arXiv:2605.09423v2 [cs.AI] 13 May 2026

SIMWORLD STUDIO: Automatic Environment Generation with Evolving Coding Agent for Embodied Agent Learning

Haoqiang Kang[1]* **Xiaokang Ye**[1]* **Yuhan Liu**[2] **Siddhant Hitesh Mantri**[1]
Lingjun Mao[1] **James Fleming**[1] **Drishti Regmi**[1] **Lianhui Qin**[1]
[1]UC San Diego [2]New York University

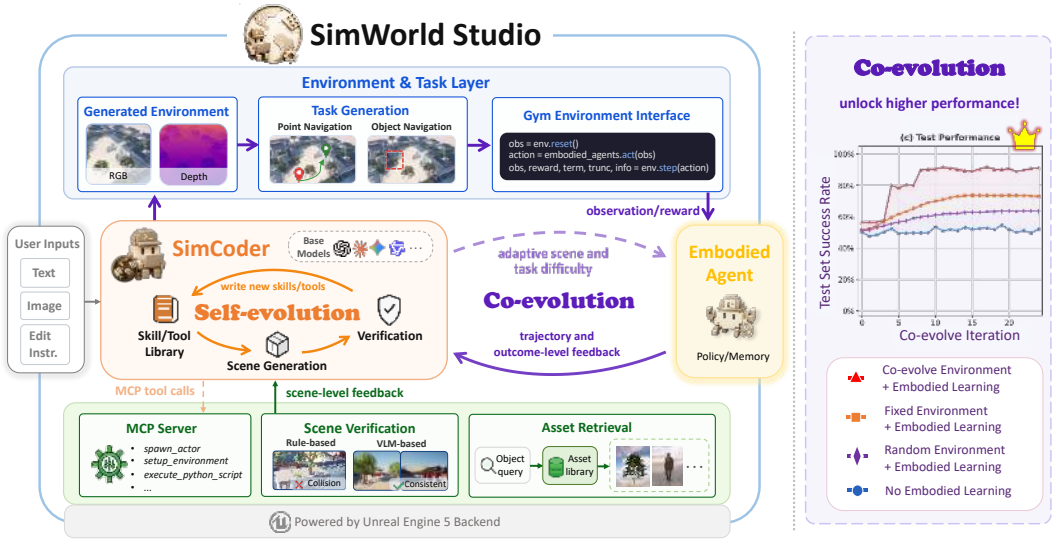


Figure 1: **SIMWORLD STUDIO: (Left)** SIMCODER automatically generates UE5 interactive environments with realistic 3D scenes, learning tasks, and Gym interfaces. **(Right)** Co-evolving environment generation with embodied learning substantially improves test success over both fixed-environment training and the untrained-agent baseline.

Abstract

LLM/VLM-based digital agents have advanced rapidly thanks to scalable sandboxes for coding, web navigation, and computer use, which provide rich interactive training grounds. In contrast, embodied agents still lack abundant, diverse, and automatically generated 3D environments for interactive learning. Existing embodied simulators rely on manually crafted scenes or procedural templates, while recent LLM-based 3D generation systems mainly produce static scenes rather than deployable environments with verifiable tasks and standard learning interfaces. We introduce SIMWORLD STUDIO, an open-source platform built on Unreal Engine 5 for generating evolving embodied learning environments. At its core is SIMCODER, a tool/skill-augmented coding agent that writes and executes engine-level code to construct physically grounded 3D worlds from language/image instructions. SIMCODER *self-evolves* by using verifier feedback (e.g., compilation errors, physics checks, VLM critiques) to revise environments and autonomously add reusable tools and skills to its library. Generated worlds are exported as Gym-style environments for embodied agent learning. SIMWORLD STUDIO further enables *co-evolution* between environment generation and embodied learning: agent performance feedback guides SIMCODER to generate adaptive curricula near the learner's

* Equal contribution.

capability frontier, so that environments become increasingly challenging as the embodied agent improves. Three case studies on embodied navigation show that self-evolution improves generation reliability, generated environments substantially improve embodied agent performance that generalizes to unseen benchmarks, and co-evolution yields an 18-point success-rate gain over fixed-environment learning and a 40-point gain over an untrained agent.

1 Introduction

Large language and vision models have recently made striking progress as *digital agents*: they can write and debug code, operate graphical user interfaces, navigate the web, and complete multi-step tasks in software environments. A key enabler of this progress is the availability of scalable interactive digital sandboxes, such as code execution environments and operating-system simulators, in which agents can act, receive feedback, and improve through repeated experience [18, 28, 101]. By contrast, progress toward similarly capable *embodied agents* remains comparatively limited. Although LLMs and VLMs provide powerful priors for perception, reasoning, and planning in 3D worlds [20, 106], embodied learning still lacks the kind of abundant, diverse, and automatically generated interactive environments that digital agents increasingly rely on.

A central bottleneck is the difficulty of simulating embodied environments at scale. Training and evaluating embodied agents require not only visually plausible 3D scenes, but also physically grounded worlds in which agents can be deployed, take actions, observe consequences, and receive task feedback. Existing embodied platforms, such as AI2-THOR [40], Habitat [56], CARLA [19], ThreeDWorld [25], and iGibson [42], provide important infrastructure for embodied AI, but they largely depend on manually designed scene collections that are expensive to construct, limited in diversity, and fixed once released. Procedurally generated platforms such as ProcTHOR [16] and Infinigen [60] improve scalability, yet their diversity is still bounded by hand-designed templates or rules. Meanwhile, a growing line of work explores LLM- or coding-agent-based 3D scene generation, either by predicting layouts or by writing executable code against a game engine [89, 35, 100, 83, 41, 47, 55]. However, these systems primarily generate *static scenes*: their outputs are typically evaluated as visual or geometric artifacts, rather than as deployable interactive environments.

The distinction between *scene generation* and *environment generation* is crucial. For embodied agent learning, a generated world must be more than a visually plausible arrangement of objects: it must be an interactive system in which agents can perceive, act, and receive feedback. Such environments should expose observations and actions through a standard interface, define verifiable tasks, provide reward signals, and support training and evaluation without manual integration. Moreover, the environment generator itself should not remain fixed. As an embodied agent improves, the simulator should be able to generate more diverse, complex, and challenging environments informed by the agent's current capabilities. Such a closed loop would turn environment generation from a one-shot content-creation problem into an adaptive curriculum mechanism, where the worlds generated for training evolve together with the agents learning inside them.

We introduce SIMWORLD STUDIO, an open-source platform built on Unreal Engine 5 for automatic generation of evolving interactive embodied learning environments (Figure 1). At its core is SIMCODER, a tool-augmented coding agent that creates realistic, physically grounded UE5 environments from natural-language instructions, image guidance, and editing requests. Rather than merely placing static assets, SIMCODER writes and executes engine-level code to construct diverse environments, ranging from simple street corners to full city districts. It uses rich verifier feedback, including compilation errors, collision reports, physics checks, and VLM critiques, to revise generated environments for improved validity (Figure 2). Over time, SIMCODER can also autonomously author new tools and reusable skills, add them to its own library for reuse in future generations, thereby improving reliability and scalability. Similar to previous tool-making LLMs [11, 72], this mechanism closes a *self-evolution* loop for the coding agent without manual intervention.

Every environment generated by SIMWORLD STUDIO can be seamlessly exported as a standardized Gymnasium-style embodied environment, with `reset()`, `step()`, and task-dependent observation spaces, action spaces, and reward signals. In this work, we use navigation as a representative case study: tasks are automatically derived from the generated scene structure, including traversable regions, obstacles, goals, and spatial relations. This allows LLM-based or other embodied agents

Code is available at `https://github.com/SimWorld-AI/SimWorld-Studio`.

