VIBEARENA: SELF-EVOLVING EXECUTABLE WORLDS FOR EMBODIED AGENTS

Anonymous authors
Paper under double-blind review



Figure 1: VibeArena generates (a) diverse environments with (b) structurally complex, physically specified assets. (c) Objects include internal contents accessible through articulated mechanisms, and (d) the worlds support interaction with different robot embodiments.

ABSTRACT

We present **VibeArena**, which builds executable 3D worlds in code and improves construction knowledge and agent harness policies using construction and execution experience. The challenge is to reuse successful builds and repairs while preserving relations between object mechanisms and spatial context. Therefore, VibeArena links scene layout, object geometry, articulation, physical properties, and tasks in a shared world program that coordinates multiple agents. Hierarchical intermediate representations carry interaction constraints into code-based object generation, while execution feedback directs targeted revisions to structures that determine task feasibility. Validated builds provide templates, accepted repairs provide conditional rules, and their histories refine inspection and repair scheduling. Self-evolution consolidates construction experience into reusable templates, conditional rules, and scheduling policies, with rules and policies validated across scenes and refined or retired as evidence accumulates. Experiments demonstrate leading performance in object and scene generation quality, physical stability, and downstream tasks, with visual examples of diverse scenes and detailed articulated assets. VibeArena uses lessons from building and testing worlds to improve how it constructs later worlds.

1 INTRODUCTION

A generated world can look complete yet fail as soon as an embodied agent tries to use it (Li et al., 2023; Nasiriany et al., 2026). Each interaction tests whether object mechanisms, spatial layout, and physical behavior work together (Wang et al., 2026c). Building such worlds goes beyond generating their appearance: they need to work when an agent acts.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

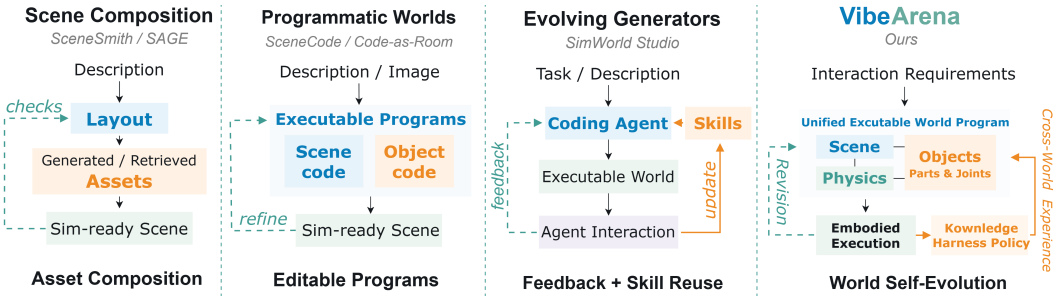


Figure 2: From world generation to generator self-evolution. VibeArena builds executable worlds through code and uses construction and execution experience to improve its construction knowledge and agent harness.

Recent advances make these worlds increasingly accessible through language-driven generation and editable scene programs. General-purpose LLM agents can turn language into 3D scene code (Tang et al., 2026; Xu et al., 2026). Specialized generators add simulation-ready layouts and assets (Pfaff et al., 2026; Xia et al., 2026a), while executable representations make construction editable (Wang et al., 2026b; Yang et al., 2026b) (Fig. 2). Yet these capabilities alone do not reveal the cause of a failed embodied interaction: an object may appear faulty because of its relation to nearby objects. Repair thus requires linking task requirements to object mechanisms and spatial context.

Consider a cabinet whose door, contents, and nearby furniture are generated at different stages. When opening or retrieval fails, a repair loop focused on the cabinet may repeatedly edit its geometry while overlooking the blocked door swing or insufficient access space. Preserving these relations throughout construction provides scene context for object modeling and links failures to the relevant construction decisions. We express them in a shared world program, linking interaction requirements to scene relations, object parts, joints, and physical properties. This supports a code-driven, *vibe* workflow in which users describe a world's intended activities and character, and the generator develops the corresponding structures through editable code.

Constructing and testing these editable worlds also produces reusable experience (Kang et al., 2026; Chen et al., 2026). A validated cabinet provides a mechanism template; a successful clearance edit yields a rule relating door swing to nearby furniture (Ran et al., 2026). Failed door edits followed by a layout fix motivate an earlier spatial check. These lessons can improve both what a generator builds and how it checks and revises worlds (Yu et al., 2026; Wei et al., 2026; Wang et al., 2026d; Zheng et al., 2026). Repeated feedback can also fit experience to the worlds that produced it (Xia et al., 2026b), motivating independent confirmation and continued review of retained experience.

We present **VibeArena**, which builds executable 3D worlds in code and improves construction knowledge and the agent harness using construction and execution experience (Fig. 3). Multiple agents coordinate construction through hierarchical intermediate representations (IRs) that carry interaction requirements and layout constraints into a shared world program linking scene layout, object geometry, articulation, physical properties, and tasks. Code-based object generation uses this context to construct articulated assets, while simulation tests the resulting interactions and supplies execution feedback for targeted program revision. Realization mappings link requirements to structures and associate execution feedback with construction decisions, and build dependencies determine which assets an edit regenerates. Across worlds, validated builds provide templates, accepted repairs provide conditional rules, and their histories refine inspection and repair scheduling. We consolidate these updates individually through cross-scene validation, with later evidence guiding refinement or retirement. The world programs, construction knowledge, and agent harness evolve while the underlying generation models and embodied controller remain fixed.

Our contributions are:

- **An agentic framework for executable world generation.** VibeArena coordinates code-based scene construction, Articraft-based articulated asset generation, simulation, and targeted revision to build interactive 3D worlds.
- **Unified, fine-grained world construction.** Hierarchical IRs coordinate these agents through a shared world program linking scene layout, object geometry, articulation, phys-

ical properties, and tasks. These links carry interaction constraints and scene context into code-based object generation and trace execution feedback to the relevant structures.

- **Self-evolution from construction experience.** VibeArena distills construction and execution experience into reusable knowledge with explicit applicability conditions and harness policies for inspection and repair scheduling.

2 Related Work

From Interactive Assets to Executable Worlds. An executable world requires both usable objects and the scene relations that make their use possible. Object generation has advanced from geometry and appearance (Xiang et al., 2026) to compositional parts (Chen et al., 2025; Yang et al., 2026c; Lin et al., 2026) and simulation-ready articulation (Cao et al., 2026a;b; Wu et al., 2026); programmatic modeling makes such structures explicit and revisable (Jones et al., 2020; Dai et al., 2026; Le et al., 2025; Zhou et al., 2026). At scene scale, procedural and agentic systems coordinate assets through spatial constraints and physical checks (Raistrick et al., 2024; Yang et al., 2024; Pfaff et al., 2026; Xia et al., 2026a; Wang et al., 2026c; Cheng et al., 2026), while executable representations make assembled worlds editable (Yang et al., 2026b; Wang et al., 2026b; Feng et al., 2026; Xiao et al., 2026). Tool-based world building (Ning et al., 2026), recursive scene programs (Li et al., 2026), and code-maintained world states (Lehrach et al., 2026) further develop this programmatic direction. Recent systems integrate asset generation, affordance annotation, and task-driven scene assembly with stateful editing (Wang et al., 2026c;a). VibeArena links interaction requirements to object mechanisms and surrounding layout through realization mappings. These links support targeted repair, while accepted revisions yield conditional knowledge for subsequent construction.

Environment Design for Embodied Agents. A generated world also matters through the tasks it affords an embodied agent. Adaptive curricula select or evolve environments according to solution transfer or the challenge they present to a learner (Wang et al., 2019; Dennis et al., 2020; Parker-Holder et al., 2022). Language models have been used to create tasks and simulator code for robot learning (Wang et al., 2024; 2023b), and agent performance can guide subsequent changes to configurations, terrain, or executable tasks (Zala et al., 2024; Liang et al., 2024; Faldor et al., 2025). SimWorld Studio combines environment generation with navigation curricula that adapt to learner performance (Kang et al., 2026). These works show how feedback can shape an agent's training experiences. In our construction loop, a fixed diagnostic controller tests whether a generated world supports its requested interactions, and the resulting evidence guides world repair. What should later worlds learn from successful construction and repair?

Self-Evolution of Agentic Generators. Successful construction and repair reveal both reusable structural conditions and better inspection and revision strategies. SimWorld Studio turns verifier feedback into reusable tools and procedural skills for UE5 environment generation (Kang et al., 2026). More generally, reflection and executable skill libraries retain feedback or behaviors across tasks (Shinn et al., 2023; Wang et al., 2023a), while long-horizon agent systems turn execution traces into localized memory or harness updates (Yu et al., 2026; Wei et al., 2026; Liu et al., 2026). Work on evolving improvement strategies also changes the procedure used to produce later solutions (Wang et al., 2026d; Zheng et al., 2026). Regularized harness evolution addresses overfitting to repeatedly reused tasks by constraining candidate proposals and selection and pruning unproductive components (Xia et al., 2026b). Compared with SimWorld Studio's tool and skill reuse, VibeArena emphasizes conditional knowledge of mechanism–scene relations and policies for inspection and repair. Our ablations isolate knowledge and harness contributions and compare conditional knowledge with retrieved examples and plain summaries under matched retrieval and context budgets. Appendix Table 6 compares related systems.

3 Method

VibeArena turns interaction requirements into executable 3D worlds through code, coordinating multiple agents through a shared world program (Fig. 3). Section 3.1 explains hierarchical world construction, Section 3.2 covers execution-guided revision, and Section 3.3 describes how validated experience updates construction knowledge and the agent harness for later worlds.

