as an alternative means for object selection or area interaction within the virtual scenario, DreamCodeVR could significantly enhance inclusivity.

8 Conclusion

DreamCodeVR represents a novel system that harnesses the capabilities of AI, specifically LLMs, to facilitate the generation of dynamic behaviors, properties, and mechanics within VR environments. Building on recent works that utilize LLMs for writing application code, DreamCodeVR makes VR application design accessible by enabling individuals without extensive coding expertise to actively participate in crafting immersive VR experiences. Our work demonstrates the potential of LLMs to generate context-aware behaviors in VR, accomplishing real-time interaction through speech inputs. The modular architecture, built on Ubiq and Ubiq-Genie [26, 57], combines speech transcription, prompt engineering, code generation, and just-in-time compilation to infuse AI-authored behaviors into Unity scenes. While initial findings suggest its potential in democratizing VR content creation, especially for non-expert users, this research is a preliminary step that underscores the need for further investigation to realize and validate DreamCodeVR's capabilities and impact fully. In this context, comparing DreamCodeVR's approach with recently emerging systems [21], which also utilize LLMs in behavior definition and world building, could provide a broader perspective on the opportunities and challenges of using LLMs for VR development. While our work has explored interesting new directions in VR content creation and rapid prototyping, numerous challenges persist. Issues related to security, reliability, and bias in LLM-generated code necessitate continued research. Furthermore, promising paths forward involve exploring robust error-handling mechanisms and refining prompt engineering strategies. Moreover, usage of our system could extend beyond the individual user, suggesting potential for collaborative endeavors in social VR programming – an intriguing direction for future VR research.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 Research and Innovation program under grant agreement No 739578.

References

[1] Gpt4all models performances benchmark. `https://gpt4all.io/index.html`. 4

[2] QuakeC. `https://en.wikipedia.org/wiki/QuakeC`, Mar. 2023. 2

[3] Y. Anand, Z. Nussbaum, B. Duderstadt, B. Schmidt, and A. Mulyar. Gpt4all: Training an assistant-style chatbot with large scale data distillation from gpt-3.5-turbo. `https://github.com/nomic-ai/gpt4all`, 2023. 4

[4] S. A. Antu, H. Chen, and C. K. Richards. Using llm (large language model) to improve efficiency in literature review for undergraduate research. 2023. 1

[5] Ö. Aydın and E. Karaarslan. Is chatgpt leading generative ai? what is beyond expectations? *What is beyond expectations*, 2023. 1

[6] C. Baber, B. Mellor, R. Graham, J. Noyes, and C. Tunley. Workload and the use of automatic speech recognition: The effects of time and resource demands. *Speech Communication*, 20(1):37–53, 1996. doi: 10.1016/S0167-6393(96)00043-X 3

[7] R. A. Bartle. From MUDs to MMORPGs: The History of Virtual Worlds. In J. Hunsinger, L. Klastrup, and M. Allen, eds., *International Handbook of Internet Research*, pp. 23–39. Springer Netherlands, Dordrecht, 2010. doi: 10.1007/978-1-4020-9789-8_2 2

[8] J. Behr, U. Bockholt, and D. Fellner. Instantreality — A Framework for Industrial Augmented and Virtual Reality Applications. In D. Ma, X. Fan, J. Gausemeier, and M. Grafe, eds., *Virtual Reality & Augmented Reality in Industry*, pp. 91–99. Springer, Berlin, Heidelberg, 2011. doi: 10.1007/978-3-642-17376-9_5 2

[9] M. Bevan. The SCUMM Diary: Stories behind one of the greatest game engines ever. `https://www.gamedeveloper.com/`, July 2013. 2

[10] N. Bevan, C. Barnum, G. Cockton, J. Nielsen, J. Spool, and D. Wixon. The" magic number 5" is it enough for web testing? In *CHI'03 extended abstracts on Human factors in computing systems*, pp. 698–699, 2003. 6

[11] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira. VR Juggler: a virtual platform for virtual reality application development. In *Proceedings IEEE Virtual Reality 2001*, pp. 89–96, Mar. 2001. doi: 10.1109/VR.2001.913774 2

[12] R. Bovo, D. Giunchi, L. Sidenmark, J. Newn, H. Gellersen, E. Costanza, and T. Heinis. Speech-augmented cone-of-vision for exploratory data analysis. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23. ACM, New York, NY, USA, 2023. doi: 10.1145/3544548.3581283 3

[13] C. Brown. The Circuits Handbook. `https://blog.recroom.com/posts/2021/5/03/the-circuits-handbook`, May 2021. 2

[14] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 1

[15] M. Cavazza, F. Charles, and S. J. Mead. Characters in search of an author: Ai-based virtual storytelling. In *International Conference on Virtual Storytelling*, pp. 145–154. Springer, 2001. 1