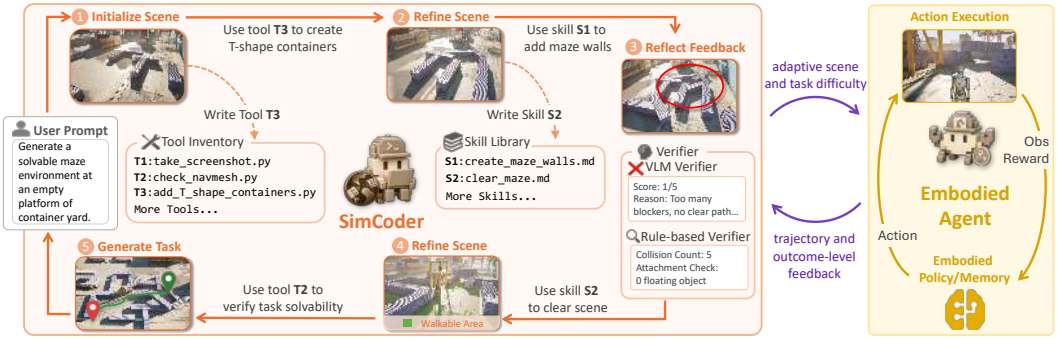


Figure 2: SIMCODER turns a user prompt into an interactive environment through an automatic self-evolving loop: it writes tools, creates reusable skills, reuses them across iterations, and refines the scene with verifier feedback. NavMesh-based tools are used to generate solvable navigation tasks. SIMCODER furthers uses embodied-agent feedback to autonomously adapt environment difficulty and co-evolve with the embodied agent (purple loop).

to be deployed directly in generated worlds and trained on verifiable downstream tasks. Crucially, SIMWORLD STUDIO also supports a *co-evolution* loop between the coding agent and the embodied agent. Performance signals from the embodied learner, such as task success, failure modes, and exploration coverage, are fed back to SIMCODER, steering future generation toward environments near the frontier of the learner's current ability. In this way, SIMWORLD STUDIO aims to provide not only a scalable source of embodied training environments, but also an adaptive platform in which environment generation and embodied agent learning improve together. Compared with preliminary attempts [94], which use LLMs to adapt predefined simple game environments for small RL agents, SIMWORLD STUDIO provides a flexible and realistic platform for environment-agent co-adaptation.

Across three case studies (Figure 4), we show that (i) SIMCODER reliably generates physically valid and prompt-aligned environments, with structured tools, verification, and self-evolution each contributing measurably to quality; (ii) embodied agents trained in the generated environments achieve substantial improvements that transfer to unseen navigation benchmarks, with environment diversity directly driving generalization; and (iii) closing the co-evolution loop between SIMCODER and the embodied agent via an adaptive curriculum yields an 18-point Success Rate gain over fixed-environment training and a 40-point gain over an untrained agent, showing that generated environments become more effective for embodied learning when shaped by agent feedback. We will open source the platform and all experiments upon acceptance.

2 SIMWORLD STUDIO

SIMWORLD STUDIO is built on the open-source Unreal Engine 5 based SIMWORLD library [91] by inheriting its assets, runtime, and Python wrapper on the UE5 backend, enabling highly realistic, physically grounded environments. SIMWORLD STUDIO makes two main methodological contributions: **(1) Automatic environment generation** (§2.1): a coding agent that synthesizes executable 3D scenes, evolves its own skill and tool library from verifier feedback, and exports each scene as a Gymnasium-compatible embodied environment. **(2) Co-evolution as an adaptive curriculum mechanism** (§2.2): embodied agent performance is fed back into environment generation, so new environments target the agent's current weaknesses and remain near the boundary of its capabilities. See our main UI page in Fig 3.

2.1 SIMCODER: Coding Agent for Automatic Environment Generation

As shown in Figure 1(Left), SIMWORLD STUDIO comprises three components: **SIMCODER**, an LLM coding agent that drives generation; *tool and skill libraries*, including an inventory of Python functions as tools and a library of skills which are reusable procedures, exposed through a Model Context Protocol [3] (MCP) bridge; and *verifiers* that return verification signals (rule- and VLM-based) to guide scene construction and revision. As illustrated in Figure 2 with a maze-generation task, generation flows in a loop: given a user prompt (text, image, or edit instruction), SIMCODER issues tool calls or skill retrievals through MCP; the backend executes them and returns a state update

Additional UI views, running cases, prompts, and generated tools/skills/examples are provided in Appendices C, H, and I.

or a verifier signal, which SIMCODER consumes as the next observation and either continues building the scene, revises in place, or, when a fix proves broadly useful, writes a new tool or skill back into its library so future generations can reuse it. Once the scene passes verification, SIMCODER derives a task from it and exports it as a Gymnasium environment, allowing embodied agents to interact with it.

SIMWORLD STUDIO User Interface

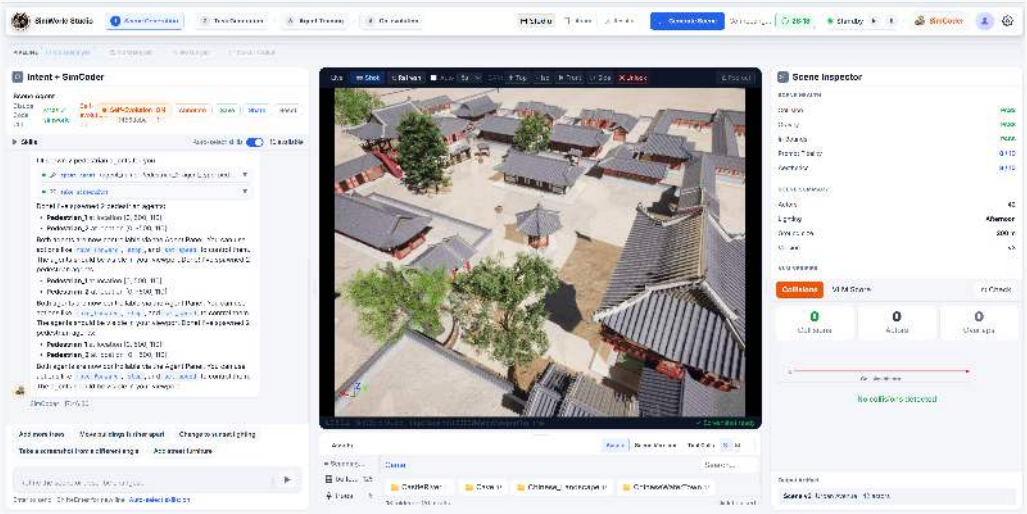


Figure 3: The workspace supports the full environment-generation workflow in a single view. The top pipeline bar tracks progress across *scene generation*, *task generation*, *agent training*, and *co-evolution*. In this scene generation page, the left panel combines natural-language interaction with SIMCODER, executable scene and agent-control skills, and quick editing actions. The central viewport streams the live Unreal Engine scene, while the bottom asset browser exposes available scene assets and saved scene versions. The right *Scene Inspector* shows scene health, scene summary, and VLM-based verification.

MCP Tools. Tools are Python function calls that SIMCODER invokes through the MCP bridge to act on the UE backend. The inventory has two parts. *Primitive tools* are the fixed, predefined set of operations needed to author a scene end-to-end (e.g., actor management, environment and asset management, and scene evaluation). *Extensible tools* cover everything outside this fixed set: a Python escape hatch runs arbitrary Unreal Engine Python for one-off operations, and any pattern that proves useful across runs is promoted via self-evolution into a named wrapper that the bridge registers as a first-class MCP tool, indistinguishable from the primitives at call time. Step 1 of Figure 2 shows one such wrapper (`add_T_shape_containers.py`) being invoked from the Tool Inventory. The full primitive inventory is in Appendix C.1.

Skill Library. Skills sit one layer above tools. Each skill is a Markdown document that records how to use a tool (or a sequence of tools) to accomplish a particular composition goal; SIMCODER retrieves applicable skills at the start of each episode and issues the underlying tool calls itself, so skills tell it how to compose tools rather than bypass them. As with tools, the library has two parts: a small set of *primitive skills* ships with the platform (covering common composition goals such as building placement, city layout, and screenshot capture for the VLM judge), and *extensible skills* accumulate over time through self-evolution. Step 2 of Figure 2 shows SIMCODER retrieving an evolved skill (`create_maze_walls.md`) to add walls to the partially-built maze.

Verification Loop. SIMWORLD STUDIO verifies generated scenes through two complementary verifiers (Step 3 of Figure 2). A *rule-based verifier* computes physical and geometric metrics (e.g., collisions, vertical support, in-bounds placement) from the scene graph and is invoked on every actor-modifying tool call. A *VLM-based verifier* captures multi-view screenshots and asks a vision-language model to score semantic alignment against the prompt, returning structured feedback after each block of construction. Verifier responses re-enter the trajectory as the next observation, and SIMCODER revises in place. In the maze episode of Figure 2, for example, the rule-based verifier reports a collision count of 5 and the VLM verifier scores the scene 1/5 (*"too many blockers, no*

clear path…"); SIMCODER then retrieves the `clear_maze.md` skill and removes the redundant containers before continuing. Full metric definitions are in Appendix E.2.

Self-evolution. Self-evolution turns one-off fixes into permanent capabilities. When a verifier failure recurs across attempts, SIMCODER restates the failure at the level of a *class* of cases and authors a new tool or skill that addresses the class rather than the specific instance, writing it to the registry so all subsequent runs can retrieve it [11, 13]. Step 4 of Figure 2 illustrates one such update: after the maze fails verification, SIMCODER writes a new skill (`clear_maze.md`) that generalizes the corrective procedure (i.e., removing redundant blockers from any container layout) and the skill is then available for all future episodes. Representative authored entries are in Appendix C.3.