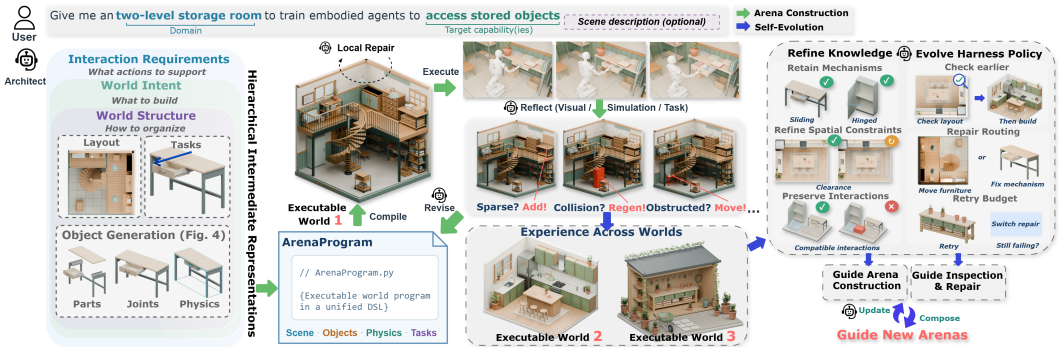


Figure 3: Overview of VibeArena. World construction (green) turns interaction requirements into executable 3D worlds through code; self-evolution (blue) uses construction and execution experience to improve construction knowledge and the agent harness.

3.1 Hierarchical World Construction

Interaction requirements constrain both scene layout and object structure: opening a cabinet requires a movable door and space for its motion. VibeArena passes these constraints from scene planning to object generation through a shared program and hierarchical intermediate representations (IRs).

ArenaProgram. We represent a world with ArenaProgram, which stores executable specifications and their construction metadata:

$$P = (S, O, \Phi, U, T; \mathcal{R}, \mathcal{B}), \tag{1}$$

where S specifies scene regions, placements, and spatial relations; O describes object geometry, materials, parts, and joints; Φ defines physical properties; and U and T specify operations and tasks. Shared entity identifiers connect these components. For example, a retrieval task refers to a target inside a cabinet, an opening operation addresses its door joint, and the scene records the relevant containment and spatial relations. These references retain the same entity identities through construction and simulation. This common representation allows a revision to address both an object's internal structure and its relation to the surrounding scene.

The metadata retains the purpose and dependencies of construction decisions. *Realization mappings* $\mathcal{R}$ link requirements to the decisions and entities that implement them, allowing feedback on an interaction to be traced to the relevant structure. *Build dependencies* $\mathcal{B}$ record the inputs used to generate each asset and determine what an edit regenerates. For the cabinet example, realization mappings connect the access requirement to the door mechanism and the surrounding layout decisions. Build dependencies then distinguish a placement-only revision, which can reuse geometry, from a change to parts or interfaces that rebuilds affected assets. Appendix A.1 details these mappings and the compilation procedure.

Hierarchical Intermediate Representations. Hierarchical IRs coordinate the planning and object agents by translating interaction requirements into concrete construction choices. From the domain and capability profiles, a construction planning agent jointly derives an interaction challenge and world intent while preserving activities stated in the request. The challenge specifies what the world should exercise; the intent describes the contents and conditions needed to support it. The agent then resolves this intent into a layout plan with functional groups, object requirements, spatial relations, and access clearances. The resulting object specifications retain these relations and access requirements during synthesis. Stage checks validate the IR structure and references before these constraints enter the next stage; the assembled world is later assessed through construction checks and execution.

Object Generation. Object generation wraps and adapts Articraft's modeling SDK and compiler (Zhou et al., 2026) for scene-conditioned synthesis; VibeArena supplies object requirements, scene context, reusable construction knowledge, and execution feedback (Fig. 4). *(i) Context-conditioned modeling.* The object agent combines ArenaProgram's object specification and scene

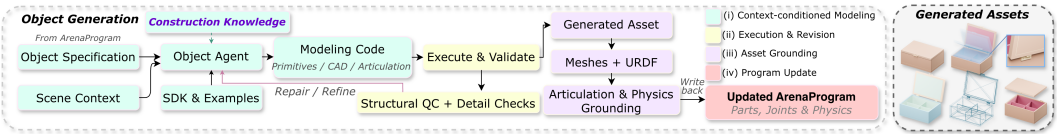


Figure 4: Object generation pipeline and examples of generated articulated assets.

context with construction knowledge, SDK resources, and examples. It writes code for geometry, materials, semantic parts, and articulation using primitives and CAD operations. *(ii) Execution and revision.* Code execution is followed by structural quality control (QC) and detail checks; compiler feedback and rendered observations guide code repair and refinement. *(iii) Asset grounding.* The generated asset is exported as meshes and URDF, binding realized parts and joint topology to the object specification. Geometry-conditioned predictions supply physical parameters; collision geometry and assigned mass determine inertial properties. *(iv) Program update.* Grounded parts, joints, and physical properties are written back to ArenaProgram for scene assembly and simulation.

3.2 Executable Worlds and Task Execution

Simulation-ready scenes still require an embodied check of whether intended interactions work. VibeArena uses execution evidence to revise the world while keeping the diagnostic controller fixed.

Executable World. We instantiate the compiled ArenaProgram with the embodied agent through a simulator backend interface. Our current implementation uses Genesis (Genesis AI Team, 2026). Generated assets are placed according to the scene layout, with articulation and physical properties bound to simulator entities. Object, part, and joint identifiers are preserved across the program and runtime, so task references and observations address the same structures. The resulting executable world can be exported and reloaded with its tasks and initial state, providing a common interface for repeated embodied execution.

Execute. Tasks express the intended interactions as initial conditions, goals, and constraints over the world state. An execution-time task planner produces a sequence of supported operations for the current task and world state. The embodied controller executes these through fixed motion, gripper, and simulation-step primitives. A shared reset protocol allows the same task conditions and seeds to be replayed across world revisions. Task progress and success are evaluated from simulator state, while execution traces retain observations, contacts, collisions, and the operation at which a failure occurs. This connects the intended use of a world to the physical behavior of its generated structures.

Reflect and Revise. Execution evidence is combined with geometric checks and synchronized visual observations to locate mismatches between intended and realized interactions. Task failures indicate where an intended interaction breaks down; geometric, visual, and physical evidence helps distinguish construction defects from execution failures. Shared identifiers and realization mappings associate the findings with relevant program components, while inspection or controlled probes can resolve ambiguous cases. A world reasoner proposes patches Δ to the accepted program at the implementation, object, scene, or physics layer. Execution state x_t and a candidate program revision follow different updates:

$$x_{t+1} = F_P(x_t, a_t), \qquad P^{\text{cand}} = \text{Apply}(P, \Delta), \tag{2}$$

where F_P is simulator dynamics and a_t a controller action. Each candidate is recompiled and evaluated on the same tasks. Acceptance requires valid construction and execution, preservation of task intent and safety, resolution of the targeted issue, and no new detected gaps or protected-task regressions. The resulting program and evidence support subsequent execution and supply construction experience for self-evolution. Appendix A.2 details diagnosis and revision acceptance.

3.3 Experience-Guided Self-Evolution

Construction and execution provide lessons for subsequent world generation (Kang et al., 2026). VibeArena distills these lessons into construction knowledge and agent harness policies. Construc-

tion knowledge guides object structure and spatial relations; harness policies guide inspection and repair scheduling.

Construction knowledge. Construction knowledge comprises validated templates and conditional repair rules. Templates preserve part proportions, joints, and relative relations; rules link a revolute or prismatic mechanism to a neighboring object or contained target. Rule features encode normalized joint-local position and size, joint range, and available motion targets, preserving structural context without world-specific names or coordinates.

Rule induction selects measured features changed by an accepted repair and places interval boundaries between their original and validated values, retaining the remaining context. Semantic and mechanism matching determines applicability in new worlds; every guided construction or repair undergoes full validation. Boundary tests can extend conditions, add exclusions, or withdraw conflicting rules. Task failure alone leaves feasibility unresolved, and ambiguous bindings, missing measurements, or conflicting evidence cause abstention.

Agent harness policies. Construction traces suggest better inspection and repair schedules: repeated object repairs followed by a successful layout revision can motivate an earlier spatial check. Policies map construction stage, issue evidence, prior attempts, and budget to check ordering, repair routing, or retry allocation, preserving available actions and validation criteria. Each candidate is compared with the current schedule on two to four inputs excluded from its proposal sources, using matched worlds, tasks, models, seeds, and budgets. Both schedules pass full validation on every input; acceptance requires no increase in calls, tokens, or time on any input and a strict reduction somewhere.

Experience consolidation. To control overfitting, we validate each rule or policy update on learning scenes excluded from its supporting traces. Subsequent uses record applicability, outcomes, and costs; repeated failures trigger paired revalidation with the entry disabled. Confirmed regressions prompt refined conditions or retirement; ambiguous failures remain unresolved. Updates occur between construction runs, with evaluation snapshots unchanged.

Appendices A.3, A.4, A.5, and D.9 detail knowledge induction, policy validation, experience snapshots, and consolidation.

4 Experiments

We structure the experiments around four questions: (1) Does construction experience improve world generation over successive evolution rounds (Section 4.2)? (2) How does VibeArena compare with methods in scene quality, object quality, and physical stability (Section 4.3)? (3) How do construction components, self-evolution sources, experience representation, and experience consolidation affect performance (Section 4.4)? (4) Can an embodied controller complete tasks in the generated worlds (Section 4.5)?

4.1 Experimental Setup

We collect construction experience from 85 descriptions from the SceneSmith dataset and 50 from the Function-Room dataset (Feng et al., 2026), then test transfer on 30 disjoint SceneEval descriptions. Scene baselines include HSM (Pun et al., 2026), LayoutVLM (Sun et al., 2025), SceneWeaver (Yang et al., 2026a), SceneSmith (Pfaff et al., 2026), SAGE (Xia et al., 2026a), SceneCode (Wang et al., 2026b), Code as Room (Yang et al., 2026b), and FuncRoom-Agent (Feng et al., 2026). We also evaluate general agents built on GPT-6 Astra and DS-V4.1-Flash (Xu et al., 2026) with access to the same 3D modeling tools, authoring resources, and per-world resource limits as VibeArena; each evaluation uses the applicable subset. SceneEval (Tam et al., 2026) measures object count (CNT), attributes (ATR), object–object (OOR) and object–architecture (OAR) relations, support (SUP), accessibility (ACC), navigability (NAV), collisions (COL), and out-of-bounds placement (OOB). On Function-Room learning descriptions, we report functional-zone coverage (FZC), functional-object coverage (FOC), and valid, relevant object rate (FOP). We use GPTEval3D (Wu et al., 2024) for object quality, stability and displacement for passive physics, and task success for embodied interaction. Unless otherwise specified, results average five independent evolution runs.