[16] V. Chamola, G. Bansal, T. K. Das, V. Hassija, N. S. S. Reddy, J. Wang, S. Zeadally, A. Hussain, F. R. Yu, M. Guizani, et al. Beyond reality: The pivotal role of generative ai in the metaverse. *arXiv preprint arXiv:2308.06272*, 2023. 1

[17] M. Chen, M. Peljhan, and M. Sra. Entanglevr: A visual programming interface for virtual reality interactive scene generation. In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*, VRST '21. Association for Computing Machinery, New York, NY, USA, 2021. doi: 10.1145/3489849.3489872 1

[18] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. 2

[19] R. J. Cox and P. S. Crowther. A Review of Linden Scripting Language and Its Role in Second Life. In M. Purvis and B. T. R. Savarimuthu, eds., *Computer-Mediated Social Networking*, Lecture Notes in Computer Science, pp. 35–47. Springer, Berlin, Heidelberg, 2009. doi: 10.1007/978-3-642-02276-0_5 2

[20] A. B. Craig, W. R. Sherman, and J. D. Will. *Developing virtual reality applications: Foundations of effective design*. Morgan Kaufmann, 2009. 2

[21] F. De La Torre, C. M. Fang, H. Huang, A. Banburski-Fahey, J. A. Fernandez, and J. Lanier. Llmr: Real-time prompting of interactive worlds using large language models. *arXiv preprint arXiv:2309.12276*, 2023. 3, 9

[22] U. Documentation. Order of execution for event functions. `https://docs.unity3d.com/Manual/ExecutionOrder.html`. 4

[23] Dotnet. Roslyn c# compiler. `https://github.com/dotnet/roslyn`. The .NET Compiler Platform. 2

[24] A. Ebrahimi, T. West, M. Schoen, and D. Urquidi. Unity: editorVR. In *ACM SIGGRAPH 2017 Real Time Live!*, SIGGRAPH '17, p. 27. Association for Computing Machinery, New York, NY, USA, July 2017. doi: 10.1145/3098333.3098918 2

[25] N. A. Ernst and G. Bavota. Ai-driven development is here: Should you worry? *IEEE Software*, 39(2):106–110, 2022. doi: 10.1109/MS.2021.3133805 2

[26] S. J. Friston, B. J. Congdon, D. Swapp, L. Izzouzi, K. Brandstätter, D. Archer, O. Olkkonen, F. J. Thiel, and A. Steed. Ubiq: A system to build flexible social virtual reality experiences. In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*, VRST '21. Association for Computing Machinery, New York, NY, USA, 2021. doi: 10.1145/3489849.3489871 1, 3, 9

[27] E. Frécon and M. Stenius. DIVE: a scaleable network architecture for distributed virtual environments. *Distributed Systems Engineering*, 5(3):91, Sept. 1998. doi: 10.1088/0967-1846/5/3/002 2

[28] A. Gatti. Virtual environments via natural language agents. In *European Conference on Multi-Agent Systems*, pp. 486–492. Springer, 2023. 1

[29] D. Giunchi, A. Sztrajman, S. James, and A. Steed. Mixing modalities of 3d sketching and speech for interactive model retrieval in virtual reality. In *ACM International Conference on Interactive Media Experiences*, IMX '21, p. 144–155. ACM, New York, NY, USA, 2021. doi: 10.1145/3452918.3458806 3

[30] J. R. Glass, T. J. Hazen, D. S. Cyphers, I. Malioutov, D. Huynh, and R. Barzilay. Recent progress in the MIT spoken lecture processing project. In *INTERSPEECH 2007, 8th Annual Conference of the International Speech Communication Association, Antwerp, Belgium, August 27-31, 2007*, pp. 2553–2556, 2007. 3

[31] M. Goto, J. Ogata, and K. Eto. Podcastle: a web 2.0 approach to speech recognition research. In *INTERSPEECH 2007, 8th Annual Conference of the International Speech Communication Association, Antwerp, Belgium, August 27-31, 2007*, pp. 2397–2400. Curran Associates, Inc., Antwerp, Belgium, 2007. 3

[32] S. Greengard. Ai rewrites coding. *Communications of the ACM*, 66(4):12–14, 2023. 2

[33] J. E. Hannay, T. Dybå, E. Arisholm, and D. I. Sjøberg. The effectiveness of pair programming: A meta-analysis. *Information and software technology*, 51(7):1110–1122, 2009. 9

[34] S. G. Hart and L. E. Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, vol. 52, pp. 139–183. Elsevier, 1988. 6

[35] M. Hirzel. Low-code programming models. *Communications of the ACM*, 66(10):76–85, 2023. 1

[36] A. Hook. Psychlorama: An immersive dome experience. In *Psychlorama: An Immersive Dome Experience*, 2021. 1