Task Generation. SIMCODER also generates a task on top of a generated scene, using the same tool-call interface to query scene structure (e.g., NavMesh for traversable regions). Step 5 of Figure 2 shows the maze scene compiled into a navigation task with a sampled start–goal pair on the walkable area. We instantiate two canonical navigation families as a representative case: *point navigation* [2] (goal = coordinate) and *object navigation* [8] (goal = semantic target). Task solvability is guaranteed by NavMesh connectivity, and verifiability follows from the same scene-query tools: during execution, we directly query the agent pose and target location and check success based on distance to the target.

Gymnasium Compilation. A generated environment then exports as a standard Gymnasium environment, with `env.reset()` and `env.step(action)` returning RGB-D observations, agent pose, and reward (top of Figure 1(Left)). Because the contract is the standard one, any off-the-shelf RL algorithm (e.g., PPO [64]) or training-free LLM policy (e.g., ReAct [90]) plugs in without modification, making each generated scene a first-class training substrate for embodied agent learning.

2.2 Co-Evolution: An Adaptive Curriculum Mechanism

So far the generator runs open-loop: it produces environments without knowing how the embodied agent fares in them. Co-evolution closes this loop and turns environment generation from a one-shot content-creation problem into an adaptive curriculum mechanism, where the scenes generated for training evolve together with the agents learning inside them. One round alternates two updates: the embodied agent trains on a batch of SIMCODER-generated environments, and SIMCODER then updates based on the resulting performance before producing the next batch. The two agents update individually, through different mechanisms.

Embodied Agent Evolving. From the embodied agent's perspective, co-evolution differs from fixed-environment training only in that the scene distribution drifts between rounds; the agent's update rule is unchanged. SIMWORLD STUDIO reuses the Gym interface of §2.1 without modification, so an RL policy (e.g., PPO [64]) updates via standard policy gradients on the reward returned by `step()`, while an LLM-based policy updates through in-context mechanisms such as incremental rule accumulation or reflection-style memory [72, 66].

Coding Agent Evolving. SIMCODER's update is in-context: between rounds the embodied agent's performance is fed back as context for the next generation episode, and SIMCODER reweights its skill retrievals and tool invocations to raise difficulty where success rates plateau, lower it where the agent stalls, and oversample structural features the agent has not yet mastered. The underlying LLM weights are not modified. The performance signal is read through three feedback channels at increasing abstraction: *scene-level* feedback reports physical validity and prompt alignment of scenes; *outcome-level* feedback provides task success and return statistics for difficulty-matching objectives [74, 17]; and *trajectory-level* feedback exposes the agent's per-episode experience for reflection-based updates to SIMCODER's generation principles [72]. A specific co-evolution *recipe* selects a subset of these channels and pairs it with the embodied agent's learning rule.

Section 3.3 instantiates this recipe for navigation tasks, using outcome-level agent outcomes to adapt SIMCODER's difficulty schedule while the agent improves through incremental rule accumulation. The resulting adaptive curriculum outperforms fixed-environment training.

3 Experiments and Analysis

We analyze SIMWORLD STUDIO through three case studies of increasing scope (Figure 4): environment generation quality (§3.1), embodied agent learning in generated environments (§3.2), and co-evolution between the environment generation and the embodied agent (§3.3).

Figure 4: **Three case studies evaluating SIMWORLD STUDIO.** Case 1 evaluates SIMCODER's scene generation quality across settings and LLM backbones. Case 2 trains embodied navigation agents in generated environments. Case 3 studies co-evolution where SIMCODER and the embodied agent iteratively improve each other.

3.1 Case Study 1: Can SIMCODER generate valid and diverse environments?

This case study evaluates whether SIMCODER can generate diverse, physically plausible 3D environments from natural language prompts, reference images, and editing instructions. As illustrated in Figure 4 (case study 1 left), SIMCODER receives a text prompt (e.g., "build a residential neighborhood with parallel streets and a park"), invokes MCP tools to spawn and arrange assets in the UE5 environment, and iteratively refines the scene through screenshot-based verification (§2.1).

Settings. We evaluate across three settings of increasing complexity: **(S1)** *Text-to-Scene*: generate a scene from a natural language prompt alone; **(S2)** *Image+Text-to-Scene*: generate with an additional reference image (hand-drawn sketch or aerial photo); **(S3)** *Scene Editing*: modify an existing scene by adding, removing, or rearranging objects without rebuilding from scratch. Each setting is tested at three difficulty levels (easy, medium, hard), yielding 9 evaluation scenes total. We use the two-axis evaluation from §2.1: rule-based metrics for physical validity (e.g., collision-free placement, gravity consistency, in-bounds placement) and VLM-as-judge metrics for semantic alignment (e.g., prompt fidelity, spatial fidelity, layout aesthetics); full definitions are in Appendix E.2.

Base Models. We benchmark four LLM backbones, including Claude Opus 4.6 [5], Claude Sonnet 4.6 [6], and Qwen3.5-27B/9B [59], all through the Claude Code agent framework [4] with the same MCP tool interface, verification loop, and skill library (§2.1). All agents differ only in the underlying LLM, isolating the contribution of model capability from platform infrastructure.

Table 1: **Scene generation quality (averaged across difficulty levels).** All metrics $\in [0, 1]$ (↑). Bold = best per column. Metric colors: quantity, physical validity, semantic, aesthetic.

	S1: Text-to-Scene							S2: Image+Text-to-Scene							S3: Scene Editing						
	Rule-Based				*VLM*			*Rule-Based*				*VLM*			*Rule-Based*				*VLM*		
LLM	*Count*	*Diversity*	*No Collision*	*Gravity*	*Fidelity*	*Aesthetics*	**Avg**	*Count*	*No Collision*	*Gravity*	*In-Bounds*	*Fidelity*	*Style*	**Avg**	*Preserve*	*Edit Count*	*No Collision*	*Coherence*	*Edit Compl.*	*Layout*	**Avg**
Qwen3.5-9B	.00	.50	.37	.58	.20	.13	.36	.60	.33	.53	1.0	.13	.13	.45	1.0	.00	.37	.23	.23	.20	.34
Qwen3.5-27B	.17	.46	.99	1.0	.43	.37	.59	.73	.75	.83	1.0	.38	.30	.67	1.0	.00	.99	.47	.13	.50	.52
Sonnet 4.6	.33	**.80**	**1.0**	1.0	.53	.37	.70	.80	.81	.87	1.0	.50	.37	.73	1.0	1.0	.98	.40	**.67**	.40	.74
Opus 4.6	**.75**	.72	.99	1.0	**.63**	**.47**	**.77**	**.83**	**.86**	**.92**	1.0	**.62**	**.47**	**.79**	1.0	1.0	.98	**.50**	.50	**.53**	**.75**

3.1.1 Results

Table 1 reports performance averaged across difficulty levels; full breakdowns are in Appendix E.1.

SimCoder with different coding models generates physically valid environments; quality scales with model capability. Near-perfect physical validity holds across all settings, Opus 4.6 and Sonnet 4.6 maintain collision-free rates $\geq$0.98 regardless of input modality or difficulty (see Appendix E.1). Semantic quality scales with model size: Opus 4.6 leads across all three settings (S1: 0.77, S2: 0.79, S3: 0.75), and image guidance consistently boosts smaller models (Qwen3.5-27B: S1 0.59→S2 0.67) by anchoring spatial layout.

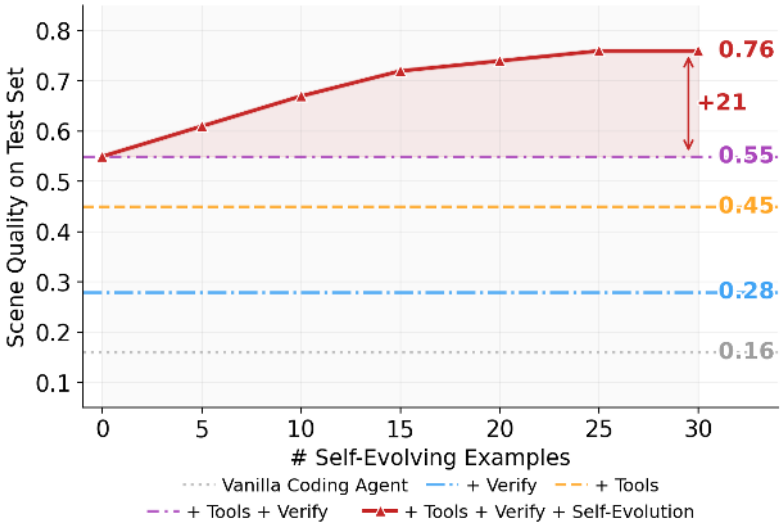


Figure 5: Ablation study results for SimCoder in Case Study 1.