Further experimental details are in Appendix D. Mesh (.glb) files for the visualized 3D models are provided in the supplementary material.

4.2 Generator Self-Evolution

To assess whether construction experience can improve subsequent world generation, we partition the 135 learning descriptions from the SceneSmith and Function-Room datasets into eight sequential evolution rounds and repeat the full eight-round process independently five times with different random seeds. The generation models, tools, and per-world resource limits remain fixed as construction knowledge and agent harness policies evolve. Figure 5 tracks progress on these learning inputs; Section 4.3 tests transfer to unseen descriptions on the independent SceneEval-30 test set.

Reusable experience. Figure 5 shows improved textual and functional alignment alongside fewer spatial violations on the learning descriptions. Support and accessibility improve over successive rounds, whereas navigability remains near saturation. These joint gains suggest that accumulated experience helps coordinate requested content with spatial constraints, improving alignment without sacrificing spatial validity while the generation models remain fixed.

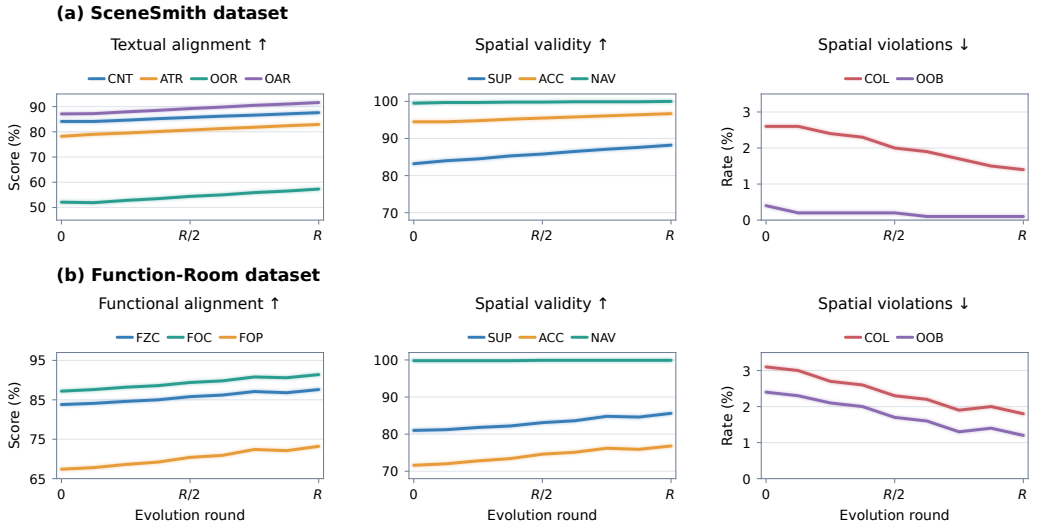


Figure 5: Generation quality on learning descriptions from the SceneSmith and Function-Room datasets.

4.3 Executable World Generation

We compare initial and evolved VibeArena with external baselines on scene quality, object fidelity, and executability, freezing cross-world experience during testing. These comparisons assess complete methods; Section 4.4 isolates VibeArena components.

Scene quality. VibeArena combines strong object–object and object–architecture relations with low collision rates even before evolution (Table 1). This joint advantage supports coordinated scene and object construction through the shared world program, preserving relational alignment without sacrificing spatial validity. Figure 6 shows qualitative comparisons on room requests from the SceneSmith, Function-Room, and SceneEval datasets.

Transfer of construction experience. The evolved generator better preserves requested relations while reducing spatial conflicts on unseen descriptions. These joint gains suggest that construction knowledge and harness policies improve subsequent world construction rather than merely retaining scene-specific fixes. Appendix D.6 reports paired changes and confidence intervals.

Object quality. VibeArena leads all five criteria in Table 2, combining prompt fidelity with plausible geometry and coherent surface detail. The matched-backend Articraft analysis in Appendix D.8

Table 1: Scene generation on 30 SceneEval descriptions (%). Dark to light green marks the top three distinct method means, respectively.

Method	CNT ↑	ATR ↑	OOR ↑	OAR ↑	SUP ↑	ACC ↑	NAV ↑	COL ↓	OOB ↓
HSM	53.3±9.3	58.1±14.7	22.3±12.3	50.0±10.4	70.9±7.2	85.8±4.8	99.1±1.0	18.4±6.8	7.4±3.6
LayoutVLM	64.2±8.0	31.1±13.7	27.2±12.1	53.1±14.8	25.7±7.7	93.3±5.2	100.0±0.0	21.1±10.0	5.9±3.5
SAGE	56.4±10.4	51.4±13.6	24.6±12.0	48.2±11.2	59.6±8.1	71.4±7.6	97.2±2.6	21.3±7.4	4.6±2.8
SceneWeaver	58.9±11.2	49.3±12.9	27.9±13.6	51.3±10.1	46.4±6.2	86.4±8.4	93.3±5.7	25.9±14.7	1.6±0.3
SceneSmith	78.8±6.7	71.6±9.4	37.2±13.3	72.4±10.3	66.7±4.0	52.2±8.7	97.8±2.1	18.1±4.8	0.6±0.7
SceneCode	79.4±6.0	74.0±11.7	32.6±12.6	61.1±12.6	44.3±12.8	74.2±9.1	100.0±0.0	11.3±4.3	0.4±0.9
Code as Room	57.2±16.1	59.8±15.5	16.7±13.2	46.9±10.3	27.3±13.7	56.1±6.7	99.9±0.1	28.9±4.3	0.7±0.5
FuncRoom-Agent*	82.4±9.5	76.8±11.3	38.1±15.1	74.0±13.9	72.8±8.3	93.5±10.6	100.0±0.0	10.7±7.8	0.3±0.2
GPT-6 Astra agent	72.4±8.1	77.2±11.4	32.8±13.6	65.4±11.8	62.1±7.9	70.6±8.8	98.4±1.6	19.2±5.8	2.1±1.4
DS-V4.1-Flash agent	67.6±9.2	70.0±12.6	29.4±13.1	60.8±12.4	57.8±8.6	66.2±9.1	97.9±2.0	22.4±6.4	2.6±1.7
VibeArena (initial)	83.8±7.1	78.6±10.2	51.4±12.6	86.8±9.4	83.6±5.4	94.2±6.8	99.6±0.4	2.8±2.2	0.3±0.3
VibeArena (evolved)	87.2±6.4	82.4±9.1	56.8±11.8	91.2±8.6	87.8±4.8	96.4±5.8	100.0±0.0	1.6±1.4	0.1±0.1

*Code not publicly released.

Table 2: Object quality in generated scenes.

Method	Align.↑	Plaus.↑	T-G.↑	Text.↑	Geo.↑
SceneCode	1044.5	939.4	867.4	1186.1	952.8
Code as Room	890.3	832.9	789.0	1106.2	905.8
GPT-6 Astra agent	1025.2	1033.7	1036.2	1037.2	1046.3
DS-V4.1-Flash agent	805.8	863.0	810.5	778.5	825.9
VibeArena	**1192.6**	**1170.7**	**1219.5**	**1250.6**	**1170.1**

Table 3: Physical stability. Rates are percentages; average shift is in centimeters.

Method	Stab.↑	>0.1 m↓	>0.01 m↓	Avg. shift (cm)↓
SceneSmith	99.1	2.7	2.7	0.4900
SAGE	96.7	3.3	4.9	0.3572
SceneCode	97.2	1.6	2.4	0.2772
VibeArena	**100.0**	**0.0**	**1.5**	**0.1180**

further shows that scene context and construction knowledge improve object quality and operation success under the same modeling backend.

Physical stability. To test passive physics under matched simulation conditions, Table 3 compares VibeArena with SceneSmith, SAGE, and SceneCode, whose outputs provide the physical or articulation information needed for the shared gravity test. VibeArena leads all four measures, indicating that its assembled worlds better preserve object poses under gravity and avoid large shifts before interaction.

4.4 Ablation and Analysis

Table 4 reports the effects of construction components, self-evolution sources, experience representation, and experience consolidation. Removing hierarchical IRs most strongly harms relation-sensitive scene metrics, consistent with the role of staged planning in preserving spatial constraints. Removing execution-guided revision causes the largest loss in interaction success and a marked increase in collisions; scene context and realization mappings also matter for interaction, as they connect object construction and failure diagnosis to the surroundings.

Table 4: Construction, self-evolution, knowledge, and experience-consolidation ablations. All values are percentages.

Variant	OOR↑	OAR↑	ACC↑	COL↓	SR↑
Construction components					
VibeArena (initial)	51.4	86.8	94.2	2.8	67.2
W/o hierarchical IRs	38.6	74.2	88.4	6.4	54.4
W/o scene context	42.8	82.6	86.1	5.2	58.4
W/o realization maps	46.2	84.1	89.4	4.6	60.0
W/o execution revision	47.8	85.2	84.6	8.8	51.2
Sources of self-evolution					
VibeArena (initial)	51.4	86.8	94.2	2.8	67.2
Knowledge only	54.2	89.6	94.8	2.4	69.6
Harness only	52.8	87.9	95.7	2.0	71.2
Full evolution	56.8	91.2	96.4	1.6	73.2
Experience representation					
VibeArena (initial)	51.4	86.8	94.2	2.8	67.2
Retrieved repair examples	52.6	87.8	94.6	2.6	68.4
Plain repair summaries	53.6	88.6	94.8	2.4	69.6
Conditional knowledge	54.9	89.8	95.2	2.1	71.6
Experience consolidation					
VibeArena (evolved)	56.8	91.2	96.4	1.6	73.2
W/o individual updates	55.9	90.5	95.9	1.8	72.1
W/o cross-scene confirmation	55.4	90.1	95.6	2.0	71.5
W/o retirement	56.2	90.8	96.1	1.7	72.6

Harness updates improve accessibility, collision rate, and diagnostic task success, while knowledge updates yield larger gains in spatial relations. Across the five evolution runs, the harness accepts 12 of 30 candidate policies, primarily affecting validation order and repair routing; Appendix Table 8 summarizes their use and outcomes. Full evolution leads across

