Laboratório VISGRAF

Instituto de Matemática Pura e Aplicada

Gen.AI Playground: Multicast-Multimodal AI Platform

Matteo Moriconi, Luiz Velho

Technical Report TR-25-02 Relatório Técnico

May - 2025 - Maio

Gen.AI Playground:
Multicast-Multimodal AI Platform

Authors: Matteo Moriconi (VFXRio), Luiz Velho (IMPA)

---

1. Introduction

The Multicast-Multimodal AI Platform, herein referred to as the "Gen.AI Playground," is an advanced, experimental environment built to explore the frontiers of artificial intelligence, distributed computing, and multimodal interfaces. It enables the creation and real-time deployment of rich, interactive media experiences where users and virtual agents engage in dynamic exchanges. By combining narrative-driven design with real-time systems and AI character embodiment, this platform empowers creators to design performances, simulations, and storytelling applications that blur the boundaries between the physical and digital worlds.

This system supports ideation, prototyping, and the live execution of AI-enhanced experiences. The goal is to fuse intelligent behavior, expressive content, and audience engagement into a seamless digital experience that adapts to various artistic, educational, and entertainment contexts.

---

2. Core Components and Architecture

2.1. Distributed Space and Modalities

- **Space:**
  - The platform's infrastructure is fully distributed, using Internet protocols and web-based technologies to interconnect participants and devices across global locations. These elements form a virtual mesh that supports real-time collaboration and AI processing.
  - Embracing the principles of New Media and Expanded Reality (XR), the platform supports immersive experiences in VR, AR, and MR. Content delivery is not confined to screens but integrated into physical installations and responsive environments.
- **Modalities:**
  - Information flows through multimodal channels, including textual interactions, audio responses, visual imagery, and structured data such as sensor readings and metadata.
  - Each node in the system can communicate across multiple formats, adapting to user context, device capability, and interaction goals. (See Fig. 1)

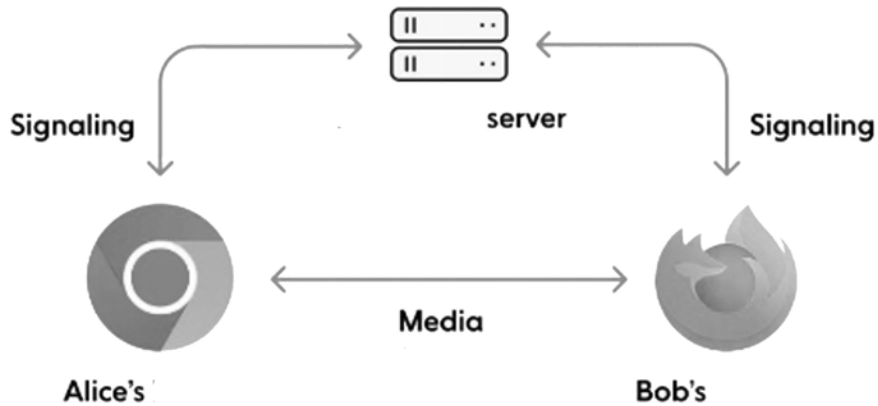

Figure 1. *Platform Overview – Distributed Multicast-Multimodal Environment*

2.2. Experience Modes

- **Conversational Interfaces (Chat):**
    - Users can engage in open-ended conversations with AI agents that simulate personality, emotion, and memory. These interactions are enhanced by context-aware mechanisms and learning-based responses.
- **Live Media Performances:**
    - Artists, performers, and AI systems co-create events where media (video, audio, animation) is dynamically generated and manipulated in response to audience feedback and narrative triggers.
- **Immersive Storytelling & Experiences:**
    - Extended and episodic interactions involving directors, actors, AI characters, and participants allow for continuously evolving experiences. These interactions are managed through narrative engines and scene graphs that adapt to input in real time.

2.3. Conceptual Framework

- **Agents:** Entities with autonomy that perform actions and maintain state.
- **Narratives:** Structured, rule-based or generative story flows that determine user progression.
- **Experiences:** The container for multimodal interaction instances involving agents and users.
- **Plans:** Sequences of intended actions guided by narrative logic or decision trees.
- **Memory:** Short- and long-term storage for user context, agent state, and shared knowledge.
- **Embodiment:** The visual, auditory, and behavioral manifestation of agents in 2D, 3D, or hybrid environments.