Ablation studies on key platform components. Figure 5 ablates three platform components beyond the vanilla coding agent: MCP tools, verification loop, and self-evolution. The evaluation is conducted on a held-out test set of 9 scenes across S1/S2/S3. First, we observe that the *vanilla coding agent* fails to construct reliable environments (scoring 0.16). Then, adding *customized MCP tools* raises quality to 0.45 (+0.29), providing the structured action space needed for reliable asset interaction. Moreover, adding the *verification loop* improves quality by +0.10, as iterative screenshot-based correction catches spatial errors that single-pass generation misses. We find that *self-evolution* can break the plateau shared by all static configurations, further raising a +0.21 quality improvement by accumulating reusable placement strategies across generations. Together, these results show that structured tool access is a hard prerequisite, while self-evolution provides the largest quality gain by converting experience into reusable knowledge. Full ablation details are in Appendix E.3.

3.1.2 Qualitative Example: Text-to-Scene Generation

Figure 6 presents a concrete example of how SimCoder transforms a natural-language prompt into a complete UE5 scene. Figure 7 further illustrates the corresponding tool calls, skills, and code used to construct the scene. Additional qualitative examples are in Appendix I.

User Prompt Text-to-Scene Setting

Build a believable medieval village street scene, in daytime. A narrow cobblestone lane runs east–west; timbered houses and a tavern flank **both sides shoulder-to-shoulder**. Outside each house there's the household's everyday clutter, a stack of barrels, a wheelbarrow, some grain sacks, and a tree or two grows between certain houses. A well sits at a small square, and one or two houses have a market canopy fused to the front wall like a stall opening onto the street. Villagers cluster around the well and the market. The aesthetic mood, counts, exact placement, and which-house-gets-what are all your creative call. Everything else is yours to decide: lane length, per-house clutter style, number of canopies, villager density, time-of-day mood.

Claude Opus 4.6 Count 0.72 Diversity 0.70 No Collision 0.78 Gravity 0.74 Fidelity 0.64 Aesthetics 0.66 **Avg. 0.71**

(1) Cart sunken into ground

Qwen3.5-27B Count 0.52 Diversity 0.58 No Collision 0.70 Gravity 0.66 Fidelity 0.48 Aesthetics 0.56 **Avg. 0.58**

(1) Missing house in continuous row

(2) Cart sunken into ground

Qwen3.5-9B Count 0.34 Diversity 0.46 No Collision 0.45 Gravity 0.38 Fidelity 0.30 Aesthetics 0.42 **Avg. 0.39**

(1) Missing house in continuous row

(2) Missing house in continuous row

(3) Cart sunken into ground

(4) Cart sunken into ground

Figure 6: **Qualitative medieval village scene comparison.** Each generated scene is paired with its corresponding text-to-scene evaluation scores. Scores are normalized to $[0, 1]$; higher is better.

(a) MCP Tool

```
def execute_python_script( script:  str,  timeout:  int = 60 ) -> dict
```

Runs a Python script inside UE5's embedded interpreter. The script can access `unreal` and staged skill modules such as `road`, `buildings`, `street_dressing`, `people`, and `lighting`.

parameter	type	description
`script`*	`str`	Python source to execute; one call builds the scene.
`timeout`	`int`	Max execution time in seconds. Default: `60`.

(b) Skill

 # pack_flush_row

Theme-agnostic shoulder-to-shoulder building packer. Given a pool of Blueprint IDs, it places mixed-width buildings along `+X` with facades flush to a wall line.

Param	Type	**Use**
`asset_ids`	`list`	Blueprint pool controlling the visual style.
`x_range`	`tuple`	Packing extent `(x_min, x_max)`.
`flush_y`	`float`	Wall-line Y coordinate.
`side`	`str`	`"south"` or `"north"` controls side and yaw.
`seed`	`int`	Randomizes ordering without mirroring rows.

(c) Generated Code

```
# Narrow cobble lane centered on the origin
x_min, x_max, y_min, y_max, _ = lay_cobble_path(
    length_tiles=26, width_tiles=5,
    start_x=-4550, start_y=-875, seed=42
)

# Pack medieval buildings tightly along both street edges
MED_POOL = [MED_BP_TAVERN, MED_BP_HOUSE_1, MED_BP_HOUSE_2, MED_BP_HOUSE_3]

pack_flush_row(MED_POOL, (x_min, x_max), flush_y=y_min,
    side="south", margin=350, seed=11, label_prefix="Bldg_Med_S_")
pack_flush_row(MED_POOL, (x_min, x_max), flush_y=y_max,
    side="north", margin=350, seed=29, label_prefix="Bldg_Med_N_")

# Add household clutter and finish the scene
dress_each_house("Bldg_Med_S_", side="south", props_per_house_range=(0, 4), seed=101)
dress_each_house("Bldg_Med_N_", side="north", props_per_house_range=(0, 4), seed=202)
place_well(x=-2730, y=0, label="Med_Well_Square")
apply_daytime_medieval_lighting(time_of_day="morning")  # daytime preset
```

Figure 7: **Example of tool, skill, and generated script used in the example in Fig 6.** The MCP tool is presented as a Python function, the skill as Markdown-style documentation, and the generated scene program as executable Python.

3.2 Case Study 2: Can embodied agents learn useful navigation abilities from SIMWORLD STUDIO-generated environments?

We evaluate whether LLM-based embodied agents can learn navigation policies in SIMWORLD STUDIO-generated environments and generalize to unseen scenes. As shown in Figure 4 (Case Study 2), an agent navigates a SIMCODER-generated city from RGB, bearing, and distance observations. Navigation serves as a functional probe of environment quality, since failures may reflect either weak policies or defects in the generated world.

Settings. We consider two outdoor tasks: *Point Navigation* (PointNav), where the agent navigates to a target coordinate, and *Object Navigation* (ObjectNav), where it locates a target object category. Episodes are generated on SIMWORLD STUDIO maps via UE5 NavMesh by sampling start–goal pairs, computing shortest-path references, and filtering by reachability and path length. For each task, we sampled 1.2K episodes as our training set and evaluated agents on 329 held-out SIMWORLD STUDIO episodes; we additionally evaluate on the external SimWorld-MMNav benchmark [104] for cross-benchmark transfer. To assess how scene diversity affects downstream learning, we vary the number of training environments from 1 to 30 with the training budget fixed at 200 episodes.

Metrics. We report Success Rate (SR) for *task success*, Success weighted by Path Length (SPL) and Soft Success weighted by Path Length (SoftSPL) for *path efficiency*, and normalized Dynamic Time Warping (nDTW) for *trajectory fidelity*. Definitions are in Appendix F.3.

Models. We evaluate Qwen3.5-2B/9B/27B [59] with and without agent memory. Our memory design is inspired by Generative Agents [52] and ExpeL [101], but extends them with *multi-level* updates at step, trajectory, and task granularities, allowing distilled strategies to be retrieved at the appropriate scope during inference (Appendix F.4).

3.2.1 Results

Table 2: **Embodied navigation on generated environments.** We report SR, SPL, SoftSPL, and nDTW. Metric colors denote task success, path efficiency, and trajectory fidelity. **Bold** marks the best result within each column; green marks improvement >0.05 over baseline.

LLM	Setting	Point Navigation				Object Navigation			
		SR(%, ↑)	SPL(↑)	SoftSPL(↑)	nDTW(↑)	SR(%, ↑)	SPL(↑)	SoftSPL(↑)	nDTW(↑)
Qwen3.5-27B	Baseline	20.21	0.0931	0.0882	0.2322	22.56	0.0795	0.0782	0.2486
	+ Memory	**26.76**	**0.1266**	**0.1036**	**0.2676**	**31.59**	**0.1429**	**0.1161**	**0.3159**
Qwen3.5-9B	Baseline	0.40	0.0039	0.1137	0.1137	0.00	0.0000	0.1160	0.1160
	+ Memory	**7.43**	**0.0722**	**0.2617**	**0.2617**	**14.62**	**0.1408**	**0.3059**	**0.3059**
Qwen3.5-2B	Baseline	1.38	0.0138	0.1030	**0.0034**	0.24	0.0024	0.1080	**0.0041**
	+ Memory	**6.23**	**0.0528**	**0.2119**	0.0022	**4.58**	**0.0389**	**0.1841**	0.0036

SIMWORLD STUDIO improves the embodied agents in both held-out environments and external benchmark. On held-out SIMWORLD STUDIO test environments (Table 2), agents trained in SIMWORLD STUDIO substantially outperform the no-training baseline across all model scales (e.g., Qwen3.5-9B: +14.62% SR on ObjectNav). These gains transfer to the external SimWorld-MMNav benchmark (Figure 8(Right); +12.0pp on 2B), indicating that SIMWORLD STUDIO captures transferable navigation knowledge rather than environment-specific shortcuts.