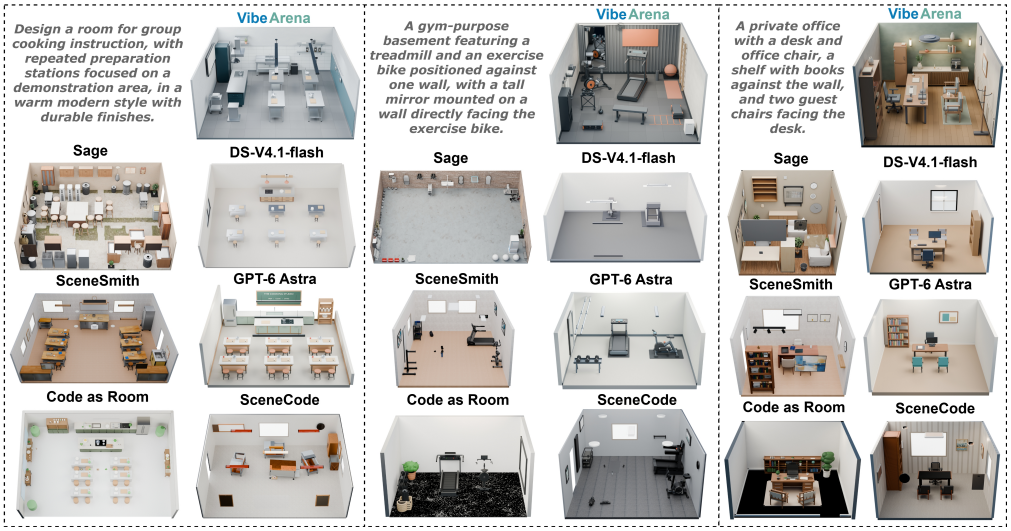


Figure 6: Qualitative scene comparisons for three room requests.

the reported metrics, supporting their complementary roles. The experience-representation ablation evaluates the downstream utility of explicit applicability conditions under matched retrieval and a fixed harness. Plain repair summaries improve on retrieved examples. Under matched retrieval and context budgets, conditional knowledge achieves the highest scores among the compared experience representations. The consolidation ablation shows that cross-scene confirmation helps prevent scene-specific fixes from becoming general knowledge, while individual updates and retirement support incremental adaptation and removal of stale experience.

4.5 Downstream Embodied Task Evaluation

Embodied task execution. We test pick-and-place, opening/retrieval, and mobile manipulation in worlds. A shared controller runs ten seeded trials for each of 30 SceneEval requests in Isaac Gym; Appendix D.5 gives the protocol. Figure 7 shows a separate Genesis trace replayed in Blender.

Table 5: Task success (%).

Generator	Pick-place ↑	Open/retr. ↑	Mobile ↑
SceneSmith	34.0	22.3	27.3
SAGE	47.0	31.3	40.7
SceneCode	37.0	31.3	29.0
VibeArena (initial)	50.3	40.0	43.0
VibeArena (evolved)	**60.7**	**53.0**	**51.0**



Figure 7: Example Panda cup manipulation in a generated world.

VibeArena achieves higher success rates across the three task families (Table 5). Figure 7 illustrates one execution, while Table 4 examines the effects of construction and revision components.

5 Conclusion

VibeArena builds executable 3D worlds by linking scene layout, articulated objects, physical properties, and tasks in a shared program. Execution feedback guides world repair and yields reusable construction knowledge and agent harness policies. Evolution improves scene quality on SceneEval and embodied task success. Tests assess object quality and stability, while ablations show the contributions of hierarchical construction, execution-guided revision, and reusable experience.

Limitations and future directions. VibeArena takes a step toward world generators that accumulate and reuse construction experience. Building on its local rules and templates, future work can explore compositional knowledge across mechanisms and construction routines learned from longer interaction sequences.

AI USE STATEMENT

Generative AI assisted with language editing, LaTeX preparation, literature collection, and discussion of the manuscript's content. The authors designed the method and experiments, ran the evaluations, verified all measurements and claims, and take responsibility for the manuscript.

ETHICS STATEMENT

VibeArena generates diverse worlds and evaluates robot behavior in simulation. Generated scenes may contain unsafe layouts or physical configurations and require validation before use with real robots. Third-party datasets and assets will be shared only where their licenses permit.

REPRODUCIBILITY STATEMENT

Section 3 describes world construction and self-evolution; Appendix A provides implementation details. Appendix B summarizes the core agent instructions and harness policy proposal and validation procedure. Appendix D specifies datasets, baselines, metrics, trial design, and success criteria. Scenes in GLB for the visualized worlds are provided in the supplementary material. The three datasets used in this paper—SceneSmith, Function-Room, and SceneEval—are all publicly available.

REFERENCES

Ziang Cao, Zhaoxi Chen, Liang Pan, and Ziwei Liu. Physx-3d: Physical-grounded 3d asset generation. *Advances in Neural Information Processing Systems*, 38:93771–93784, 2026a.

Ziang Cao, Fangzhou Hong, Zhaoxi Chen, Liang Pan, and Ziwei Liu. Physx-anything: Simulation-ready physical 3d assets from single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5839–5848, 2026b.

Guanxiong Chen, Qianjun Xia, Jiawei Peng, Heng Zhang, Bole Ma, Justin Qian, Ziyi Jiao, Bingyang Zhou, Luoxin Ye, Kaifeng Zhang, et al. Agentic real2sim: Physics-based world modeling with vision-language agents. *arXiv preprint arXiv:2607.19190*, 2026.

Minghao Chen, Roman Shapovalov, Iro Laina, Tom Monnier, Jianyuan Wang, David Novotny, and Andrea Vedaldi. Partgen: Part-level 3d generation and reconstruction with multi-view diffusion models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5881–5892. IEEE, 2025.

Qinchuan Cheng, Zhantao Gong, Pengzhan Sun, Angela Yao, Xulei Yang, and Shijie Li. Ego2world: Compiling egocentric cooking videos into executable worlds for belief-state planning. *arXiv preprint arXiv:2605.13335*, 2026.

Bingquan Dai, Luo Li, Qihong Tang, Jie Wang, Xinyu Lian, Hao Xu, Minghan Qin, Xudong Xu, Bo Dai, Haoqian Wang, et al. Meshcoder: Llm-powered structured mesh code generation from point clouds. *Advances in Neural Information Processing Systems*, 38:8917–8954, 2026.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

Maxence Faldor, Jenny Zhang, Antoine Cully, and Jeff Clune. Omni-epic: Open-endedness via models of human notions of interestingness with environments programmed in code. In *International Conference on Learning Representations*, volume 2025, pp. 85260–85385, 2025.

Hao Feng, Zhi Zuo, MingJian Liang, Jingyu Hu, Xiaowei Hu, Liupengfei Wu, Dian Zhang, Guoxin Fang, and Zhengzhe Liu. Funcroom-agent: Sequential feed-forward 3d functional indoor scene generation. *arXiv preprint arXiv:2608.29519*, 2026.

Genesis AI Team. The role of simulation in scalable robotics, Genesis World 1.0, and the path forward. *Genesis AI Blog*, May 2026.

R Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J Mitra, and Daniel Ritchie. Shapeassembly: Learning to generate programs for 3d shape structure synthesis. *ACM Transactions on Graphics (TOG)*, 39(6):1–20, 2020.

Haoqiang Kang, Xiaokang Ye, Yuhan Liu, Siddhant Hitesh Mantri, Lingjun Mao, James Fleming, Drishti Regmi, and Lianhui Qin. Simworld studio: Automatic environment generation with evolving coding agent for embodied agent learning. *arXiv preprint arXiv:2605.09423*, 2026.

Long Le, Jason Xie, William Liang, Hung-Ju Wang, Yue Yang, Yecheng Jason Ma, Kyle Vedder, Arjun Krishna, Dinesh Jayaraman, and Eric Eaton. Articulate-anything: Automatic modeling of articulated objects via a vision-language foundation model. In *International Conference on Learning Representations*, volume 2025, pp. 17578–17602, 2025.

Wolfgang Lehrach, Daniel Hennes, Miguel Lazaro-Gredilla, Xinghua Lou, Carter Wendelken, Zun Li, Antoine Dedieu, Marc Lanctot, Atil Iscen, John Schultz, et al. Code world models for general game playing. In *International Conference on Learning Representations*, volume 2026, pp. 133870–133927, 2026.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pp. 80–93. PMLR, 2023.

Zhiqi Li, Yuxuan Liao, and Bo Zhu. Recursive code world models: Building complex worlds through recursive scene programs. *arXiv preprint arXiv:2609.11499*, 2026.

William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. Eurekaverse: Environment curriculum generation via large language models. *arXiv preprint arXiv:2411.01775*, 2024.

Yuchen Lin, Chenguo Lin, Panwang Pan, Honglei Yan, Feng Yiqiang, Yadong Mu, and Katerina Fragkiadaki. Partcrafter: Structured 3d mesh generation via compositional latent diffusion transformers. *Advances in Neural Information Processing Systems*, 38:35387–35415, 2026.

Bo Liu, Simon Yu, Yiding Jiang, Ao Qu, Andrew Zhao, Zichen Liu, Junsu Kim, Zijian Zhou, Seungone Kim, Tongzheng Ren, et al. Spade: Self-play in adaptive synthetic executable environments. *arXiv preprint arXiv:2608.19197*, 2026.

Soroush Nasiriany, Sep Nasiriany, Abhiram Maddukuri, and Yuke Zhu. Robocasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *International Conference on Learning Representations*, volume 2026, pp. 98643–98667, 2026.

Yansong Ning, Jingwen Ye, Zhongkai Wu, Yang Sun, Yiqin Zhu, Xingyi Li, Weidong Zhang, and Hao Liu. Vibeworlding: Can multimodal agents construct 3d open worlds end-to-end? *arXiv preprint arXiv:2608.15265*, 2026.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*, pp. 17473–17498. PMLR, 2022.

Nicholas Pfaff, Thomas Cohn, Sergey Zakharov, Rick Cory, and Russ Tedrake. Scenesmith: Agentic generation of simulation-ready indoor scenes. In *Forty-third International Conference on Machine Learning*, 2026.

