LLM-Obby: Large Language Models for Iterative Obstacle-Course Synthesis

Nathan A. Chi[1], **Vishnu Sarukkai**[1], **Sharon Zhang**[1], **Zander Majercik**[1], **Kayvon Fatahalian**[1]
[1]Stanford University
nathanchi@cs.stanford.edu

Abstract

We present **LLM-Obby**, a novel pipeline for the controllable synthesis of 3D obstacle-course environments with LLM-driven DSL generation and human-in-the-loop refinement. Central to our approach is an *iterative memory stream*—a persistent buffer that records past DSL scripts along with two forms of human reward feedback: modular selection (gating) and feature-level comments. In each cycle, a strong LLM agent ingests this enriched context, augmented by exploratory DSL exemplars and rollout perturbations to the current state, and emits a refined course proposal. The human user then selects preferred components and annotates desired features, which a smaller parser LLM converts into additional DSL fragments. By seeding subsequent prompts with these gated exemplars, LLM-Obby progressively biases generation toward designs that harmonize user preferences with novel gameplay experiences. We show that our closed-loop method produces obstacle courses that are substantially more playable and better aligned with user objectives than those from off-the-shelf LLM baselines. We conclude with an analysis of computational overhead and a discussion of the challenges inherent in automating evaluation within orchestrated LLM workflows.

1 Introduction

Recent advancements in large language models (LLMs), used in tandem with specialized domain-specific languages (DSLs) for 3D scenes, have made automatic scene generation and editing markedly more accessible (Zhang et al., 2024). By coupling an LLM's ability to parse natural language prompts with a DSL's structured control over geometric and semantic parameters, these pipelines allow users to rapidly prototype and refine virtual environments.

Despite these promising developments, general-purpose solutions (like LLMs) often struggle to adapt to the highly specialized requirements of many digital design domains—especially when the desired metrics (e.g. "scene alignment to player preferences" in a parkour game) are abstract or require human-like judgment. There is a need for task-specific customization of LLMs, allowing them to learn from previous design failures or user feedback and thus better accommodate the abstract metrics that govern user experiences in complex 3D scenes.

To this end, we harness large language models to generate and preference-align obbies—3D obstacle-course games on Roblox that blend precision jumps, narrow ledges, spinning platforms, and other navigational puzzles. Each element of an obby can be parametrized to calibrate overall challenge and playability, yet vanilla LLMs lack the domain expertise needed to balance these factors effectively. Treating obbies as a testbed for abstract, domain-specific design objectives, we introduce an online bootstrapping pipeline that iteratively refines text-to-obby generation through alternating phases of stochastic exploration and human-in-the-loop exploitation. In each exploration phase, we inject random DSL exemplars and perturbations to diversify candidate layouts; in the subsequent exploitation phase, users gate their favorite components and supply feature-level annotations that a secondary LLM converts into new DSL constraints. By seeding the next exploration with these gated exemplars and constraints, our pipeline dynamically balances creative novelty with targeted preference alignment, rapidly converging on obstacle courses that hit bespoke difficulty and playability targets.

2 Method

3 Related Work

In-Context Learning A growing body of work has demonstrated that selecting or optimizing examples within a prompt can substantially improve an LLM's performance on downstream