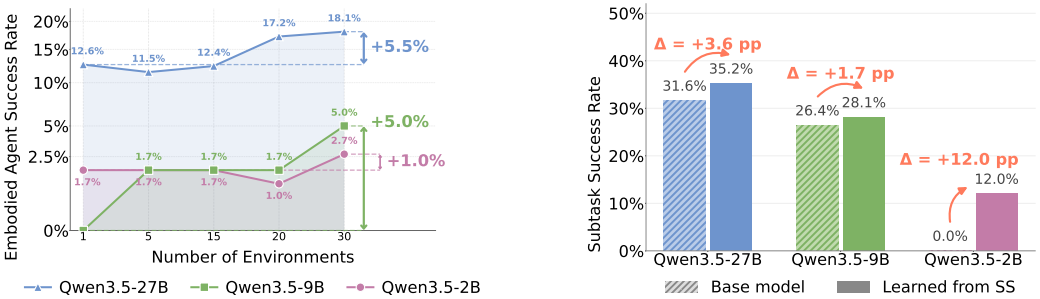


Figure 8: **Generalization analysis. (Left)** More diverse SIMWORLD STUDIO environments yield stronger test-time generalization. **(Right)** Embodied agents learned in SIMWORLD STUDIO transfer to SimWorld-MMNav across model scales.

SIMWORLD STUDIO's open-ended environment generation capability translates into stronger embodied agent generalization. A central capability of SIMWORLD STUDIO is generating an essentially unbounded number of distinct environments on demand, and we find this has a measurable downstream effect: test success rises with the number of distinct SIMWORLD STUDIO-generated training environments (Figure 8(Left); +5.5pp on Qwen3.5-27B). Since the training budget is fixed, these gains are attributable to scene diversity rather than additional experience, which means the platform's scalable generation directly converts into stronger embodied agents. We additionally ablate observation modalities, finding that SIMWORLD STUDIO's built-in RGB-D interface outperforms text-only inputs by providing complementary geometric and semantic cues (Table 10, Appendix F.5).

3.3 Case Study 3: Can SIMCODER co-evolve with embodied agents with agent feedback?

Case Studies 1 and 2 evaluate SIMCODER and the embodied agent in isolation. As a demonstration of the platform's closed-loop capability, Case Study 3 instantiates a simple performance-gated adaptive curriculum. As shown in Figure 9 (a), SIMCODER generates progressively harder environments while the embodied agent updates its policy from experience, with each agent's output steering the other's next step via the co-evolution framework of §2.2.

Method. We instantiate a specific adaptive curriculum under the co-evolution framework of §2.2. Navigation difficulty is parameterized along two axes: path length and obstacle density, which are jointly quantized into eight levels of increasing challenge. Each co-evolution round proceeds in three explicit steps: **(i) Environment Generation:** SIMCODER generates a batch of navigation episodes at the current difficulty level. **(ii) Embodied Agent Learning:** The embodied agent attempts the batch and records any failed episodes. It then distills these failure trajectories into a small set of prioritized decision rules (e.g., *"if the goal lies behind the agent, turn around before exploring forward"*). Crucially, it *appends* these to its existing rule set. Unlike GEPA [1], which rewrites the full prompt each round, this incremental accumulation preserves proven strategies from earlier levels while adding new failure-derived corrections. **(iii) Environment Adaptation:** SIMCODER evaluates the agent's mean Success rate on the batch. If the agent clears a level-specific mastery threshold, SIMCODER advances the next batch of environments to a higher difficulty; otherwise, it holds the current level until the agent catches up. Full details are in Appendix G.

Setting. We evaluate on the external SimWorld-MMNav benchmark [104] and compare four conditions using Qwen3.5-9B: **(1)** co-evolving environment (our full closed-loop system), **(2)** fixed-difficulty environment (held constant at level 3), **(3)** random-difficulty environment (random level from 0 to 7), and **(4)** base model (no embodied agent learning).

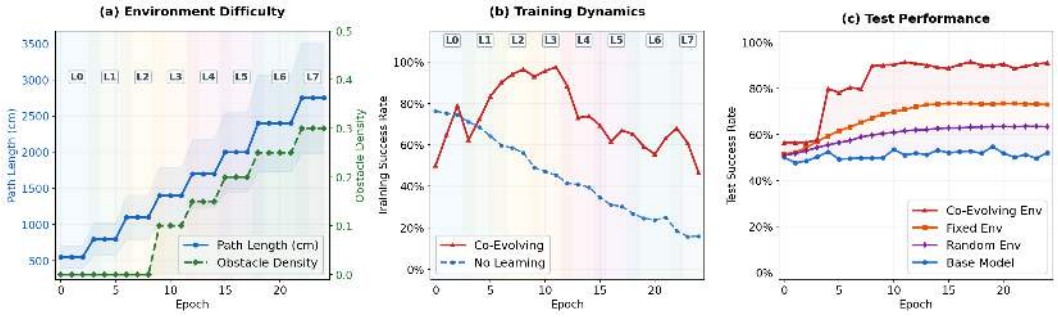


Figure 9: **Co-evolution of SIMCODER and embodied agent. (a)** Environment difficulty across 8 levels. **(b)** Training dynamics: the co-evolving agent drops at each level transition then recovers. **(c)** Test performance on the SimWorld-MMNav benchmark.

3.3.1 Results

Adaptive curricula drive continuous improvement and prevent early saturation. The training dynamics of the co-evolving system (Figure 9b) exhibit a characteristic *drop-and-recover* pattern [71]: the agent's Success rate dips exactly when SIMCODER introduces harder tasks, but recovers rapidly as distilled strategies transfer to the new difficulty tier. Ultimately, this closed-loop co-evolution yields a 90% test Success rate on SimWorld-MMNav. This represents a massive 18-point gain over the fixed-environment learning baseline (72%), which saturates midway through training because it is not pushed to its capability frontier, and a 40-point gain over the untrained baseline (50%; Figure 9c).

Table 3: Comparison with representative embodied simulation platforms, generative scene-construction systems, and environment-agent co-evolution methods. **Diverse Gen.**: supports diverse environment generation at scale. **Phys./Vis. Realism**: fidelity of physics simulation and visual rendering. **Gym Interface**: provides a standard Gymnasium-compatible API. **Self-Evolution**: autonomous improvement of the platform's own generation or operation behavior from verifier feedback. **Co-Evolution**: environment generation adapts based on downstream embodied agent performance.

Platform	Engine	Diverse Gen.	Phys./Vis. Realism	Gym Interface	Self-Evolution	Co-Evolution
CARLA [19]	UE4	✗	+++	✗	✗	✗
ThreeDWorld [25]	Unity	✗	++	✗	✗	✗
AI2-THOR [40]	Unity	✗	++	✗	✗	✗
MineDojo [22]	Minecraft	✗	+	✓	✗	✗
ProcTHOR [16]	Unity	✓	++	✗	✗	✗
Habitat 3.0 [56]	Habitat-Sim	✗	++	✓	✗	✗
MetaUrban [81]	PyBullet	✓	++	✓	✗	✗
GRUtopia [73]	Isaac Sim	✓	++	✓	✗	✗
EmbodiedCity [26]	UE4	✗	+++	✗	✗	✗
UnrealZoo [102]	UE4/5	✗	+++	✓	✗	✗
Virtual Community [103]	Genesis	✓	++	✗	✗	✗
VirtualEnv [68]	UE5	✗	+++	✗	✗	✗
Holodeck [89]	AI2-THOR/Unity	✓	++	✗	✗	✗
SAGE [83]	Isaac Sim	✓	+++	✗	✓	✗
GenEnv [28]	AlfWorld/Text-only	✗	+	✗	✗	✓
SIMWORLD STUDIO	**UE5**	✓	+++	✓	✓	✓

4 Related Work

Embodied simulation platforms. Embodied AI research relies on three families of interactive simulators, each with structural limitations. *Hand-built indoor and outdoor platforms* support navigation, manipulation, driving, urban robotics, and language-grounded games [40, 42, 56, 43, 14, 19, 73, 81, 26, 67, 22, 102, 68], but rely on manually authored scene catalogs that are fixed and expensive to extend. *Procedural generators* [16, 60] scale environment count but remain constrained by hand-designed templates and rules. *LLM-based scene synthesizers* [89, 35, 83, 100, 55] enable open-ended diversity, but output static 3D content only evaluated as visual artifacts without task definitions, agent interfaces, or learning signals. SIMWORLD STUDIO combines the photorealism of hand-built UE5 platforms with the open-ended diversity of agentic generation, and closes the loop between scene generation and embodied training: every generated scene is exported through a standard Gym interface and adapted based on downstream agent feedback (Table 3).