Hou In Derek Pun, Hou In Ivan Tam, Austin T Wang, Xiaoliang Huo, Angel X Chang, and Manolis Savva. HSM: Hierarchical scene motifs for multi-scale indoor scene generation. In *2026 International Conference on 3D Vision (3DV)*, pp. 1356–1367. IEEE, 2026.

Alexander Raistrick, Lingjie Mei, Karhan Kayan, David Yan, Yiming Zuo, Beining Han, Hongyu Wen, Meenal Parakh, Stamatis Alexandropoulos, Lahav Lipson, et al. Infinigen indoors: Photorealistic indoor scenes using procedural generation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21783–21794. IEEE, 2024.

Xingjian Ran, Shujie Zhang, Weipeng Zhong, Li Luo, and Bo Dai. Pair2scene: Learning local object relations for procedural scene generation. *arXiv preprint arXiv:2604.11808*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Fan-Yun Sun, Weiyu Liu, Siyi Gu, Dylan Lim, Goutam Bhat, Federico Tombari, Manling Li, Nick Haber, and Jiajun Wu. LayoutVLM: Differentiable optimization of 3D layout via vision-language models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 29469–29478. IEEE, 2025.

Hou In Ivan Tam, Hou In Derek Pun, Austin T Wang, Angel X Chang, and Manolis Savva. SceneEval: Evaluating semantic coherence in text-conditioned 3D indoor scene synthesis. In *2026 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 7355–7365. IEEE, 2026.

Yolo Y Tang, Daiki Shimada, Jiayue Meng, Jing Bi, Pinxin Liu, Yicheng Wang, Yunzhong Xiao, Zhangyun Tan, Zeliang Zhang, Chao Huang, et al. Bvb: Benchmarking agentic video understanding via programmatic reconstruction in blender. *arXiv preprint arXiv:2609.15478*, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a.

Hanyang Wang, Yimo Cai, Weiliang Chen, Jiawei Chi, Haowen Sun, Qiyu Dai, Yi-Hsin Hung, Xingzhuo Guo, Jinshan Ren, Runmao Yao, et al. Code as worlds: Agentic discovery of executable world representations for physical reasoning. *arXiv preprint arXiv:2608.27549*, 2026a.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. Gensim: Generating robotic simulation tasks via large language models. In *International Conference on Learning Representations*, volume 2024, pp. 4890–4924, 2024.

Puyi Wang, Yuhao Wang, Linjie Li, Zhengyuan Yang, Kevin Qinghong Lin, Yangguang Li, and Yu Cheng. Scenecode: Executable world programs for editable indoor scenes with articulated objects. *arXiv preprint arXiv:2605.19587*, 2026b.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

Xinjie Wang, Liu Liu, Taojun Ding, Andrew Choi, Chaodong Huang, Mengao Zhao, Ziang Li, Jackson Jiang, Chunlei Yu, Shengxiang Liu, et al. Embodiedgen v2: An agentic, simulation-ready 3d world engine for embodied ai. *arXiv preprint arXiv:2607.07459*, 2026c.

Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. *arXiv preprint arXiv:2311.01455*, 2023b.

Zefeng Wang, Minxi Yan, Jinhe Bi, Sikuan Yan, Volker Tresp, and Yunpu Ma. Metaskill-evolve: Recursive self-improvement of llm agents via two-timescale meta-skill evolution. *arXiv preprint arXiv:2607.05297*, 2026d.

Tianxin Wei, Zhan Shi, Minhua Lin, Bing He, Zewen Liu, Yisi Sang, Yuanchen Bei, Xuying Ning, Jiaru Zou, Ting-Wei Li, et al. Evo-harness: Context-to-harness skill compilation for self-evolving agents. *arXiv preprint arXiv:2608.15071*, 2026.

Tong Wu, Guandao Yang, Zhibing Li, Kai Zhang, Ziwei Liu, Leonidas Guibas, Dahua Lin, and Gordon Wetzstein. GPT-4V(ision) is a human-aligned evaluator for text-to-3D generation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22227–22238. IEEE, 2024.

Zhuangzhe Wu, Yue Xin, Chengkai Hou, Minghao Chen, Yaoxu Lyu, Jieyu Zhang, and Shanghang Zhang. Urdf-anything+: End-to-end generation for simulation-ready articulated assets. *arXiv preprint arXiv:2603.14010*, 2026.

Hongchi Xia, Xuan Li, Zhaoshuo Li, Qianli Ma, Jiashu Xu, Ming-Yu Liu, Yin Cui, Tsung-Yi Lin, Wei-Chiu Ma, Shenlong Wang, et al. Sage: Scalable agentic 3d scene generation for embodied ai. *arXiv preprint arXiv:2602.10116*, 2026a.

Peng Xia, Rujun Han, Zifeng Wang, Yanfei Chen, Yufan Zhuang, Yoonho Lee, Chengsong Huang, Han Yu, Zhongying CuiZhu, Yifei Ming, Huaxiu Yao, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. RRSI: Regularized recursive self-improvement of agent harnesses. *arXiv preprint arXiv:2609.24972*, 2026b.

Jianfeng Xiang, Xiaoxue Chen, Sicheng Xu, Ruicheng Wang, Zelong Lv, Yu Deng, Hongyuan Zhu, Yue Dong, Hao Zhao, Nicholas Jing Yuan, et al. Native and compact structured latents for 3d generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14419–14429, 2026.

Zijian Xiao, Zipeng Ye, Jinkun Hao, Xiong Yang, Yuchen Xie, and Ran Yi. Beyond placement and articulation: Usage-driven code scenes for embodied interaction. *arXiv preprint arXiv:2608.18840*, 2026.

Anyi Xu, B Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, CC Yu, et al. Deepseek-v4. 1-flash: Pushing the limits of kv cache compression. *arXiv preprint arXiv:2609.19969*, 2026.

Yandan Yang, Baoxiong Jia, Shujie Zhang, and Siyuan Huang. Sceneweaver: All-in-one 3d scene synthesis with an extensible and self-reflective agent. *Advances in neural information processing systems*, 38:140319–140351, 2026a.

Yixuan Yang, Zhen Luo, Wanshui Gan, Jinkun Hao, Junru Lu, Jinghao Yan, Zhaoyang Lyu, and Xudong Xu. Code-as-room: Generating 3d rooms from top-down view images via agentic code synthesis. *arXiv preprint arXiv:2605.18451*, 2026b.

Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language guided generation of 3d embodied ai environments. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16277–16287. IEEE, 2024.

Yunhan Yang, Yuanchen Guo, Yukun Huang, Zi-Xin Zou, Zhipeng Yu, Yangguang Li, Yanpei Cao, and Xihui Liu. Holopart: Generative 3d part amodal segmentation. In *International Conference on Learning Representations*, volume 2026, pp. 52657–52678, 2026c.

Zhaochen Yu, Yingcheng Wu, Zhenfei Yin, Kaiyuan Chen, Zhe Zhao, Mengdi Wang, Shuicheng Yan, and Ling Yang. Recursive experiential-working memory evolution for long-horizon agent harnesses. *arXiv preprint arXiv:2608.24876*, 2026.

Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. Envgen: Generating and adapting environments via llms for training embodied agents. *arXiv preprint arXiv:2403.12014*, 2024.

Tong Zheng, Xidong Wu, Zheng Zhang, Zhankui He, Chaoyi Zhang, Benjamin Coleman, Ruoqiao Wei, Di Bai, Haolin Liu, Rui Liu, et al. Dream-rsi: Recursive self-improvement through evolving worlds. *arXiv preprint arXiv:2609.14858*, 2026.

Matt Zhou, Ruining Li, Xiaoyang Lyu, Zhaomou Song, Zhening Huang, Chuanxia Zheng, Christian Rupprecht, Andrea Vedaldi, and Shangzhe Wu. Articraft: An agentic system for scalable articulated 3d asset generation. *arXiv preprint arXiv:2605.15187*, 2026.

A Supplementary Method Details

A.1 World Representation and Construction

World program and runtime state. VibeArena maintains ArenaProgram, the editable shared world program P_t, together with a simulator state and persistent construction records. The program links scene layout, object geometry and articulation, physical properties, operations, and tasks. Capability profiles, interaction challenges, world intent, and layout plans provide the context from which the program is constructed. Compiled assets and simulator inputs are derived artifacts.

Robot execution changes the simulator state without necessarily changing the program. Opening a door updates its runtime joint state; revising the door's joint specification changes the world program. Accepted program patches preserve protected interaction requirements and task semantics. Derived assets are regenerated or reused according to their dependencies. Accepted programs, patches, and validation evidence are stored together so that later execution starts from a consistent construction state.

Realization mappings. The capability planner produces interaction challenges and world intent jointly. Challenges describe the capabilities to exercise, while world intent specifies the contents and conditions needed to support them. The layout planner resolves these conditions into spatial organization, functional groups, asset requirements, and local clearances. Scoped synthesis subsequently constructs scene, object, and physical specifications.

Realization mappings connect requirements, declared construction decisions, and executable entities. Each decision has a stable identifier and references the requirements it serves, associated objects or parts, and relevant construction layers. Task and operation references connect these entities to executable interactions.

A retrieval requirement may motivate separate decisions about an opening mechanism, internal support, and approach space. Conversely, one object or part may participate in several tasks. This representation allows diagnosis to recover both the affected entity and the interaction requirement behind its construction.

References explicitly present in the program are extracted deterministically. Additional declared mappings are checked for valid requirement, entity, and part identities. Missing relationships remain unresolved rather than being linked to all available planning context.

Build dependencies and asset reuse. A realization mapping records the role of a structure in satisfying a requirement. A build dependency records an input consumed during asset generation or compilation. Sharing a requirement does not by itself make two assets dependent on one another for generation.

Build dependencies include object specifications, relevant physical configurations, and cross-object interfaces. Placement-only relationships are handled separately from asset-generation inputs. A placement edit can therefore reuse unchanged object geometry, whereas a change to parts or interfaces can require asset regeneration and renewed physical grounding.

Dependency analysis compares the original and revised programs, including added and removed entities and changes to shared interfaces. Reuse requires matching input, configuration, tool, and dependency identities. Incomplete dependency information triggers conservative rebuilding.

Asset reuse does not reduce candidate validation coverage. Each candidate undergoes the complete configured execution, visual, and physical validation procedure.

