

An Automated Evaluation Framework for Large Language Models in 3D Game Environments Using Structural and Behavioral Unit Testing

Dr. Archana Kumar¹, Siddharth Upadhyay²

¹Professor (AI&DS)

²Scholar, B.Tech (AI&DS) 4th Year

Department of Artificial Intelligence and Data Science

Dr. Akhilesh Das Gupta Institute of Professional Studies, New Delhi, India

¹profdrarchanakumar@gmail.com, ²siddharthu333@gmail.com

Abstract - This document presents a novel automated evaluation framework designed to facilitate the training of Large Language Models (LLMs) for 3D game engine scripting via Reinforcement Learning from Human Feedback (RLHF). While LLMs excel at standard text-based code generation, evaluating spatial and event-driven programming requires more than static syntactic checks. We propose a comprehensive testing pipeline wherein a pre-conditioned environment state (setup) and a human-authored reference solution are established based on a natural language prompt. The reference solution is subsequently swapped with the LLM-generated code. The generated code is evaluated through a dual-modal testing methodology: structural testing (edit mode) to verify instance hierarchy and property modifications, and behavioural testing (play mode) to assert runtime server-client interactions and temporal game states. This test-driven approach generates structured rubrics and high-fidelity reward signals, significantly improving the efficacy of RLHF in complex, multi-agent game environments.

Key Words: Large Language Models, RLHF, Automated Evaluation, Game Engine Scripting, Structural Testing, Behavioural Testing.

1. INTRODUCTION

The rapid advancement of Large Language Models (LLMs) has revolutionized automated code generation. However, deploying these models in complex 3D game engines introduces unique challenges. Unlike traditional software environments where functional input-output tests suffice, game development involves spatial reasoning, complex instance hierarchies (e.g., scene graphs, server script environments), and real-time server-client network architectures.

To fine-tune LLMs effectively for game scripting using Reinforcement Learning from Human Feedback (RLHF), standard string-matching or static Abstract Syntax Tree (AST) comparisons are insufficient. A generated script might be syntactically perfect but fail to execute intended runtime behaviors, or it might achieve the behavior using highly inefficient or insecure methods.

This paper details a Test-Driven LLM Evaluation Framework designed specifically for spatial computing environments. By defining an initial environment setup, establishing a human-verified reference script, and rigorously applying both structural (edit-time) and behavioral (run-time) unit tests, we create an automated, scalable pipeline. This pipeline scores LLM generations against strict rubrics, providing the high-quality penalty and reward signals necessary for advanced RLHF model alignment.

2. LITERATURE REVIEW

[1] Ouyang et al. (2022) introduced the Reinforcement Learning from Human Feedback (RLHF) paradigm, demonstrating that fine-tuning language models with human-aligned reward signals significantly improves instruction-following capabilities. This foundational methodology directly underpins our framework's core objective: generating high-fidelity, automated reward signals from unit test rubrics to align LLMs for complex game scripting tasks.

[2] Rozière et al. (2023) documented the capabilities of foundational models specifically trained on code, such as Code Llama. While these models exhibit strong performance on static code generation and standard algorithmic tasks, our research extends this domain by addressing the distinct gap in their ability to reason about 3D spatial hierarchies, coordinate math,

and event-driven game architectures.

[3] Cassano et al. (2023) presented MultiPL-E, a scalable approach to evaluating large language models for code generation across multiple programming languages. Their work highlights the fundamental necessity of execution-based evaluation over static text matching. Our framework builds upon this principle by introducing a dual-modal execution testing pipeline (structural and behavioral) specifically tailored for the unique constraints of spatial game engines.

[4] Zheng et al. (2023) demonstrated the effectiveness of automated benchmarking and rubric-based grading for evaluating model alignment. While their research focuses primarily on conversational alignment and using LLMs as judges, our framework adapts structured, rubric-based benchmarking to deterministic execution environments, replacing subjective conversational judges with objective, sandbox-driven game-state assertions.

3. METHODOLOGY AND PIPELINE

The body of the paper consists of the architectural methodology used to evaluate the LLM outputs. The evaluation pipeline operates on an isolated sandbox environment where scripts interact with the core data model of the 3D engine.