3. Computational Model

3.1. Framework Definitions

- **Nodes:**
  - **Entity Nodes** represent active elements in the experience—AI agents, avatars, visual displays, and input devices.
  - **Service Nodes** provide utility functions like AI model inference, web proxies, external APIs, or data formatting.
  - **User Nodes** are client interfaces where participants, directors, or actors interact with the system.
- **Edges:**
  - Define the type and direction of multimodal communication: text messages, media streams, sensor input/output, and control signals.
- **Graph Type:**
  - The platform utilizes a distributed, reactive computational graph, where connectivity is fluid and adapts based on interaction states, system load, and narrative requirements. (See Fig 2.)

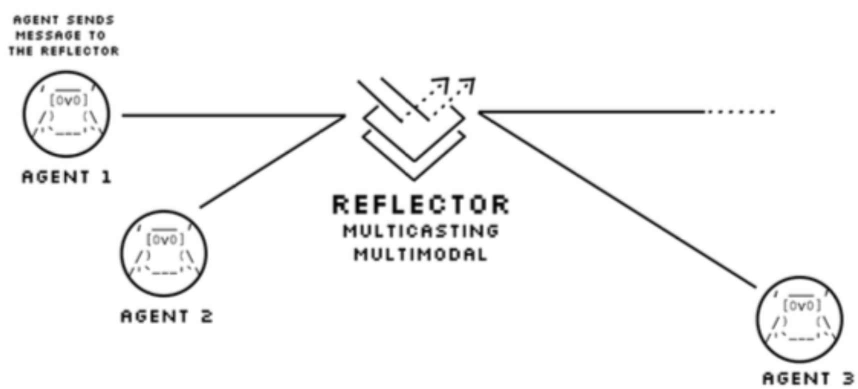

Figure 2. *Reflector-based architecture for multicast multimodal communication*

3.2. Dynamics and Communication

- **Endpoints:**
  - Clients act as emitters (sources of interaction), receivers (consumers), or hybrid nodes.
  - The server functions as a central relay and logic processor that routes, augments, or interprets messages.
- **Topology Control:**
  - The server dynamically configures graph topology based on the narrative script or agent activity. For instance, nodes may be connected or isolated depending on a user's progression through the experience.

- **Message Protocol:**
  - Messages carry timestamps and IDs for sequencing and accountability. All communication follows structured formats with metadata for tracing and synchronization. Technologies like WebSockets and WebRTC ensure low-latency and resilient message transport.

---

4. Platform Functionality: The Playground

4.1. Server Functions

- Maintains a persistent state of the entire environment, tracking active users, ongoing sessions, and live agent deployments.
- Handles synchronization, registration and deregistration of clients, issuing unique tokens and managing command execution.
- Logs interactions for replay and machine learning feedback

4.2. Client Functions

- Connects to chat and performance sessions with full multimodal capabilities. Clients initiate and receive communication through a command interface. They may simulate characters or interact as users.
- Users can switch personas, send private messages, and view or influence agent behavior.
- The chat interface is highly configurable, enabling private messaging, and multi-user threads.

4.3. Commands

- **Client-Side:**
  - Commands allow users to activate AI agents, switch personas, and define communication rules:
    - :start_gpt, :stop_gpt — launches or halts AI responses
    - :start_autoreply_gpt, :stop_autoreply_gpt — enables looping replies
    - :persona — defines or changes character traits
    - :character — assigns visual/behavioral traits
    - :respond_* — controls reply logic to individuals or groups
    - :turns, :knw — access memory/history or knowledge modules
    - :pvt [recipient] — private messaging
- **Server-Side:**
  - Scene-level control tools for synchronizing context:
    - :scene — defines the scenario
    - :location — spatial setting
    - :datetime — temporal alignment of narrative or session