Agent–environment co-evolution. Prior work on *unsupervised environment design* edits parametric environments to keep difficulty near the agent's frontier [17, 37, 53, 74, 62]; LLM-based extensions such as EnvGen and Eureka revise reward functions, configurations, or task programs from agent feedback [28, 49, 21, 44], but only tune parameters of pre-built simulators rather than construct scenes. Closer to our setting, EnvGen [94] co-trains an LLM generator with an RL agent in a controlled gridworld, whereas SIMCODER synthesizes full photorealistic UE5 scenes from scratch through engine-level tool calls and reusable skills; and Agent-World [18] mines MCP databases for *digital*-agent training, whereas our generated environments expose RGB-D observations, agent pose, and physical reward through a standard Gym contract for *embodied* policies. This gap is especially pronounced for embodied training, where useful environments must be not only adaptive but also photorealistic, physically plausible, and scalable. See additional related work in Appendix D.

5 Conclusion

We presented SIMWORLD STUDIO, a platform that overcomes the bottleneck of static scene generation by synthesizing scalable, interactive 3D environments for embodied learning. Driven by a self-evolving coding agent, SIMWORLD STUDIO automatically translates prompts into Gymnasium-compatible worlds and adapts their difficulty based on the embodied agent's performance. Our results demonstrate that this closed-loop co-evolution prevents training saturation and boosts zero-shot generalization, establishing a self-improving paradigm for embodied AI research.

References

[1] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts,

Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2026. URL https://arxiv.org/abs/2507.19457.

[2] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir R. Zamir. On evaluation of embodied navigation agents, 2018. URL https://arxiv.org/abs/1807.06757.

[3] Anthropic. Introducing the model context protocol. https://www.anthropic.com/news/model-context-protocol, 2024. Published November 25, 2024. Accessed: 2026-05-03.

[4] Anthropic. Claude code by anthropic. https://www.anthropic.com/product/claude-code, 2026. Accessed: 2026-05-07.

[5] Anthropic. Introducing claude opus 4.6. https://www.anthropic.com/news/claude-opus-4-6, February 2026. Accessed: 2026-05-07.

[6] Anthropic. Claude sonnet 4.6. https://www.anthropic.com/claude/sonnet, February 2026. Accessed: 2026-05-07.

[7] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

[8] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. Objectnav revisited: On evaluation of embodied agents navigating to objects. *CoRR*, abs/2006.13171, 2020. URL https://arxiv.org/abs/2006.13171.

[9] Aleksey Bokhovkin, Quan Meng, Shubham Tulsiani, and Angela Dai. Scenefactor: Factored latent 3d diffusion for controllable 3d scene generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 628–639, 2025.

[10] Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *Forty-first International Conference on Machine Learning*, 2024.

[11] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers, 2024.

[12] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

[13] Qian Cheng, Xin Cong, Zhong Zhang, Yesai Wu, Yankai Lin, Xie Zhiyuan, Liu Zhiyuan, and Sun Maosong. Creator: Tool creation for disentangling abstract and concrete reasoning of large language models, 2023.

[14] Zhili Cheng, Zhitong Wang, Jinyi Hu, Shengding Hu, An Liu, Yuge Tu, Pengkai Li, Lei Shi, Zhiyuan Liu, and Maosong Sun. LEGENT: Open platform for embodied agents. In Yixin Cao, Yang Feng, and Deyi Xiong, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 335–345, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-demos.32. URL https://aclanthology.org/2024.acl-demos.32/.

[15] Jaeyoung Chung, Suyoung Lee, Hyeongjin Nam, Jaerin Lee, and Kyoung Mu Lee. Luciddreamer: Domain-free generation of 3d gaussian splatting scenes, 2023. URL https://arxiv.org/abs/2311.13384.

[16] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Jordi Salvador, Kiana Ehsani, Winson Han, Eric Kolve, Ali Farhadi, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation, 2022. URL https://arxiv.org/abs/2206.06994.

[17] 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.

[18] Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, Jiajie Jin, Yutao Zhu, Hanbin Wang, Fangyu Lei, Qinyu Luo, Mingyang Chen, Zehui Chen, Jiazhan Feng, Ji-Rong Wen, and Zhicheng Dou. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence, 2026. URL https://arxiv.org/abs/2604.18292.

[19] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator, 2017. URL https://arxiv.org/abs/1711.03938.

[20] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model, 2023. URL https://arxiv.org/abs/2303.03378.

[21] 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. *arXiv preprint arXiv:2405.15568*, 2024. URL https://arxiv.org/abs/2405.15568.

[22] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge, 2022. URL https://arxiv.org/abs/2206.08853.

[23] Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, et al. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*, 2025. URL https://arxiv.org/abs/2508.07407.

[24] Tianqing Fang, Hongming Zhang, Zhisong Zhang, Kaixin Ma, Wenhao Yu, Haitao Mi, and Dong Yu. Webevolver: Enhancing web agent self-improvement with co-evolving world model. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 8970–8986, 2025.

[25] Chuang Gan, Jeremy Schwartz, Seth Alter, Damian Mrowca, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, Kuno Kim, Elias Wang, Michael Lingelbach, Aidan Curtis, Kevin Feigelis, Daniel M. Bear, Dan Gutfreund, David Cox, Antonio Torralba, James J. DiCarlo, Joshua B. Tenenbaum, Josh H. McDermott, and Daniel L. K. Yamins. Threedworld: A platform for interactive multi-modal physical simulation, 2021. URL https://arxiv.org/abs/2007.04954.

[26] Chen Gao, Baining Zhao, Weichen Zhang, Jun Zhang, Jinzhu Mao, Zhiheng Zheng, Fanhang Man, Jianjie Fang, Zile Zhou, Jinqiang Cui, Xinlei Chen, and Yong Li. Embodiedcity: A benchmark platform for embodied agent in real-world city environment. *arXiv preprint*, 2024.

[27] Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025. URL https://arxiv.org/abs/2507.21046.

[28] Jiacheng Guo, Ling Yang, Peter Chen, Qixin Xiao, Yinjie Wang, Xinzhe Juan, Jiahao Qiu, Ke Shen, and Mengdi Wang. Genenv: Difficulty-aligned co-evolution between llm agents and environment simulators. *arXiv preprint arXiv:2512.19682*, 2025. URL https://arxiv.org/abs/2512.19682.

[29] Yanjiang Guo, Tony Lee, Lucy Xiaoyang Shi, Jianyu Chen, Percy Liang, and Chelsea Finn. Vlaw: Iterative co-improvement of vision-language-action policy and world model. *arXiv preprint arXiv:2602.12063*, 2026. URL https://arxiv.org/abs/2602.12063.

[30] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Hongming Zhang, Tianqing Fang, Zhenzhong Lan, and Dong Yu. Openwebvoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27545–27564, 2025.

[31] Lukas Höllein, Ang Cao, Andrew Owens, Justin Johnson, and Matthias Nießner. Text2room: Extracting textured 3d meshes from 2d text-to-image models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7909–7920, 2023.

[32] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*, 2023.

[33] Yicong Hong, Yiqun Mei, Chongjian Ge, Yiran Xu, Yang Zhou, Sai Bi, Yannick Hold-Geoffroy, Mike Roberts, Matthew Fisher, Eli Shechtman, et al. Relic: Interactive video world model with long-horizon memory. *arXiv preprint arXiv:2512.04040*, 2025. URL https://arxiv.org/abs/2512.04040.

[34] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024. URL https://arxiv.org/abs/2408.08435.

[35] Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A. Ross, Cordelia Schmid, and Alireza Fathi. Scenecraft: An llm agent for synthesizing 3d scene as blender code, 2024. URL https://arxiv.org/abs/2403.01248.

[36] Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. 2025. URL https://arxiv.org/abs/2508.05004.

[37] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized Level Replay, 2020.

[38] Zhennan Jiang, Shangqing Zhou, Yutong Jiang, Zefang Huang, Mingjie Wei, Yuhui Chen, Tianxing Zhou, Zhen Guo, Hao Lin, Quanlu Zhang, Yu Wang, Haoran Li, Chao Yu, and Dongbin Zhao. Wovr: World models as reliable simulators for post-training vla policies with rl. *arXiv preprint arXiv:2602.13977*, 2026. URL https://arxiv.org/abs/2602.13977.

[39] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023. URL https://arxiv.org/abs/2310.06770.

[40] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai, 2022. URL https://arxiv.org/abs/1712.05474.

[41] Zhengfei Kuang, Rui Lin, Long Zhao, Gordon Wetzstein, Saining Xie, and Sanghyun Woo. Vulcan: Tool-augmented multi agents for iterative 3d object arrangement, 2026. URL https://arxiv.org/abs/2512.22351.

[42] Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, et al. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks. *arXiv preprint arXiv:2108.03272*, 2021. URL https://arxiv.org/abs/2108.03272.

[43] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Wensi Ai, Benjamin Martinez, Hang Yin, Michael Lingelbach, Minjune Hwang, Ayano Hiranaka, Sujay Garlanka, Arman Aydin, Sharon Lee, Jiankai Sun, Mona Anvari, Manasi Sharma, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Yunzhu Li, Silvio Savarese, Hyowon Gweon, C. Karen Liu, Jiajun Wu, and Li Fei-Fei. Behavior-1k: A human-centered, embodied ai benchmark with 1,000 everyday activities and realistic simulation, 2024. URL https://arxiv.org/abs/2403.09227.