3.1 STATE INITIALIZATION (SETUP)

Before an LLM's code is evaluated, the environment must be pre-conditioned. The setup phase injects the necessary physical instances (e.g., 3D parts, GUI elements) and variables into the game's environment file. This ensures that the evaluation is strictly isolated to the logic requested in the user prompt, rather than the model failing due to missing prerequisite assets in the environment.

3.2 REFERENCE SOLUTION AND CODE SWAPING

For every prompt in our dataset, a human expert scripts a reference solution. This acts as the baseline for ground-truth capability. During the evaluation phase, the framework dynamically swaps the expert's reference solution with the LLM-generated trajectory. The subsequent tests are entirely agnostic to the author of the

code; they simply interact with the game state after the code executes.

3.3 STRUCTURAL TESTING (EDIT MODE)

Structural testing, or scene-check evaluation, occurs without advancing the game engine's internal clock. It verifies the static state of the environment post-execution. Assertions in this phase check if:

- Specific instances (Parts, Scripts, Models) were instantiated in the correct architectural hierarchy.
- Physical properties (Color, Size, spatial coordinate vectors) match the prompt's constraints. T
- The LLM successfully retrieved target components using API methods like `GetChildByName()`.

3.4 BEHAVIORAL TESTING (PLAY MODE)

Because game scripts govern temporal loops and remote events, structural testing alone is inadequate. Behavioral testing is divided into server-side and client-side checks. The testing framework mimics runtime execution where:

- Coroutine yields and temporal loops (e.g., thread sleeping) are simulated.
- Server-to-Client communication via Network RPCs (Remote Procedure Calls) is intercepted and asserted.
- Player tracking logic (e.g., updating a leaderboard over 10 seconds of gameplay) is validated dynamically.

Fig -2: Pipeline architecture for automated LLM testing in 3D environments.

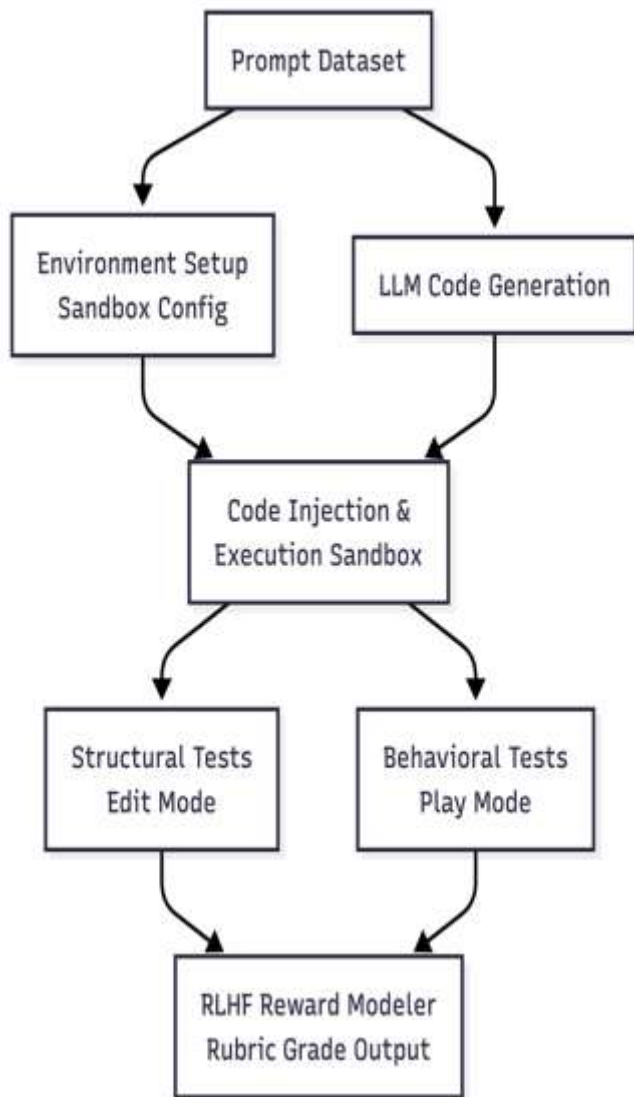


Figure 2 illustrates the comprehensive end-to-end workflow designed to automate the evaluation of LLM-generated scripts within 3D spatial environments.