5. Interaction Model

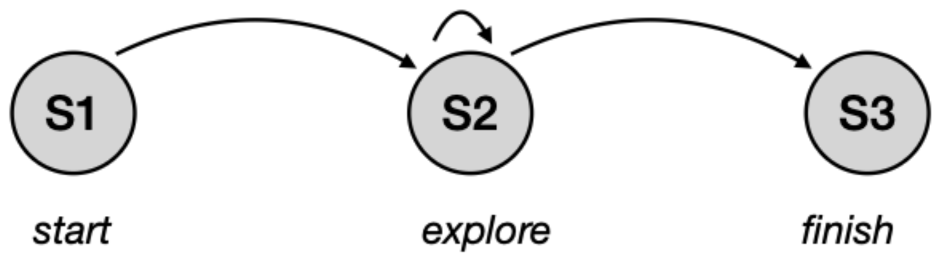

Figure 3. *Finite state machine guiding narrative flow*

5.1. FSM Structure

Each interactive session is guided by a Finite State Machine (FSM), with each state representing a phase in the user or narrative journey (e.g., onboarding, exploration, climax, resolution). (See Fig. 3)

Transition probabilities can be fixed or learned through user interaction logs and feedback [5].

- Interactions are modeled using Finite State Machines (FSMs), where each node represents a state (e.g., onboarding, exploration, decision point, climax, resolution)
- Transitions are triggered by events, user actions, or probabilistic models informed by interaction history
- This approach allows experiences to be modular, reusable, and reactive

5.2. Node Mapping

- Each session consists of mapped nodes with specific functions:
  - Terminals for users, agents, and observers
  - Routing nodes for media
  - Logging/debugging interfaces for directors and developers
- Example:
  - **User 1:** e.g., a viewer sending text
  - **User 2:** e.g., a performer receiving prompts
  - **Server:** central logic and relay for all messaging

6. Agents and Avatars

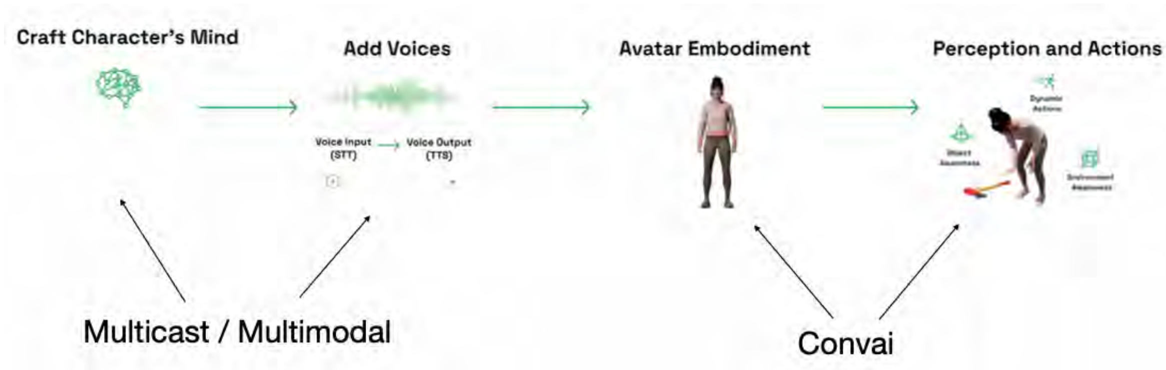

Figure 4. *Pipeline for crafting AI characters: from mind modeling to avatar embodiment*

6.1. Digital Human Architecture

- Characters are defined by modular anatomy including appearance, behavior, dialogue management, and contextual memory: (See Fig. 4)
    - **Physical Layer:** avatar models, facial rigging, audio sync
    - **Cognitive Layer:** LLM or rule-based dialogue manager
    - **Emotional Layer:** affective state control, expressive gestures
    - **Memory Layer:** long-term memory, character knowledge base

6.2. Convai Integration

- Use of the Avatar Studio and Convai APIs to build embodied AI agents capable of real-time interaction and expression. These agents respond naturally to inputs, maintain context, and act within a spatial environment [6].
- Integration with Unreal Engine allows spatialized embodiment in XR [7].

---

7. Conclusion

The Gen.AI Playground is a modular and expressive infrastructure for AI-powered digital experiences. It unifies storytelling, performance, and multimodal computing in a robust architecture that can adapt to a wide range of applications—from virtual exhibitions to conversational games and live performances. By embracing distributed intelligence, real-time communication, and emotional AI, it enables rich, interactive narratives that are both technically sophisticated and creatively compelling. The platform is implements the technological infrastructure that powers AI Story-Worlds [8].

8. Case Study: Celestia Borealis

Figure 5. *Celestia Borealis*

8.1 Overview

Celestia Borealis is a flagship demonstration of the Multicast-Multimodal AI Platform, designed as a real-time conversational agent capable of interacting with both humans and non-human entities. She represents a tangible synthesis of the platform's capabilities in distributed communication, multimodal interaction, character embodiment, and emotional AI.