[44] 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. URL https://arxiv.org/abs/2411.01775.

[45] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023. URL https://arxiv.org/abs/2308.03688.

[46] Xiaokang Liu, Zechen Bai, Hai Ci, Kevin Yuchen Ma, and Mike Zheng Shou. World-vla-loop: Closed-loop learning of video world model and vla policy. *arXiv preprint arXiv:2602.06508*, 2026. URL https://arxiv.org/abs/2602.06508.

[47] Zishan Liu, Zecong Tang, RuoCheng Wu, Xinzhe Zheng, Jingyu Hu, Ka-Hei Hui, Haoran Xie, Bo Dai, and Zhengzhe Liu. Imagine a city: Citygenagent for procedural 3d city generation, 2026. URL https://arxiv.org/abs/2602.05362.

[48] Calvin Luo, Zilai Zeng, Mingxi Jia, Yilun Du, and Chen Sun. Self-adapting improvement loops for robotic learning. *arXiv preprint arXiv:2506.06658*, 2025. URL https://arxiv.org/abs/2506.06658.

[49] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023. URL https://arxiv.org/abs/2310.12931.

[50] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

[51] NVIDIA et al. Cosmos world foundation model platform for physical ai. *arXiv preprint arXiv:2501.03575*, 2025. URL https://arxiv.org/abs/2501.03575.

[52] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023. URL https://arxiv.org/abs/2304.03442.

[53] 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*, pages 17473–17498. PMLR, 2022.

[54] Despoina Paschalidou, Amlan Kar, Maria Shugrina, Karsten Kreis, Andreas Geiger, and Sanja Fidler. Atiss: Autoregressive transformers for indoor scene synthesis. *Advances in neural information processing systems*, 34:12013–12026, 2021.

[55] Nicholas Pfaff, Thomas Cohn, Sergey Zakharov, Rick Cory, and Russ Tedrake. Scenesmith: Agentic generation of simulation-ready indoor scenes. *arXiv preprint arXiv:2602.09153*, 2026. URL https://arxiv.org/abs/2602.09153.

[56] Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander William Clegg, Michal Hlavac, So Yeon Min, Vladimír Vondruš, Theophile Gervet, Vincent-Pierre Berges, John M. Turner, Oleksandr Maksymets, Zsolt Kira, Mrinal Kalakrishnan, Jitendra Malik, Devendra Singh Chaplot, Unnat Jain, Dhruv Batra, Akshara Rai, and Roozbeh Mottaghi. Habitat 3.0: A co-habitat for humans, avatars and robots, 2023. URL https://arxiv.org/abs/2310.13724.

[57] Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024. URL https://arxiv.org/abs/2411.02337.

[58] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL https://arxiv.org/abs/2307.16789.

[59] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

[60] Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, et al. Infinite photorealistic worlds using procedural generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12630–12641, 2023.

[61] Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*, 2025. URL https://arxiv.org/abs/2504.15228.

[62] Mikayel Samvelyan, Akbir Khan, Michael Dennis, Minqi Jiang, Jack Parker-Holder, Jakob Foerster, Roberta Raileanu, and Tim Rocktäschel. Maestro: Open-ended environment design for multi-agent reinforcement learning. *arXiv preprint arXiv:2303.03376*, 2023. URL https://arxiv.org/abs/2303.03376.

[63] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL https://arxiv.org/abs/2302.04761.

[64] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[65] Ansh Kumar Sharma, Yixiang Sun, Ninghao Lu, Yunzhe Zhang, Jiarao Liu, and Sherry Yang. World-gymnast: Training robots with reinforcement learning in a world model, 2026. URL https://arxiv.org/abs/2602.02454.

[66] 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.

[67] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning, 2021. URL https://arxiv.org/abs/2010.03768.

[68] Kabir Swain, Sijie Han, Ayush Raina, Jin Zhang, Shuang Li, Michael Stopa, and Antonio Torralba. Virtualenv: A platform for embodied ai research, 2026. URL https://arxiv.org/abs/2601.07553.

[69] Jiapeng Tang, Yinyu Nie, Lev Markhasin, Angela Dai, Justus Thies, and Matthias Nießner. Diffuscene: Denoising diffusion models for generative indoor scene synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20507–20518, 2024.

[70] GigaBrain Team, Boyuan Wang, Bohan Li, Chaojun Ni, et al. Gigabrain-0.5m*: a vla that learns from world model-based reinforcement learning. *arXiv preprint arXiv:2602.12099*, 2026. doi: 10.48550/arXiv.2602.12099. URL https://arxiv.org/abs/2602.12099.

[71] Lev S. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, Cambridge, MA, 1978.

[72] 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, 2023.

[73] Hanqing Wang, Jiahe Chen, Wensi Huang, Qingwei Ben, Tai Wang, Boyu Mi, Tao Huang, Siheng Zhao, Yilun Chen, Sizhe Yang, Peizhou Cao, Wenye Yu, Zichao Ye, Jialun Li, Junfeng Long, Zirui Wang, Huiling Wang, Ying Zhao, Zhongying Tu, Yu Qiao, Dahua Lin, and Jiangmiao Pang. Grutopia: Dream general robots in a city at scale, 2024. URL https://arxiv.org/abs/2407.10943.

[74] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. POET: Open-Ended Coevolution of Environments and Their Optimized Solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 142–151. ACM, 2019. doi: 10.1145/3321707.3321799.

[75] Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeff Clune, and Kenneth O Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *Proceedings of the 37th International Conference on Machine Learning*, pages 9940–9951. PMLR, 2020. URL http://proceedings.mlr.press/v119/wang20l/wang20l.pdf.

[76] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*, 2024.

[77] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024. URL https://arxiv.org/abs/2407.16741.

[78] 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*, 2023. URL https://arxiv.org/abs/2311.01455.

[79] Zhiruo Wang, Daniel Fried, and Graham Neubig. Trove: Inducing verifiable and efficient toolboxes for solving programmatic tasks. *arXiv preprint arXiv:2401.12869*, 2024. URL https://arxiv.org/abs/2401.12869.

[80] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*, 2024.

[81] Wayne Wu, Honglin He, Jack He, Yiran Wang, Chenda Duan, Zhizheng Liu, Quanyi Li, and Bolei Zhou. Metaurban: An embodied ai simulation platform for urban micromobility, 2024. URL https://arxiv.org/abs/2407.08725.

[82] Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, et al. Agentgym: Evaluating and training large language model-based agents across diverse environments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27914–27961, 2025.

[83] Hongchi Xia, Xuan Li, Zhaoshuo Li, Qianli Ma, Jiashu Xu, Ming-Yu Liu, Yin Cui, Tsung-Yi Lin, Wei-Chiu Ma, Shenlong Wang, Shuran Song, and Fangyin Wei. Sage: Scalable agentic 3d scene generation for embodied ai, 2026. URL https://arxiv.org/abs/2602.10116.

[84] Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. *arXiv preprint arXiv:2309.11489*, 2023. URL https://arxiv.org/abs/2309.11489.

[85] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

[86] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[87] John Yang, Carlos E Jimenez, Alex L Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R Narasimhan, et al. Swe-bench multimodal: Do ai systems generalize to visual software domains? *arXiv preprint arXiv:2410.03859*, 2024. URL https://arxiv.org/abs/2410.03859.

[88] Sherry Yang, Yilun Du, Kamyar Ghasemipour, Jonathan Tompson, Leslie Kaelbling, Dale Schuurmans, and Pieter Abbeel. Learning interactive real-world simulators. *arXiv preprint arXiv:2310.06114*, 2023. URL https://arxiv.org/abs/2310.06114.

[89] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, Chris Callison-Burch, Mark Yatskar, Aniruddha Kembhavi, and Christopher Clark. Holodeck: Language guided generation of 3d embodied ai environments, 2024. URL https://arxiv.org/abs/2312.09067.

[90] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. URL https://arxiv.org/abs/2210.03629.

[91] Xiaokang Ye, Jiawei Ren, Yan Zhuang, Xuhong He, Yiming Liang, Yiqing Yang, Mrinaal Dogra, Xianrui Zhong, Eric Liu, Kevin Benavente, Rajiv Mandya Nagaraju, Dhruv Sharma, Ziqiao Ma, Tianmin Shu, Zhiting Hu, and Lianhui Qin. Simworld: An open-ended simulator for agents in physical and social worlds. In *Advances in Neural Information Processing Systems*, 2025.

[92] Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursive self-improvement. *arXiv preprint arXiv:2410.04444*, 2024. URL https://arxiv.org/abs/2410.04444.