The pipeline initiates with a curated prompt dataset that serves a dual purpose: it prompts the LLM to generate the target algorithmic logic while simultaneously establishing the initial physical and hierarchical state of the testing sandbox (Environment Setup).

4. EVALUATION RUBRICS AND ARTIFACTS

To provide granular feedback for RLHF, test assertions are mapped to a structured rubric. Each test yields a pass/fail metric weighted by its to the overall task.

Once the LLM produces the code, it is securely injected into this pre-conditioned execution sandbox. The framework then executes a dual-modal evaluation strategy, subjecting the state to Structural Tests (Edit Mode) to verify static constraints such as object instantiations and property modifications, alongside Behavioral Tests (Play Mode) to dynamically validate temporal loops and client-server network interactions. Ultimately, the deterministic outcomes from these robust unit tests are synthesized into a weighted, graded rubric. This rubric seamlessly feeds high-fidelity penalty and reward signals into the RLHF Reward Modeler, driving continuous and precise alignment of the underlying language model.

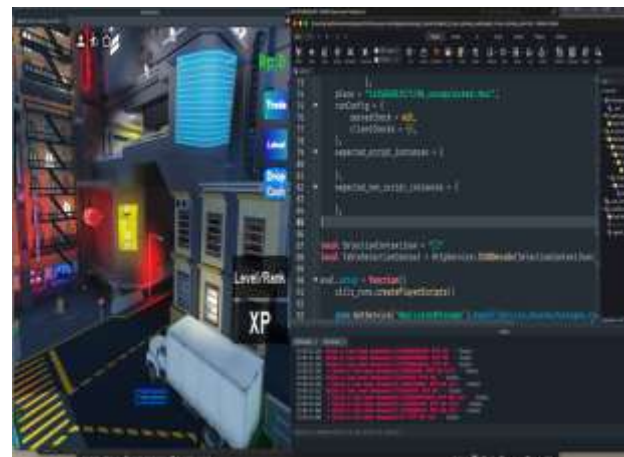


Fig -1: Developer interface during the evaluation phase, showcasing the visual state of the simulated engine interacting directly with the automated test configuration and assertion parameters.

4. EXPERIMENTAL ANALYSIS AND FAILURE TAXONOMY

Implementing an automated testing framework for LLM-generated code in a live 3D engine introduces complexities not present in standard text-based programming environments (such as Python data structures or web frameworks). During the empirical evaluation of our framework, we identified critical challenges regarding execution determinism and categorized the unique failure modes exhibited by LLMs in spatial computing contexts.

4.1 OVERCOMING NON-DETERMINISTIC EXECUTION

Rubric ID	Test Category	Weight	Description & Rationale
correctness-1	Structural (Edit Mode)	Major	Verify Control Script: Asserts that the LLM placed the executable script in the Server Logic Provider. Fails if absent.
correctness-2	Behavioral (Server Check)	Major	Verify Spawn Logic: Asserts that instances spawn at designated temporal intervals using thread spawning mechanisms.
completeness-1	Behavioral (Client Check)	Minor	Jump Verification: Asserts the client's GUI state updates properly when the player character jumps.

Unlike pure algorithmic challenges, game engine scripting is inherently tied to physics ticks, frame rendering rates, and network latency. Evaluating behavioral tests (Play Mode) requires simulating time passing. We observed that LLM-generated code often relies on asynchronous thread yielding (e.g., `task.wait()`) or event-driven callbacks.

To ensure the RLHF reward model receives deterministic and reproducible signals, our framework enforces strict temporal boundaries. Execution environments are sandboxed with hard timeouts to prevent engine hangs caused by unyielded infinite loops (e.g., a `while true do` block missing a `yield`). Furthermore, server-client RPCs (Remote Procedure Calls) are mocked via synchronized event queues to ensure behavioral assertions execute only after the necessary packet transmission intervals have elapsed.