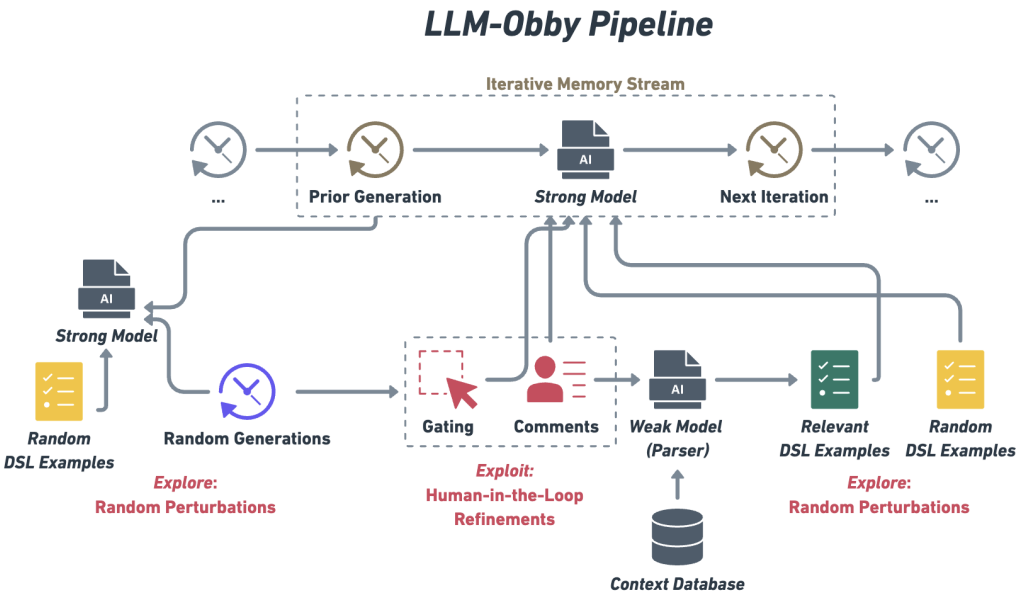

Figure 1: System overview for **LLM-Obby**. The pipeline alternates between *exploration* (stochastic generation of candidate obbies) and *exploitation* (human-in-the-loop selection, gating, and textual feedback), forming a closed-loop iterative refinement.

tasks (Khattab et al., 2023; Agarwal et al., 2024; Opsahl-Ong et al., 2024). Notably, past work has shown that including curated examples or intermediate reasoning traces can reduce the need for extensive human-authored prompts (Agarwal et al., 2024) and surpass purely instruction-tuned approaches (Opsahl-Ong et al., 2024). Our approach adopts a similar philosophy, but rather than requiring offline or supervised selection of examples, we employ an iterative method akin to Sarukkai et al., 2025, where successful instances of obby designs and playthrough data are collected on the fly and reused to guide future generations.

Iterative Improvement and Human in the Loop Signal Prior work on refining LLM-generated DSL outputs has centered on embedding structured feedback loops to incrementally enhance code quality. Reflexion, for example, treats the model as an agent that appends succinct "reflections" on its own errors to subsequent prompts—enabling self-correction without any parameter updates (Shinn et al., 2023). Re-ReST builds on this by adding a dedicated reflection model during self-training, which diagnoses low-quality generations and generates targeted repairs that enrich the downstream DSL training set (Dou et al., 2024). However, in our pipeline, we replace purely automated feedback with direct human-in-the-loop signals, drawing on extensive interactive-systems research (Wu et al., 2022). This human-centric setup is by design – so that a user can iteratively guide the agent to produce a playable obby without needing to write or understand the underlying code. Unlike text-only improvement schemes that adjust prompts in isolation (Zhao et al., 2024), our approach fuses both textual critiques and visual gating decisions into concrete edits of the DSL primitives, creating a fully end-to-end, closed-loop refinement process.

4 Method

We propose an iterative text-to-obstacle-course generation pipeline that couples the obstacle DSL from (Majercik et al., 2025) with a human-in-the-loop self-improvement mechanism (Figure 1). At its core is our *iterative memory stream*, a persistent buffer that carries forward every salient artifact of the obby-generation loop: the working obby, human-in-the-loop rewards, and random perturbations to break out of local minima. Our approach proceeds in two main phases:

1. **Procedural generation of "gold" examples.**
   - A custom DSL synthesizer produces a library of high-quality obstacle courses.
   - Although each example is low-entropy (containing only one obstacle type and generally uninspiring to play), this library encodes correct DSL usage and desirable design patterns that serve as in-context examples for later iterations (Majercik et al., 2025).