Described as a "groundbreaking multicasting multimodal agent," Celestia is deployable as either an advanced AI or a lifelike hologram, engaging audiences through expressive behaviors, natural dialogue, and reactive presence. (See Fig 6.)

8.2 Architecture and Deployment

- **Celestia's Installation Context:**
  - **User Interaction Space:** VISGRAF Lab (physical or XR)
  - **Embodiment Engine:** Unreal Engine + Convai Avatar Studio
  - **Receiver:** Local interface or web-connected client
- **Backstage System:**
  - **Server Management:** VISGRAF Data Center
  - **Emissive Actor:** A character-driven AI backend (e.g., "Matteo")
  - **VFX Contribution:** Collaboration with VFX Roma

[User] ⇄ [Celestia Avatar (Unreal/Convai)] ⇄ [VISGRAF Server] ⇄ [Matteo AI Core]

Figure 6. Celestia Borealis Node Setup - *Deployment Diagram*

8.3 Intelligence and Emotion Modeling

Mind Architecture:

- **Personality:** Composed of psychological trait modules defining behavioral consistency
- **Knowledge Base:** Internet-derived world model fused with memory and dialogue capabilities

Emotional Core:

- **Basic Emotions:** Happy, Angry, Surprise, Serious (See Fig 7.)
- **State Transitions:** Emotion modulation depending on user inputs, narrative progression, and internal triggers

Figure 7: *Emotional Modulation Spectrum*
[Happy] ⇄ [Happier] ⇄ [Serious] ⇄ [Surprise]

8.4 Retrofit and Adaptability

Celestia supports continuous updates in style and behavior. Her design evolution includes:

- **Visual Retrofits:** Stylistic changes to maintain engagement
- **Behavioral Additions:** Movement scripting, expressive animations (e.g., "Shake it, Baby!")

These upgrades are pushed through modular content updates via the platform's backend, demonstrating its flexibility and creative extensibility.

8.5 Significance

Celestia Borealis exemplifies the integration of narrative intelligence, modular AI character building, and distributed deployment. As a live demonstrator, she validates the Gen.AI Playground's use cases in performance arts, XR storytelling, and adaptive digital interfaces.