Code-based object generation and physical grounding. Object synthesis generates modeling code that realizes object specifications as geometry, materials, semantic parts, and articulated mechanisms. Compiler feedback and rendered observations guide bounded revisions of the generated assets.

Physical configuration associates contact parameters and joint dynamics with the generated structure. Collision geometry grounds the distribution of link masses, centers of mass, and inertia. The grounded assets and world specifications are then compiled into simulator inputs.

The program retains entity and part identities across this process. These identities connect the synthesized structures to task references and allow execution evidence to be associated with the construction decisions that produced them.

A.2 Diagnosis and World Revision

Execution evidence and diagnostic records. Execution produces a versioned evidence bundle containing task traces, runtime states, visual observations, geometry checks, and physical-binding reports. Detected gaps reference the affected requirements, decisions, entities, and parts wherever those relationships are available.

Each repair issue maintains competing explanations at the implementation, object, scene, and physics layers. An explanation records the proposed failure mechanism and intervention layer, supporting and contradicting evidence, missing observations, and an inspection or probe that could distinguish it from alternatives. Its status is *undetermined*, *supported*, or *refuted*.

Model-generated explanations remain hypotheses until supported by execution evidence; ambiguous observations preserve multiple alternatives.

Diagnostic records are bound to the world-program version and grounded asset identities; after a relevant construction change, diagnosis uses evidence from the new state.

Static and runtime inspection. Diagnosis first uses existing verifier outputs and observations. An explicit specification violation can directly support a corresponding intervention.

When no executable world is available, static inspection examines the world program, generated source, interface constraints, and existing geometry reports. With an instantiated world, read-only inspection can query entity poses, joint states, contacts, asset bounds, and physical bindings.

Unavailable model or rendering services block the affected operation.

Controlled probes. A controlled probe specifies the hypothesis under examination, one intervention factor, and an expected observation. The implemented interventions temporarily change an entity pose or an object's joint state.

Control and treatment executions restore the same runtime snapshot for each task, seed, and rollout. The original runtime state is restored after the probe, including when an exception occurs. The diagnostic intervention therefore remains separate from an accepted world revision.

At least two matched pairs with consistent outcome direction and verifiable program and asset identities are required to support or refute the tested intervention direction. Insufficient or conflicting observations leave the explanation undetermined.

Probe traces remain diagnostic records and are excluded from formal task-success evidence.

Revision candidates across construction layers. Revision can operate at four construction levels. Implementation repair changes generated asset code while preserving the declared object specification and interfaces. Object-level revision changes the specified structure. Scene-level revision changes placements or spatial relations. Physics-level revision changes permitted physical parameters.

The intervention level is selected according to evidence. A scene-level obstruction can motivate a placement candidate directly. Repeated implementation failures motivate further diagnosis unless evidence supports a broader intervention.

An undetermined explanation may motivate an experimental candidate with an expected observation.

A single world reasoner generates formal revision patches from the accepted program, diagnostic hypotheses, dependency context, attempted interventions, and remaining budget. Candidates compared in one round share the same parent program:

$$P_t^{(j)} = \mathrm{Apply}(P_t, \Delta_t^{(j)}). \tag{3}$$

The verifier checks changed fields and entities against the permitted intervention scope, including justified dependencies.

Requirement-grounded repair may add missing interaction structure but must preserve task objectives, success predicates, and protected safety conditions.

Candidate validation and selection. Each candidate is compiled and executed independently against the accepted parent world. Acceptance requires valid construction and execution, preserved task intent and safety, resolution of the target issue, and no new detected gaps or protected-task regressions. Among eligible candidates, selection minimizes changed fields and rebuilding scope, using construction-rule verdicts to break ties. Only the selected, validated patch is committed; otherwise the parent world remains unchanged.

Diagnosis and search budget. A persistent work budget and per-issue retry limits bound diagnosis and revision, including failed attempts. Search stops after resolution, budget exhaustion, repeated interventions, or blocked progress; token usage and elapsed time are recorded separately.

A.3 Construction Knowledge: Representation and Update

Successful builds and revisions reveal reusable construction conditions. VibeArena represents construction knowledge as successful construction templates and rules with explicit applicability conditions. A successful world can provide a template, whereas rule induction requires an accepted before/after revision pair, relevant diagnosis, and complete successful validation. This preserves the link between a construction change and its measured outcome.

Rule representation and induction. A rule binds a mechanism and a related object through semantic roles, joint type and axis, explicit relations, and requirement references. Each rule binds one unambiguous revolute or prismatic mechanism to its related object. It measures the related object's displacement in the joint's local frame normalized by mechanism scale, relative size, joint range, and the required joint target fraction when explicitly available. This removes dependence on world-specific entity names and absolute coordinates while retaining structural context. Ambiguous joints, missing dimensions, or incompatible requirement bindings produce an unknown result.

A rule combines interval predicates over these features into alternative admissible clauses and explicit exclusions. Its versioned record retains source programs, requirements, changed features, supporting validations, counterexamples, and uncertain outcomes. Induction is deterministic: it selects measured features changed by the accepted revision and places interval boundaries between their old and validated values, retaining the other measured conditions. The resulting rule remains a candidate empirical condition. For illustration, a repaired cabinet with a formerly blocked door can yield a candidate condition on the neighboring object's joint-local clearance; a later cabinet must match the mechanism and spatial roles before the condition is considered.

Testing and revising the conditions. During exploration, the system proposes bounded changes to measured factors near a rule's boundary. A heuristic prioritizes nearby bounds and shared requirements; a random-order alternative uses the same proposal pool and budget. Each proposal enters the existing world-revision path. The sole patch producer generates the edit, and scope checks enforce the declared factor change while protecting unrelated construction fields. Structural edits regenerate affected assets before full validation.

Independent evidence determines the rule update. A formally feasible instance outside the current conditions adds an alternative clause. A compiler-supported violation adds an exclusion at the measured configuration. Contradictory feasibility evidence can withdraw the rule; rule testing refines its recorded validity scope.

Reuse in subsequent worlds. Before concrete objects exist, retrieved rules provide provisional role guidance to layout and staged synthesis. Once the program is available, deterministic binding checks structural compatibility and evaluates each predicate as satisfied, violated, or unknown. Compatible conditions enter object generation and repair context. Missing or conflicting evidence causes abstention. Rule verdicts never replace formal validation: among otherwise equivalent eli-

gible repair candidates, fewer rule violations provide only a final preference after modification and rebuilding costs. New accepted revisions can in turn supply additional construction knowledge.

Successful construction templates. Experience is extracted only from worlds with complete successful task evidence and safety, visual, and physical validation. Objects with explicit requirement mappings are eligible for experience extraction.

A construction record contains applicable requirement categories and structural conditions, recorded construction problems where available, a validated construction template, and verification provenance. Templates retain dimensionless part proportions, joint types, and selected relative spatial relationships. World-specific entity identifiers and absolute coordinates are excluded from the reusable template, while source information remains in its provenance.

Extraction is deterministic. Duplicate templates share supporting provenance, while distinct templates remain separate conditional alternatives. Each template records a construction validated under its source conditions.

Template retrieval. For object generation or repair in a subsequent world, deterministic retrieval matches semantic tags, part counts, and joint types. At most three relevant records enter the existing generation context.

Current object requirements take precedence over retrieved examples. The generated result contributes additional experience after passing the same construction and execution checks.

Template experience enters object synthesis and revision. It changes the contextual guidance available to the generator while leaving the high-level planning procedure, DSL, intermediate representations, and generation-model parameters fixed.

A.4 Agent Harness Policies: Proposal and Evaluation

Repeated construction trajectories also reveal when the generation process performs ineffective work. VibeArena retains this experience as agent harness policies, separately from construction knowledge. An agent harness policy maps a bounded context (stage, issue class, available evidence, previous failed attempts, and remaining budget) to an existing scheduling action. Actions can reorder mandatory checks, invoke the deterministic layout checker before asset generation, request an existing inspection, prioritize a supported repair layer, limit local retries, or stop unresolved work. Policies change scheduling only; requirements and validators remain fixed.

Proposing scheduling changes. An adaptive run records controller outcomes, formal world-validation results, and costs, including failed calls. The deterministic proposer derives trigger thresholds and retry caps from effective repair sequences in at least two distinct, formally successful exploration inputs. For example, repeated geometry failures resolved through supported scene revisions can motivate checking layout earlier.

Independent paired validation. Policy activation uses a matched comparison against the current schedule on two to four held-out learning inputs, with fixed tasks, validators, construction knowledge, and budgets. Both schedules must pass formal validation on every input. For the measured cost vector $c = (n_{\text{calls}}, n_{\text{tokens}}, t)$, acceptance requires

$$c_{i,k}^{\text{candidate}} \leq c_{i,k}^{\text{current}} \quad \text{for every input } i \text{ and cost component } k, \tag{4}$$

with a strict improvement in at least one component on one input. Accepted policy versions can govern later worlds, but remain fixed within each world-construction run. The harness-only row of Table 4 separately measures its effect on evaluation inputs with policy updates disabled during testing.

A.5 Experience Persistence and Evaluation

Construction knowledge and agent harness policies have separate histories, versions, and immutable snapshots. Exploration can accumulate construction rules, perform bounded boundary tests, and propose agent harness policies. Policy activation uses the independent paired procedure above.

Evaluation uses frozen experience snapshots; spatial challenge adjustment is used only during exploration.

Within exploration, an optional spatial challenge mechanism can revisit previously validated configurations after repeated task outcomes. Self-evolution changes world programs, construction knowledge, and harness policies while keeping generation-model parameters and the embodied controller fixed.

Spatial configuration revisits. After repeated task outcomes, challenge adjustment can select another previously feasible spatial configuration. The selected placements preserve task semantics, object structures, and physical parameters and pass the standard patch and validation pipeline.

B Core Agent Prompts

VibeArena uses separate instructions for planning, staged world construction, object authoring, and execution-guided revision. The excerpts below show the main responsibilities of these agents. World-specific requirements, construction experience, and execution evidence are supplied as runtime context. Harness policy generation is described at the end because its default proposer is deterministic rather than prompt-based.