2. **Iterative refinement.** We then repeat the following steps until the user is satisfied:

   a. **Generate perturbations.**
      - Randomly sample a subset of gold examples and extract one obstacle component from each.
      - Apply these high-quality features as stochastic DSL perturbations to the current working obby; these serve as potential high-temperature rollouts to the next iteration of the process and are displayed to the user as example variants.
      - In the first iteration (when the memory stream is empty) these features become directly included in the initial candidate designs.

   b. **Collect human feedback.** The user rewards generated rollout variant through:

      i. **Gating.** In our GUI, the user selects portions of the perturbed obbies to include in the next working state. This selection acts as a gate in a residual-style architecture, allowing only favored stochastic features into the memory stream.

      ii. **Comments (LLM-driven RAG-like updates).** The user enters text queries describing desired features. A lightweight backend LLM parses these comments into the closest-matching DSL keys (e.g., `LavaDoors`). Then, the pipeline retrieves matching obstacles from the gold library, and injects them into the memory stream.

   c. **Update the working obby.** The LLM is prompted, using the accumulated context in the memory stream, to apply minimal edits that preserve previously rewarded components, producing the next iteration's working obby.

When the working obby meets the user's specifications, it can be directly exported as a fully playable Roblox obstacle course.

Implementation details We implemented this pipeline fully in Python; we created the user-facing GUI in `matplotlib`. For our strong LLM, we used `gpt-o4`; for our weaker parser LLM, we used `gpt-4o`.

5 Results and Analysis

Our empirical results demonstrate that our iterative pipeline can generate custom obbies that meet user specifications, something a vanilla LLM-to-DSL approach consistently failed to achieve.

5.1 Experiments

As with any human–in–the–loop system, large-scale evaluation is challenging without fully automated pipelines. To approximate real-world use, we selected a set of 15 diverse human-written obby descriptions (simulating typical player specifications) covering precision jumps, lava hazards, narrow walkways, obstacle walls, and mixed-obstacle layouts. We compared two generation strategies:

- **Naïve LLM baseline:** A single-shot prompt to generate DSL code (with the same number of initial DSL context examples) directly from the description.
- **Iterative LLM-Obby pipeline:** Our memory-stream–based, human-in-the-loop refinement process.

Using the iterative pipeline, every description converged to a human-approved, playable obby in an average of three iterations. In contrast, the naïve baseline failed to produce a fully playable course for any of the test descriptions. One example of our method's generation is in Figure 2.

5.2 Playability

Although playability was not explicitly optimized, our pipeline produced playable obbies in 100% of cases, whereas the naïve baseline yielded none. We attribute this improvement to two complementary factors:

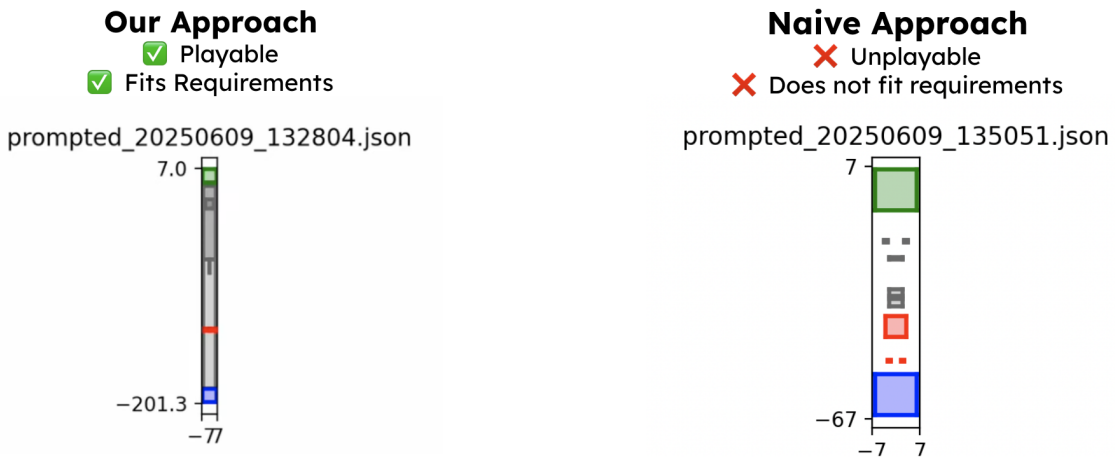

Figure 2: Bird's-eye comparison between the naïve LLM baseline and our LLM-Obby pipeline for the prompt "an obby with a long vertical axis, wall jumps, tightrope walks, lava doors, lava jumps, and staircase elements." The LLM-Obby output is fully playable and meets every requirement; the naïve baseline produces uncrossable jumps and misuses obstacle components, rendering it unplayable. See the full pipeline approach here.

1. *Human-in-the-loop guidance:* User selections and comments implicitly steer the design toward feasible, traversable layouts.

2. *Gold-example context:* The memory stream's intelligent inclusion (via gating and RAG-like comment updates) of proven, playable obstacle patterns biases the LLM toward structurally sound designs.

5.3 Reflection, Limitations and Future Directions

Our iterative text-to-obby pipeline—including the memory-stream architecture, custom DSL, and interactive GUI—successfully enables users to co-design playable obstacle courses without writing code by hand. In early tests, players found that the combination of stochastic "inspiration" from gold examples and fine-grained gating or comment-driven edits felt intuitive, and that the tool struck a good balance between creative freedom and structural guidance.

Performance and Scalability A key limitation of the current system is its runtime cost. Each iteration requires orchestrating multiple LLM calls (for perturbation generation, comment parsing, and context-aware editing), even when parallelized across GPUs or CPU threads. As a result, one round of iteration currently takes on the order of 60–90 seconds. While this is still faster than manual DSL coding, it introduces friction for users who wish to make rapid, exploratory tweaks. Moreover, the end-to-end latency limits our ability to scale to larger user studies or to support real-time collaborative sessions.

Dependence on External APIs Because we rely on a hosted LLM for both generation and comment parsing, our pipeline inherits the usual trade-offs: network latency, API rate limits, and usage costs. In environments with poor connectivity or tight budgets, these dependencies could become prohibitive. In addition, frequent context re-encoding and token churn can drive up API usage without commensurate improvements in quality.

Evaluation Challenges Our initial evaluation (with hand-curated descriptions and human judgments) demonstrates feasibility, but does not fully capture long-term usability, learning effects, or design diversity. Quantitatively assessing "playability" requires either large-scale human playtesting or the development of automated agents capable of navigating generated courses. Without such tools, it remains difficult to compare designs objectively or to tune the pipeline for specific difficulty levels.

References

Rishabh Agarwal, Avi Singh, Lei Zhang, Bernd Bohnet, Luis Rosias, Stephanie Chan, Biao Zhang, Ankesh Anand, Zaheer Abbas, Azade Nova, et al. 2024. Many-shot in-context learning. *Advances in Neural Information Processing Systems*, 37:76930–76966.

Zi-Yi Dou, Cheng-Fu Yang, Xueqing Wu, Kai-Wei

Chang, and Nanyun Peng. 2024. Re-rest: Reflection-reinforced self-training for language agents. *arXiv preprint arXiv:2406.01495*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*.

Zander Majercik, Sharon Zhang, William Wang, Brennan Shacklett, Maneesh Agrawala, and Kayvon Fatahalian. 2025. Modeling human limitations in reinforcement learning agents for difficulty assessment of obstacle course games.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. *arXiv preprint arXiv:2406.11695*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Xingjiao Wu, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma, and Liang He. 2022. A survey of human-in-the-loop for machine learning. *Future Generation Computer Systems*, 135:364–381.

Yunzhi Zhang, Zizhang Li, Matt Zhou, Shangzhe Wu, and Jiajun Wu. 2024. The scene language: Representing scenes with programs, words, and embeddings. *arXiv preprint arXiv:2410.16770*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.