8.6 Installation

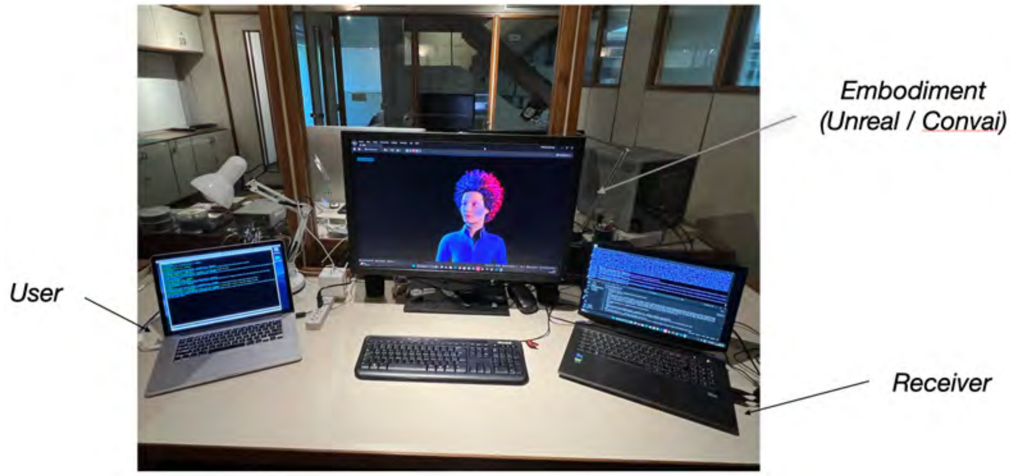

Figure 8: *VISGRAF Lab Installation*

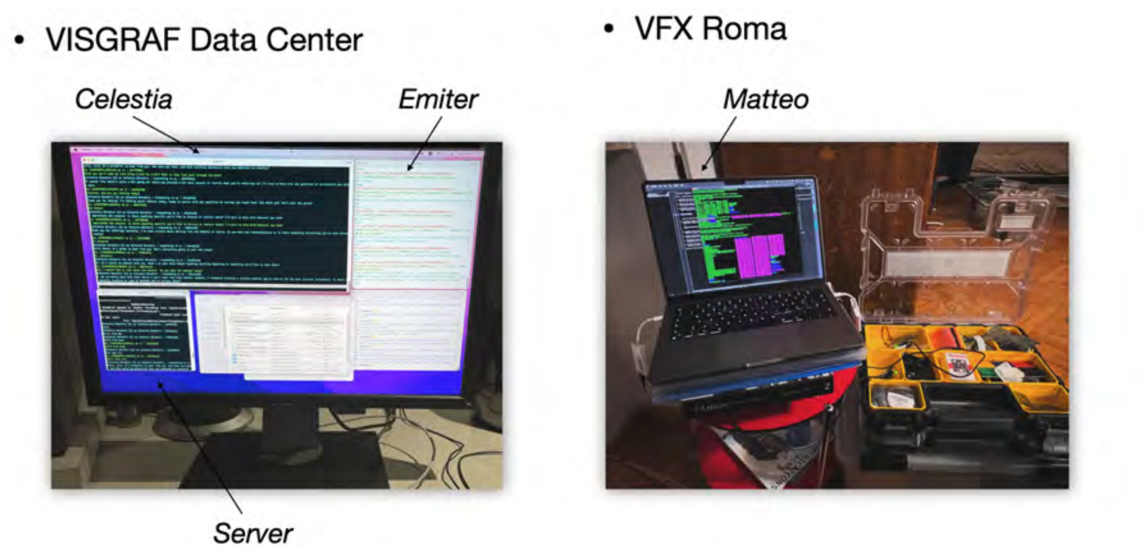

Figure 9: *VISGRAF Data Center / VFX Rome*

Through Celestia, the platform showcases:

- Seamless interaction loops between embodied agents and users
- Real-time emotional engagement
- Distributed rendering and AI processing across physical and virtual domains

References

[1] M. Weiser, “The Computer for the 21st Century,” *Scientific American*, vol. 265, no. 3, 1991.

[2] O. Vinyals et al., “A Neural Conversational Model,” *ICML Deep Learning Workshop*, 2015.

[3] M. Mateas and A. Stern, “Structuring Content in the Façade Interactive Drama Architecture,” *AI and Interactive Digital Entertainment*, 2005.

[4] R. Shea and J. Liu, “Performance Modeling and Evaluation of Real-Time Multimedia Services,” *IEEE Transactions on Multimedia*, vol. 9, no. 3, 2007.

[5] C. Rich, B. Ponsler, and C. L. Sidner, “Recognizing Engagement in Human-Robot Interaction,” *HRI*, 2010.

[6] Convai. (n.d.). *Conversational AI for Virtual Worlds*. Retrieved May 17, 2025, from https://www.convai.com/

[7] Epic Games. (n.d.). *Unreal Engine: The most powerful real-time 3D creation tool*. Retrieved May 17, 2025, from https://www.unrealengine.com/

[8] L. Velho. and M. Moriconi, *AI Story Worlds: A Multimodal Platform for Creative Narrative Co-Creation*, Technical Report TR-01-2025, VISGRAF Lab - IMPA, 2025.

Appendix:
Multicast-Multimodal Story Worlds Framework Code Description

Version: v1.1

---

1. Outline

The **Multicast-Multimodal Story Worlds** framework is a Python-based communication system designed for building distributed, interactive, and multimodal narrative environments. It supports synchronous messaging among clients orchestrated by a central server, forming the backbone of a **real-time narrative platform**. This platform can be extended to support diverse storytelling agents—such as avatars, LLMs, sound engines, and visualization systems—each participating as a client in a shared story world.

---

2. System Architecture

The system implements a modular **client-server architecture** using low-level TCP socket programming. Each client communicates with a single server instance which manages incoming messages and rebroadcasts them to all connected clients.

Architecture Diagram

```
                   +-----------------+
                   |     Server      |
                   |-----------------|
                   | Accepts clients|
                   | Broadcast msgs |
                   +--------+-------+
                            |
        +-------------------------+-------------------------+
        |                         |                         |
 +------v------+        +-------v------+        +-------v------+
 |   Client 1  |        |   Client 2   |        |   Client N   |
 | (Narrator)  |        | (Visualizer) |        | (LLM Agent)  |
 +-------------+        +--------------+        +--------------+
```

Each client can represent a different modality or agent, including visual renderers, dialogue systems, or interaction sensors.