4.2 TAXONOMY OF LLM SPATIAL REASONING FAILURES

By evaluating thousands of LLM-generated trajectories against our dual-modal rubrics, we established a taxonomy of common failure modes unique to 3D engine scripting. Traditional static analysis (like Abstract Syntax Tree matching) fails to detect these errors, proving the necessity of our execution-based framework. The primary failure categories include:

1. **Hierarchical Misplacement:** The LLM generates syntactically flawless code but instances the script or object in the wrong architectural container. For example, placing a UI-handling `LocalScript` inside the

global `Workspace` rather than the secure `PlayerGui` container, rendering the script inert. Our Structural Tests (Edit Mode) explicitly catch these architecture violations.

2. **Temporal and Yielding Errors:** The model hallucinates the lifecycle of an instance, attempting to manipulate properties of a 3D part before it has fully replicated from the server to the client. Our Behavioral Tests (Play Mode) detect these race conditions.

3. **Spatial and Geometric Hallucinations:** When prompted to manipulate spatial logic, LLMs frequently miscalculate coordinate math. For instance, calculating a bounding box or using quaternions/coordinate frames to rotate an object often results in overlapping geometries.

4. **Security and Trust-Boundary Violations:** LLMs occasionally attempt to grant the client authority over server-authoritative physics or monetary transactions. The sandbox constraints identify when a client script improperly attempts to write to a server-locked property.

7. CONCLUSIONS

The deployment of Large Language Models for 3D spatial programming necessitates specialized evaluation pipelines.

By abstracting the evaluation process into Setup, Reference substitution, Structural Testing, and Behavioral Testing, we created a highly reliable programmatic grader. This framework successfully bridges the gap between text-based code generation and real-time game engine execution.

Consequently, the granular rubrics generated by our unit tests offer robust reward signals, proving highly effective for refining LLMs through Reinforcement Learning from Human Feedback (RLHF). Future work will explore automating the generation of the unit tests themselves to further scale the RLHF dataset without manual overhead.

ACKNOWLEDGEMENT

The author thanks Dr. Archana Kumar, Professor and Head of Department, Department of Artificial Intelligence and Data Science, Dr. Akhilesh Das Gupta Institute of Professional Studies, New Delhi, for her constant support and guidance throughout the course of this project work.

REFERENCES

- [1] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... & Lowe, R. "Training language models to follow instructions with human feedback." *Advances in Neural Information Processing Systems*, 35 (2022): 27730-27744.
- [2] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., ... & Zaremba, W. "Evaluating Large Language Models Trained on Code." *arXiv preprint arXiv:2107.03374* (2021).
- [3] Rozière, M., Lachaux, M. A., Chatusot, L., & Lample, G. "Code Llama: Open Foundation Models for Code." *arXiv preprint arXiv:2308.12950* (2023).
- [4] Liu, J., Xia, C. S., Wang, Y., & Zhang, L. "Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation." *Advances in Neural Information Processing Systems*, 36 (2023).
- [5] Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., ... & Anandkumar, A. "Voyager: An Open-Ended

Embodied Agent with Large Language Models." *arXiv preprint arXiv:2305.16291* (2023).

- [6] Zheng, L., Chiang, W. L., Ying, Y., Hao, S., ... & Xing, E. P. "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena." *Advances in Neural Information Processing Systems*, 36 (2023).
- [7] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. "Reflexion: Language Agents with Verbal Reinforcement Learning." *Advances in Neural Information Processing Systems*, 36 (2023).
- [8] Zan, D., Chen, B., Zhang, F., Lu, D., Wu, B., ... & Lou, J. G. "Large Language Models Meet NL2Code: A Survey." *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics* (2023).
- [9] Cassano, F., Gouwar, J., Nguyen, D., ... & Guha, A. "MultiPL-E: A Scalable and Polyglot Approach to Evaluating Large Language Models for Code." *IEEE Transactions on Software Engineering*, 49(7) (2023): 3840-3855.
- [10] Ammann, P., & Offutt, J. *Introduction to Software Testing*, 2nd Edition. Cambridge University Press (2016).