Challenge and world intent. The planning agent jointly specifies an interaction challenge and a world intent. The challenge identifies the capability to exercise; the intent specifies the contents and conditions needed for the requested activities. The system instruction begins:

> "Jointly produce one CapabilityChallenge and its dependent WorldIntent."

The remaining instruction asks the agent to select a challenge at the supplied capability frontier and derive a world that supports it, without producing the layout or executable program at this stage.

Spatial layout. The layout agent plans where interaction can take place before objects are constructed:

> "Plan spatial organization only: extent, functional zones, connectivity, circulation, support surfaces, large-object ordering, interaction workspace, articulated swept-volume clearance, and robot approach space."

Staged world construction. The scene, object, and physics agents fill different parts of ArenaProgram. Their respective instructions constrain the scene to spatial structure, the object stage to objects and placements, and the physics stage to physical specifications for the supplied objects. The scene stage defines regions and free space; the object stage constructs and places assets; and the physics stage assigns rigid-body, contact, and joint-level physical properties.

Articulated asset authoring and repair. VibeArena wraps the Articraft SDK and compiler for object authoring and validation. ObjectSpec provides the object requirements and any already grounded parts and joints. When articulation is pending, the VibeArena object agent resolves the concrete mechanism using Articraft modeling primitives; once grounded, it preserves the realized topology and interfaces. The object-authoring instruction states:

> "Use the supplied part IDs as the generated link names and use stable semantic joint IDs."

For failed builds, compiler and quality-control findings guide repair:

> "First re-evaluate the mechanism against compiler/QC evidence."

Repeated geometry failures can lead to redesign of the failing subassembly while the grounded interface is preserved.

Execution-guided world revision. The world reasoner receives diagnostic evidence and proposes a scoped action. Its instruction distinguishes program edits from inspection and runtime probes:

> "CONSTRUCT/EXPAND/MODIFY are editing actions. INSPECT requires InspectionPlan and changes no program. PROBE requires ProbePlan and creates only ephemeral runtime intervention."

Proposed patches are compiled and evaluated before acceptance.

Harness policy proposal and validation. The default harness proposer uses recorded exploration trajectories rather than an agent prompt. It groups effective repair sequences by construction stage and issue class, requiring support from at least two distinct successful exploration inputs. A candidate records its trigger conditions, including evidence, prior failed attempts, and remaining budget, together with an existing scheduling action and retry cap. Available actions include reordering checks, requesting inspection, prioritizing a repair layer, and limiting local retries. A separate paired comparison on two to four inputs tests the candidate against the current schedule. Both schedules must pass the same validation; the candidate is accepted only when model calls, tokens, and elapsed time do not increase on any comparison input and at least one cost decreases. Policies do not change world requirements, validators, acceptance thresholds, or program contents.

C Comparison with Related World-Generation Systems

Table 6 compares the mechanisms emphasized in recent systems for constructing and improving executable or interactive worlds; the corresponding papers are cited in Section 2. The entries describe reported mechanisms, rather than asserting that a system lacks an unlisted capability.

Table 6: Construction, feedback, and reusable experience in related world-generation systems.

System	Reported construction and experience mechanisms
SceneSmith	Hierarchical scene generation combines a designer, critic, physical-property estimation, and mixed asset synthesis and retrieval. Agent memory supports iterative refinement within construction stages.
SceneCode	Executable object programs undergo execution-guided repair. A registry links requests, programs, geometry, and simulation assets for traceable local editing.
EmbodiedGen V2	Simulation-ready assets, affordance annotations, and task-driven assembly share a world representation that preserves state across physics-validated edits.
VibeWorlding	Multimodal agents use retrieval and editing tools with feasibility and intent verification. Agent capabilities improve through supervised and reinforcement-learning post-training.
SimWorld Studio	UE5 code generation uses compiler, physics, and visual verification. Verifier feedback adds reusable tools and procedural skills; learner feedback guides curricula.
VibeArena	ArenaProgram links task requirements, object mechanisms, and scene relations. Execution evidence guides targeted revisions; validated repairs yield conditional construction knowledge, while harness policies adjust inspection and repair scheduling.

SimWorld Studio also learns across generation episodes, so the distinction is in the form of reusable experience: VibeArena stores measured applicability conditions over mechanism–scene relations and updates the construction harness.

D Supplementary Experimental Details

D.1 Implementation Details

Scene-generation baselines include HSM, LayoutVLM, SceneWeaver, SceneSmith, SceneCode, Code as Room, SAGE, and FuncRoom-Agent; each evaluation uses its relevant comparison set. The general-purpose agents use GPT-6 Astra and DeepSeek-V4.1-Flash (DS-V4.1-Flash) with the same modeling tools, authoring resources, and resource limits as VibeArena, but receive direct tool-use prompts without the specialized construction pipeline or cross-world memory. Unless otherwise

specified, all agentic components in VibeArena use GPT-6 Astra with reasoning effort set to medium via the OpenAI API. External baselines retain their published model configurations; controlled agent comparisons and ablations match tools and budgets, and ablations keep the backbone fixed. SceneEval reports nine metrics for prompt consistency and spatial validity, while functional metrics on the Function-Room dataset track evolution. FZC measures coverage of required functional zones, FOC coverage of functional-object slots, and FOP the fraction of generated objects that are valid and relevant under the prompt-specific annotation. A separate interaction suite specifies opening, retrieval, and placement tasks before generation. SceneEval-30 test scores are not used for experience updates or checkpoint selection. The reference environment specifies Python 3.10, PyTorch 2.8, CUDA 12.6, Genesis 1.3.3, and Articraft. The implementation stores run contracts with inputs, seeds, model settings, budgets, and snapshot identities, together with versioned programs, model-call reports, and execution traces.

D.2 Baseline Result Sources and Evaluation Protocols

We report the baselines on SceneEval-30 using SceneCode's main-table protocol. Following SceneSmith, our uncertainty terms are 95% confidence-interval half-widths computed using the two-sided Student t critical value with one fewer degree of freedom than the scene count, multiplied by the sample standard deviation of scene-level scores and divided by the square root of the scene count (30 for SceneEval-30). The intervals describe uncertainty in the mean across scenes, rather than variation across independent evolution runs.

D.3 Object-Level Evaluation

Inputs and comparisons. Table 2 compares the quality of objects constructed through code by scene generators from the same scene descriptions. Each method derives object requirements within its own scene-generation process. SceneCode retains its native modeling interface, and general-purpose agents receive the shared SDK and authoring resources. Target object roles and attributes are defined from each scene description before viewing any generated output and are used only for evaluation. The table reports the evolved VibeArena checkpoint. Generator self-evolution is evaluated separately at scene level. This comparison excludes object-prompted generators such as Articraft and methods that source objects from external 3D asset generators.

Scoring. We follow the GPTEval3D protocol and evaluate five criteria: text–asset alignment (Align.), 3D plausibility (Plaus.), texture–geometry coherency (T-G.), texture details (Text.), and geometry details (Geo.). Target object roles are matched across methods before applying the standard GPTEval3D pairwise evaluation and Elo aggregation. For each matched role, an independent visual evaluator compares assets from two methods using matched multiview RGB and surface-normal renders. Pairwise preferences are aggregated into criterion-specific Elo ratings using the official GPTEval3D implementation *compute_glide_elo*. A controlled same-backend attribution is reported in Appendix D.8.

D.4 Physical Stability Evaluation

Published metrics. Stab. follows SAGE's Section 4.1.1: the percentage of objects whose translation is at most 0.2 m and rotation is at most 8 degrees after 120 simulation steps. The remaining columns follow SceneWeaver's Appendix B.6, Table A5: the percentages of objects displaced by more than 0.1 m and 0.01 m after three seconds, and the mean displacement, reported in centimeters in Table 3. Both source protocols use Isaac Sim; the metrics measure passive stability.

Comparison methods. Table 3 compares SceneSmith, SAGE, SceneCode, and VibeArena. We exclude Code as Room because its released pipeline does not provide the dynamics configuration required for this evaluation. Scene-level stability comparisons use the shared conversion and evaluation settings below.

Shared evaluation conditions. All methods are evaluated on the same scene descriptions with identical simulator settings, gravity, time step, collision handling, and rules for fixed architectural or mounted elements. Each measurement starts from the exported scene configuration without an

evaluation-time settling or repair pass. Stability is measured at step 120, while displacement statistics are measured at step 180. Displacement is the Euclidean distance between initial and final positions; the mean is reported in centimeters. All objects, including large falls, have equal weight. Object identities, scales, and authored articulation are preserved during conversion. Failed loads are excluded from the object-level stability denominator.

D.5 Downstream Fixed-Controller Evaluation

Baseline scope. We evaluate generated worlds from SceneSmith, SAGE, SceneCode, and VibeArena under a common task protocol.

Shared protocol. Each method receives matched task descriptions and robot specifications for pick-and-place, opening and retrieval, and mobile manipulation. For each task family and method, we evaluate 30 SceneEval requests with ten matched reset seeds per request, giving 300 trials. The task planner, motion, gripper, and navigation primitives, simulator configuration, trial seeds, and goal predicates are fixed across methods. For VibeArena, construction-time diagnosis uses Genesis; the Table 5 trials run in Isaac Gym. Worlds and experience snapshots are frozen before these trials; evaluation failures do not trigger world repair or cross-world updates. Success is determined from simulator-state goal predicates. End-to-end task success uses all 300 requested trials as its denominator; a generation or loading failure contributes ten unsuccessful trials. Initial and evolved VibeArena use paired requests and trial seeds.

Qualitative result. Figure 7 visualizes one successful Panda cup manipulation result in a VibeArena-generated world. The 350-step Genesis execution is replayed in Blender for display and is separate from the Isaac Gym trials used to compute Table 5.

D.6 Self-Evolution Evaluation Protocol

Inputs and checkpoints. The learning inputs comprise 85 additional single-room descriptions from the SceneSmith dataset, excluding the SceneEval-100 subset and multi-room prompts, and 50 descriptions from the Function-Room dataset. We partition the 135 learning descriptions into eight sequential rounds and repeat the full process five times with different seeds. The descriptions supply construction experience throughout each mixed-source run; Figure 5 reports mean generation quality across the five runs on these learning sets rather than on a held-out subset. The independent SceneEval-30 test set consists of the 30 SceneEval-100 descriptions used in prior comparisons; it does not contribute to construction knowledge, agent harness updates, or checkpoint selection. For each run, the initial checkpoint precedes cross-world experience accumulation, and the evolved checkpoint is the final snapshot at round $R = 8$. Both checkpoints from each run are evaluated on the same SceneEval-30 descriptions with experience updates disabled during testing; the main-table VibeArena means aggregate the five runs.