Key Architectural Features:

- **Decentralized client behavior**: Clients act autonomously but are synchronized through the central server.
- **Threaded concurrency**: Each client connection is handled in its own thread for parallel I/O.
- **Extensibility**: Simple design makes it easy to integrate with audio, video, or AI modules.

---

3. Codebase Overview

3.1 `common.py`

Provides shared utilities:

- `HOST`, `PORT`: Network configuration.
- `encode(msg)`: Encodes messages to UTF-8 bytes.
- `decode(data)`: Decodes bytes to strings.
- `log(msg)`: Timestamps and prints messages for traceability.

These utilities standardize communication and diagnostics across server and clients.

3.2 `server.py`

Implements the `Server` class with the following logic:

Initialization:

- Binds to a specified `HOST:PORT`.
- Listens for incoming client connections.

Connection Handling:

- Each new client is assigned a dedicated thread.
- The thread listens for messages from the client, then rebroadcasts them to all others using `broadcast()`.

Key Methods:

- `handle_client(conn, addr)`: Main loop for a client's communication.
- `broadcast(msg, sender_conn)`: Sends messages to all clients except the sender.
- `shutdown()`: Graceful exit, closes all connections.

Technical Highlights:

- Uses `threading.Thread` for each client.
- Keeps a global list of active connections for broadcast logic.
- Robust to client disconnections with try-except blocks.

3.3 `client.py`

Implements the `Client` class with a dual-thread model:

Startup:

- Connects to the server socket.
- Starts two threads:
  - One reads from the server and prints incoming messages.
  - One reads user input and sends messages to the server.

Key Features:

- Line-based input/output.
- Graceful exit on KeyboardInterrupt.
- Can be easily modified to automate input (e.g., from AI model or sensor).

---

4. Communication Protocol

Format:

- Messages are newline-delimited strings (`\n`).
- Encoded in UTF-8 for compatibility.

Example Session:

```
Client 1 sends: "User A: Hello!"
Server receives and rebroadcasts:
Client 2 receives: "User A: Hello!"
Client 3 receives: "User A: Hello!"
```

Limitations (Current Work):

- No message framing or compression.
- No structured metadata (e.g., sender ID, timestamp).
- No security layer (e.g., authentication, encryption).

---

5. Use Case: Multimodal AI Story-Worlds

This framework enables scenarios where multiple agents co-create or co-navigate story environments:

Example Modalities:

- **LLM Narrator**: Receives prompts, sends narrative events.
- **Virtual Cameraman**: Receives camera commands and renders view.
- **Voice Actor Client**: Synthesizes dialogue or sound effects.
- **Sensor Client**: Triggers story branches based on gestures or gaze.

Interactive Example:

1. **Narrator**: Describes a scene ("A storm begins...")
2. **Sound Engine Client**: Plays thunder sounds
3. **Camera Client**: Pans toward a character
4. **LLM Agent**: Speaks a response ("We must find shelter!")

All clients remain synchronized through message rebroadcasting.

---

6. Extensibility and Future Work

Immediate Enhancements:

- **JSON Messaging**: Structured messages with keys like `{"type": "dialogue", "sender": "AI", "content": "Hello"}`.
- **Client Identifiers**: Associate messages with roles or modalities.
- **Server-side State**: Maintain global state (e.g., current scene, active characters).

Advanced Features:

- **WebSocket Support**: For web integration and low-latency bi-directional streaming.
- **Scene Graph Synchronization**: Directly integrate with game engines (e.g., Unity, Unreal).
- **AI Integration**: Allow automatic decision-making from GPT or other models.
- **Multimodal Message Parsing**: Route messages to appropriate subsystems (e.g., text to speech engine, gesture to camera).

---

7. Final Remarks

The Multicast-Multimodal Story Worlds framework establishes a lightweight and extensible infrastructure for real-time, distributed, narrative environments. Its modular design and simplicity make it an ideal platform for prototyping AI-driven and multimodal storytelling systems, including those involving natural language, sound, animation, and virtual agents.

Its utility can scale from classroom educational tools to complex interactive fiction platforms, especially when combined with large language models and immersive media systems.