[37] B. Interactive. Chat gpt for games - ai integration. https://assetstore.unity.com/packages/tools/ai-ml-integration/chat-gpt-for-games-ai-integration-248841. 3

[38] T. Interactive. Roslyn c# - runtime compiler. `https://assetstore.unity.com/packages/tools/integration/roslyn-c-runtime-compiler-142753`. Unity Asset for Roslyn C# Compiler. 4

[39] Extensible 3D (X3D) — Part 1: Architecture and base components. Standard ISO/IEC 19775-1, International Organization for Standardization / International Electrotechnical Commission, Geneva, CH, Aug. 2023. Status: Under development. 2

[40] N. Jain, S. Vaidyanath, A. S. Iyer, N. Natarajan, S. Parthasarathy, S. Rajamani, and R. Sharma. Jigsaw: Large language models meet program synthesis. *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pp. 1219–1231, 2021. doi: 10.1145/3510003.3510203 2

[41] J. M. Janas. The semantics-based natural language interface to relational databases. In L. Bolc and M. Jarke, eds., *Cooperative Interfaces to Information Systems*, pp. 143–188. Springer Berlin Heidelberg, Berlin, Heidelberg, 1986. doi: 10.1007/978-3-642-82815-7_6 3

[42] J. Kelso, L. Arsenault, S. Satterfield, and R. Kriz. Diverse: a framework for building extensible and reconfigurable device independent virtual environments. In *Virtual Reality, 2002. Proceedings. IEEE*, pp. 183–190, 2002. doi: 10.1109/VR.2002.996521 2

[43] M. Lebens, R. J. Finnegan, S. C. Sorsen, and J. Shah. Rise of the citizen developer. *Muma Business Review*, 5:101–111, 2022. 2

[44] M. Lee and M. Billinghurst. A wizard of oz study for an ar multimodal interface. In *Proceedings of the 10th International Conference on Multimodal Interfaces*, ICMI '08, p. 249–256. ACM, New York, NY, USA, 2008. doi: 10.1145/1452392.1452444 3

[45] J. R. Lewis. Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, 7(1):57–78, 1995. 6

[46] F. Li and H. V. Jagadish. Nalir: An interactive natural language interface for querying relational databases. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pp. 709–712. ACM, New York, NY, USA, 2014. doi: 10.1145/2588555.2594519 3

[47] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. 2

[48] J. Liu, C. S. Xia, Y. Wang, and L. Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *arXiv preprint arXiv:2305.01210*, 2023. 2

[49] L. LLC. Spark ai - gpt dialogue & tools. https://assetstore.unity.com/packages/tools/ai-ml-integration/spark-ai-gpt-dialogue-tools-144575. 3

[50] B. MacIntyre, M. Gandy, S. Dow, and J. D. Bolter. DART: a toolkit for rapid design exploration of augmented reality experiences. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, UIST '04, pp. 197–206. Association for Computing Machinery, New York, NY, USA, Oct. 2004. doi: 10.1145/1029632.1029669 2

[51] A. Madaan, A. Shypula, U. Alon, M. Hashemi, P. Ranganathan, Y. Yang, G. Neubig, and A. Yazdanbakhsh. Learning performance-improving code edits. *arXiv preprint arXiv:2302.07867*, 2023. 8

[52] D. Martin, A. Serrano, A. W. Bergman, G. Wetzstein, and B. Masia. Scangan360: A generative model of realistic scanpaths for 360 images. *IEEE Transactions on Visualization and Computer Graphics*, 28(5):2003–2013, 2022. 1

[53] S. McGlashan and T. Axling. A speech interface to virtual environments, 1996. 3

[54] J. Muller, C. Krapichler, L. S. Nguyen, K. Hans Englmeier, and M. Lang. Speech interaction in virtual reality. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*, vol. 6, pp. 3757–3760 vol.6, 1998. doi: 10.1109/ICASSP.1998.679701 3

[55] J. T. Murray. RealityFlow: Open-Source Multi-User Immersive Authoring. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 65–68, Mar. 2022. doi: 10.1109/VRW55335.2022.00024 2

[56] J. Noyes. Speech technology in the future. In *Interactive speech technology: Human factors issues in the application of speech input/output to computers*, pp. 189–208. Taylor & Francis London, 1993. 3

[57] N. Numan, D. Giunchi, B. Congdon, and A. Steed. 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)*, pp. 497–501, 2023. doi: 10.1109/VRW58643.2023.00108 1, 3, 9

[58] N. Numan, Z. Lu, B. Congdon, D. Giunchi, A. Rotsidis, A. Lernis, K. Larmos, T. Kourra, P. Charalambous, Y. Chrysanthou, et al. Towards outdoor collaborative mixed reality: Lessons learnt from a prototype system. In *2023 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 113–118. IEEE, 2023. 9