Paired improvement analysis. For each metric and evolution run r, we compute d_r as the evolved-minus-initial difference between mean scores on the same 30 SceneEval descriptions. Table 7 reports the mean difference across the five independent runs and its 95% Student-t confidence interval; negative changes improve COL and OOB.

Harness policy statistics. Table 8 organizes policy acceptance and use across the five evolution runs, together with outcomes by issue type. Issue-type changes compare triggered cases with matched runs without that policy type and should not be summed across types. Collision-check ordering increases cost in these runs despite passing the separate policy-acceptance gate.

Construction knowledge statistics. Table 9 summarizes construction knowledge and rule use across five evolution runs, each using the same 135 learning descriptions. A determinate recommendation is counted as effective when the rule-guided construction or repair is followed by successful validation of the resulting world.

Cost accounting. Table 10 compares per-input generation cost for four scene-generation baselines and VibeArena at rounds 0, $R/2$, and R, where R is the total number of evolution rounds.

Table 7: Paired SceneEval-30 changes in percentage points. Intervals use five evolution runs as statistical units; scene-level method intervals use descriptions.

Metric	Mean change	95% CI
CNT	$+3.4$	$[+2.76, +4.04]$
ATR	$+3.8$	$[+2.94, +4.66]$
OOR	$+5.4$	$[+3.64, +7.16]$
OAR	$+4.4$	$[+3.19, +5.61]$
SUP	$+4.2$	$[+3.26, +5.14]$
ACC	$+2.2$	$[+0.91, +3.49]$
NAV	$+0.4$	$[+0.20, +0.60]$
COL	-1.2	$[-1.70, -0.70]$
OOB	-0.2	$[-0.40, 0.00]$

Table 8: Harness policy acceptance and use across five independent evolution runs. Issue-type changes are relative to matched runs without the corresponding policy type. Quality changes are in percentage points; token and time changes are relative percentages.

Metric	Value
Candidate policies evaluated	30
Accepted policies	12
Rejected policies	18
Acceptance rate (accepted / evaluated)	40% (12/30)
Policy invocations	478
Trigger rate (triggered / eligible contexts)	30% (478/1600)
Run coverage (world runs with policy use / total world runs)	39% (261/675)

Issue type	Policies	Triggers	ΔSR	ΔCOL	ΔTokens	ΔTime
Accessibility and clearance	5	200	+1.0	-0.2	-3%	-5%
Collision-check ordering	4	160	+1.4	-0.4	$+2\%$	$+4\%$
Failure diagnosis and repair routing	3	120	+1.8	-0.3	-6%	-8%

Each method is measured on the same requests within each input set; cost includes inspections, failed attempts, and revisions, averaged over all requests. Checkpoint costs reflect joint changes in construction knowledge and harness policies on the same learning requests. Table 11 decomposes the overhead measured over 20 source scenes into three components: source-world construction; knowledge induction and its validation; and agent harness policy proposal and paired validation. The mean cost is 1.85 million tokens and 2.52 hours per source scene. Knowledge updates and harness search account for 0.69 million tokens and 1.03 hours of that per-scene cost. Shared work is counted once, and failed candidates remain in the totals. Elapsed time uses a fixed concurrency configuration.

Table 10: Mean generation cost per learning description from the SceneSmith and Function-Room datasets across checkpoints. Tokens are in millions; time is in seconds.

	SceneSmith dataset		Function-Room dataset	
Method	Tokens	Time (s)	Tokens	Time (s)
SAGE	1.85	8440	2.54	8160
SceneSmith	0.32	8020	0.48	15300
SceneCode	0.98	30969	0.89	22560
Code as Room	0.89	1980	1.31	2330
VibeArena (initial)	**1.19**	**5611**	**1.13**	**5131**
VibeArena (round 4)	0.93	4840	0.92	4596
VibeArena (round 8)	0.79	4160	0.82	4240

Table 9: Construction knowledge and rule-use statistics across five evolution runs, each using 85 descriptions from the SceneSmith dataset and 50 from the Function-Room dataset. Percentages are rounded to whole numbers.

Metric	Value
Deduplicated conditional rules	16
Validated construction templates	12
Independent supporting instances per rule	5 (most 4–7)
Recorded exclusion conditions	12
Rule–instance judgments	675
Abstention rate	20% (135/675)
Effective recommendations among determinate judgments	76% (410/540)

D.7 Ablation Protocols

Table 4 contains four controlled comparisons: construction components, independently learned sources of self-evolution, experience representation, and experience consolidation. The construction comparison uses the full initial pipeline as its reference; the evolution comparison uses the initial generator; the representation comparison uses no retrieved experience. The scene columns are evaluated on the same 30 SceneEval descriptions, and SR is evaluated on the same diagnostic interaction suite across all 4 groups.

Data sources and shared controls. The SceneSmith and Function-Room datasets provide the learning descriptions used to collect accepted construction revisions for self-evolution. SceneEval-30 is reserved for independent scene evaluation and does not contribute to experience collection or checkpoint selection. The complete scene evaluation records CNT, ATR, OOR, OAR, SUP, ACC, NAV, COL, and OOB; Table 4 presents OOR, OAR, ACC, and COL alongside diagnostic interaction success (SR). All variants retain the generation models, modeling SDK, output schema, and evaluation criteria. Matched inputs and seeds support paired comparisons. Resource limits are shared upper bounds rather than requirements to consume identical numbers of tokens. Generation and scoring coverage are recorded alongside scene scores.

Table 11: VibeArena self-evolution cost for 20 source scenes. Tokens are in millions; time is in hours.

Component	Tokens	Time
Source worlds	23.2	29.8
Knowledge updates	3.4	5.6
Harness search	10.4	14.9
Total	37.0	50.3
Mean per source scene	1.85	2.52

Interaction evaluation. The diagnostic suite contains 50 tasks with identical goals, initial conditions, controller, and simulation settings across variants. For each task, each variant generates one world and evaluates it in five execution trials, resetting the world to its initial state before each trial. The same trial seeds are used across variants. Success is determined from simulator-state goal predicates. Generation or loading failure contributes five unsuccessful trials; an execution failure contributes one unsuccessful trial. SR measures task completion across the execution trials, including these failures. Cross-world construction knowledge updates and agent harness updates are disabled throughout evaluation. Tasks and repetitions sharing a generated world are grouped for uncertainty estimation. The cross-generator evaluation in Section 4.5 uses the same fixed-controller principle to compare the task usability of worlds from different methods.

Component isolation. Construction ablations use the initial harness without learned cross-world knowledge. Without hierarchical IRs, the generator writes ArenaProgram directly. Without scene context, generation and repair omit neighboring relations and access constraints. Without realization maps, diagnosis and revision omit requirement-to-structure mappings. Without execution revision, simulation observations do not guide construction edits. All variants use the same final evaluation.

Independent learning paths. Knowledge-only evolution fixes the initial agent harness and updates construction knowledge; harness-only evolution disables learned knowledge retrieval and up-

dates the agent harness. Full evolution enables both mechanisms. Each configuration starts from the same generator and follows the same learning descriptions from the SceneSmith and Function-Room datasets in the same order under matched resource caps. All evaluation snapshots remain frozen throughout testing.

Experience representation. The representation comparison fixes the generator, initial harness, and archive of accepted source repairs; no experience omits the archive. Repair examples and conditional rules derive from the same records and use identical retrieval, ranking, and context-token limits. The plain-summary control uses the same selected repairs, describing the problem, accepted edit, and validation outcome in prose without explicit applicability predicates or numeric bounds. All variants retain SDK documentation and modeling examples, and are tested on varied dimensions and neighboring layouts within supported mechanism families.

Cost accounting. For each learning-path variant, test-time tokens and elapsed time are averaged over attempted worlds, including failed generations and repairs. Learning overhead includes source construction, rule induction and testing, and agent harness policy search where enabled.

D.8 Controlled Object-Level Attribution

Object-level attribution. To isolate VibeArena's contributions, we compare ObjectSpec-only, scene-context, and construction-knowledge conditions under the same Articraft SDK and compiler, object requirements, target scenes, authoring model, seeds, initial harness, and resource limits. Scene context adds neighboring relations and access clearances; knowledge adds validated templates and applicable conditional rules. We refit GPTEval3D Elo ratings over these three conditions and measure joint validity (successful compilation and required motion without joint-limit or self-collision violations) and target-operation success under the same controller. Generation failures count as functional failures.

Table 12: Object-level attribution with a matched Articraft backend and ObjectSpec requirements. Visual columns are GPTEval3D Elo ratings refit within these three conditions; joint validity and operation success are percentages.

Condition	Align.	Plaus.	T-G.	Text.	Geo.	Joint valid	Op. success
ObjectSpec only	947	983	981	997	992	71%	46%
+ Scene context	1008	999	1003	998	1002	79%	56%
+ Construction knowledge	1042	1021	1012	1008	1014	86%	66%

D.9 Experience Consolidation and Ablation

Scope. Consolidation manages conditional rules and scheduling policies accumulated during self-evolution.

Individual updates and cross-scene confirmation. Each update adds or revises one rule or policy while leaving the remaining active entries unchanged. Before activation, new entries are evaluated on learning scenes distinct from their supporting traces. Rules require validated support, while scheduling policies use the paired quality-and-cost criterion described in Appendix A.4. Confirmation and revalidation costs are included in the evolution budget.

Review, refinement, and retirement. Subsequent uses provide additional evidence about each active entry. Repeated failures across applicable learning worlds trigger revalidation. Confirmed regressions either refine an entry's applicability conditions or retire it from active use.

Controlled ablation. Table 4 compares full consolidation with variants that remove individual updates, cross-scene confirmation, or retirement. All variants use matched eight-round evolution runs with the same learning order, random seeds, resource budgets, and frozen evaluation snapshots. SceneEval-30 provides scene-level metrics, and the diagnostic interaction suite provides SR.