[59] T. R. D. Oliveira, B. B. Rodrigues, M. M. D. Silva, R. A. N. Spinassé, G. G. Ludke, M. R. S. Gaudio, G. I. R. Gomes, L. G. Cotini, D. D. S. Vargens, M. Q. Schimidt, R. V. Andreão, and M. Mestria. Virtual reality solutions employing artificial intelligence methods: A systematic literature review. *ACM Computing Surveys*, 55:1 – 29, 2022. doi: 10.1145/3565020 1

[60] M. Oltrogge, E. Derr, C. Stransky, Y. Acar, S. Fahl, C. Rossow, G. Pellegrino, S. Bugiel, and M. Backes. The rise of the citizen developer: Assessing the security impact of online app generators. In *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 634–647. IEEE, 2018. 2

[61] C. Popp and D. T. Murphy. Creating audio object-focused acoustic environments for room-scale virtual reality. *Applied Sciences*, 12(14):7306, 2022. 1

[62] J. Ratican, J. Hutson, and A. Wright. A proposed meta-reality immersive development pipeline: Generative ai models and extended reality (xr) content for the metaverse. *Journal of Intelligent Learning Systems and Applications*, 15, 2023. 1

[63] J. Roberts, A. Banburski-Fahey, and J. Lanier. Steps towards prompt-based creation of virtual worlds. *arXiv preprint arXiv:2211.05875*, 2022. 1, 3

[64] J. Roberts, A. Banburski-Fahey, and J. Lanier. Surreal VR Pong: LLM approach to Game Design. In *36th Conference on Neural Information Processing Systems (NeurIPS 2022)*, Dec. 2022. 1

[65] M. Romero, B. Sewell, and L. Cataldi. *Blueprints Visual Scripting for Unreal Engine 5: Unleash the true power of Blueprints to create impressive games and applications in UE5*. Packt Publishing Ltd, 2022. 1

[66] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023. 1, 2

[67] J. G. Schoeberlein and Y. Wang. Usability evaluation of an accessible collaborative writing prototype for blind users. *Journal of Usability Studies*, 10(1), 2014. 6

[68] A. Siyaev and G.-S. Jo. Towards aircraft maintenance metaverse using speech interactions with virtual objects in mixed reality. *Sensors*, 21(6), 2021. doi: 10.3390/s21062066 3

[69] A. W. Stedmon. *Putting Speech in, taking speech out: human factors in the use of speech interfaces*. PhD thesis, University of Nottingham, 2005. 3

[70] A. W. Stedmon, H. Patel, S. C. Sharples, and J. R. Wilson. Developing speech input for virtual reality applications: A reality based interaction approach. *International Journal of Human-Computer Studies*, 69(1):3–8, 2011. doi: 10.1016/j.ijhcs.2010.09.002 3

[71] A. Steed, D. Archer, K. Brandstätter, B. J. Congdon, S. Friston, P. Ganapathi, D. Giunchi, L. Izzouzi, G. W. W. Park, D. Swapp, et al. Lessons learnt running distributed and remote mixed reality experiments. *Frontiers in Computer Science*, 4:966319, 2023. 9

[72] R. M. Taylor, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser. VRPN: a device-independent, network-transparent VR peripheral system. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '01, pp. 55–61. Association for Computing Machinery, New York, NY, USA, Nov. 2001. doi: 10.1145/505008.505019 2

[73] N. ter Heijden and W.-P. Brinkman. Design and evaluation of a virtual reality exposure therapy system with automatic free speech interaction. *Journal of CyberTherapy and Rehabilitation*, 4(1):41–55, 2011. 3

[74] J. van Stegeren and J. Myśliwiec. Fine-tuning gpt-2 on annotated rpg quests for npc dialogue generation. In *Proceedings of the 16th International Conference on the Foundations of Digital Games*, pp. 1–8, 2021. 3

[75] R. Volum, S. Rao, M. Xu, G. A. DesGarennes, C. Brockett, B. V. Durme, O. Deng, A. Malhotra, and B. Dolan. Craft an Iron Sword: Dynamically Generating Interactive Game Characters by Prompting Large Language Models Tuned on Code. In *The Third Wordplay: When Language Meets Games Workshop*, July 2022. 1, 3

[76] VRChat Inc. Udon | VRChat Creation. `https://creators.vrchat.com/worlds/udon/`, Aug. 2023. 2

[77] W3C. WebXR Device API. `https://www.w3.org/TR/webxr/`, 2023. 2

[78] World Viz. Vizard virtual reality software toolkit. `https://www.worldviz.com/vizard-virtual-reality-software`, 2023. 2

[79] L. Zhou, M. Shaikh, and D. Zhang. Natural language interface to mobile devices. In Z. Shi and Q. He, eds., *Intelligent Information Processing II*, pp. 283–286. Springer US, Boston, MA, 2005. doi: 10.1007/0-387-23152-8_37 3