[93] Hong-Xing Yu, Haoyi Duan, Charles Herrmann, William T. Freeman, and Jiajun Wu. Wonderworld: Interactive 3d scene generation from a single image, 2025. URL https://arxiv.org/abs/2406.09394.

[94] Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. Envgen: Generating and adapting environments via llms for training embodied agents, 2024. URL https://arxiv.org/abs/2403.12014.

[95] Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojian Zhong, Aoyan Li, et al. Multi-swe-bench: A multilingual benchmark for issue resolving. *arXiv preprint arXiv:2504.02605*, 2025. URL https://arxiv.org/abs/2504.02605.

[96] Guangyao Zhai, Evin Pınar Örnek, Shun-Cheng Wu, Yan Di, Federico Tombari, Nassir Navab, and Benjamin Busam. Commonscenes: Generating commonsense 3d indoor scenes with scene graph diffusion, 2023. URL https://arxiv.org/abs/2305.16283.

[97] Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025. URL https://arxiv.org/abs/2505.22954.

[98] Qihang Zhang, Chaoyang Wang, Aliaksandr Siarohin, Peiye Zhuang, Yinghao Xu, Ceyuan Yang, Dahua Lin, Bolei Zhou, Sergey Tulyakov, and Hsin-Ying Lee. Scenewiz3d: Towards text-guided 3d scene composition. *arXiv preprint arXiv:2312.08885*, 2023. URL https://arxiv.org/abs/2312.08885.

[99] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025. URL https://arxiv.org/abs/2510.04618.

[100] Yi Zhang, Yunshuang Wang, Zeyu Zhang, and Hao Tang. Code2worlds: Empowering coding llms for 4d world generation. *arXiv preprint arXiv:2602.11757*, 2026. URL https://arxiv.org/abs/2602.11757.

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

[102] Fangwei Zhong, Kui Wu, Churan Wang, Hao Chen, Hai Ci, Zhoujun Li, and Yizhou Wang. Unrealzoo: Enriching photo-realistic virtual worlds for embodied ai, 2025. URL https://arxiv.org/abs/2412.20977.

[103] Qinhong Zhou, Hongxin Zhang, Xiangye Lin, Zheyuan Zhang, Yutian Chen, Wenjun Liu, Zunzhe Zhang, Sunli Chen, Lixing Fang, Qiushi Lyu, Xinyu Sun, Jincheng Yang, Zeyuan Wang, Bao Chi Dang, Zhehuan Chen, Daksha Ladia, Quang Vinh Dang, Jiageng Liu, and Chuang Gan. Virtual community: An open world for humans, robots, and society, 2026. URL https://arxiv.org/abs/2508.14893.

[104] Yan Zhuang, Jiawei Ren, Xiaokang Ye, Jianzhi Shen, Ruixuan Zhang, Tianai Yue, Muhammad Faayez, Xuhong He, Ziqiao Ma, Lianhui Qin, Zhiting Hu, and Tianmin Shu. Simworld-robotics: Synthesizing photorealistic and dynamic urban environments for multimodal robot navigation and collaboration, 2026. URL https://arxiv.org/abs/2512.10046.

[105] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024. URL https://arxiv.org/abs/2406.15877.

[106] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2165–2183. PMLR, 2023.

A Limitation

A current limitation is that the effectiveness of SIMWORLD STUDIO is still bounded by the capability of the underlying coding agent. In particular, generating complex 3D environments requires strong spatial reasoning: the agent must understand object placement, geometric constraints, physical plausibility, navigability, and long-range layout consistency. While the current system can already produce useful scenes, failures may still occur when the task requires fine-grained spatial planning or precise multi-object arrangement. Improving spatial reasoning for coding agents, especially in 3D interactive environments, is therefore an important direction for future work.

B Broader Impact

SIMWORLD STUDIO has the potential to improve productivity in 3D environment creation, especially for complex engines such as Unreal Engine where existing coding-agent support remains limited. By allowing coding agents to directly edit, validate, and reuse scene-building skills, the system can reduce repetitive engineering effort and make interactive environment construction more accessible to researchers and developers. It also provides a scalable route for embodied AI research, where the lack of diverse, controllable, and interactive environments is often a major bottleneck.

At the same time, more capable automated scene-generation tools may affect existing workflows in game development, simulation design, and digital-content production. Some routine environment-authoring tasks could become increasingly automated, which may shift the role of human creators from manual construction toward supervision, design specification, and quality control. We believe such systems should be developed as assistive tools that augment human creativity and engineering productivity, while preserving human oversight over artistic direction, safety, and deployment decisions.

Potential misuse includes generating restricted, unsafe, or policy-violating simulated environments, using automated scene construction for surveillance-like or tactical planning scenarios, or recombining licensed assets outside their permitted terms. Our release is intended for research use. We do not release trained harmful policies, scraped personal data, or human-subject datasets. The Python execution interface is local/self-hosted, and users are expected to follow the licenses and deployment policies of the engine, model providers, and asset libraries.

C SIMWORLD STUDIO: Platform Details

C.1 MCP Tool Reference

Table 4 lists all 14 MCP tools exposed by the SIMWORLD STUDIO server. Tools are organized into four functional groups: actor management, environment and asset management, scene evaluation, and a Python escape hatch for operations not covered by the predefined API.

C.2 SIMCODER Generation Pipeline

Figure 2 illustrates the overall SIMCODER architecture. Here we describe each stage of the generation pipeline in detail.

Stage 1: Context acquisition. SIMCODER begins every generation episode by querying its context through tool calls rather than a manually engineered prompt. It calls `list_assets` to enumerate available mesh categories and asset counts, queries the `SkillRegistry` to retrieve applicable skills (e.g., *building-placement*, *city-layout*), and calls `setup_environment` to initialize the scene. This tool-driven context acquisition ensures that SIMCODER always operates with an accurate, up-to-date view of its action space.

Stage 2: Layout planning. Given the natural-language prompt and optional reference image, SIMCODER drafts a high-level layout plan specifying spatial zones (e.g., street grid, park, commercial row), asset categories per zone, and approximate density. For image-guided generation (S2), the plan is anchored to spatial structure inferred from the reference. Layout planning is performed in-context; the plan is not externalized but is used to sequence subsequent tool calls.

Table 4: **SIMWORLD STUDIO MCP server tool reference.** * = required parameter.

Group	Tool	Purpose	Key Parameters
Actor management	`spawn_blueprint_actor`	Spawn a CityDatabase Blueprint, such as a building, tree, vehicle, or prop; accepts full path or shorthand.	`actor_name`*, `blueprint_id`*, `location`*, `rotation`, `scale`
	`spawn_actor`	Spawn a static-mesh actor, including engine primitives or any `SM_` asset.	`name`*, `static_mesh`*, `location`*, `rotation`, `scale`
	`delete_actor`	Delete a named actor.	`name`*
	`delete_all_spawned`	Delete all session-spawned actors.	—
	`get_actors_in_level`	List every actor currently in the UE level.	—
	`find_actors_by_name`	Search actors by name pattern.	`pattern`*
	`set_actor_transform`	Move, rotate, or scale an actor.	`name`*, `location`, `rotation`, `scale`
	`take_screenshot`	Capture the UE viewport as PNG.	`filename`
Environment & assets	`setup_environment`	Initialize lighting, sky, fog, ground plane, and view distance. Must be called before spawning assets.	`ground_size`, `time_of_day`
	`list_assets`	List available SimWorld assets by category, including buildings, trees, vehicles, street furniture, and roads.	`category`
Scene evaluation	`verify_scene`	VLM verifier: captures the scene and evaluates placement against the original request; returns `PASS`, `NEEDS_IMPROVEMENT`, or `FAIL` with actionable suggestions.	`original_request`*, `focus_areas`
	`check_collisions`	Geometric overlap check: computes world-space AABBs and returns intersecting pairs with area in cm^2.	`names`, `scope`, `min_area_cm2`
	`check_vertical_support`	Detects floating objects by reporting actors whose AABB bottom is above ground and not supported by another actor.	`names`, `scope`, `ground_z`, `tolerance_cm`
Python escape hatch	`execute_python_script`	Execute arbitrary Unreal Engine Python for operations not covered by dedicated tools; successful scripts can be saved as reusable skills.	`script`*

Stage 3: Iterative scene construction. SIMCODER builds the scene incrementally, spawning and arranging actors via actor-management tools. After each spawn batch, it calls `check_collisions` and `check_vertical_support`; violations are returned inline as tool-call responses and resolved before proceeding. This per-call verification loop (§2.1) prevents error compounding across a long construction trajectory.

Stage 4: VLM-based semantic verification. After completing a logical block of construction, SIMCODER calls `verify_scene` with the original prompt. The VLM (Claude in our experiments) receives six multi-angle screenshots and the current actor list, scores semantic alignment, and returns structured feedback (PASS / NEEDS_IMPROVEMENT / FAIL) with specific issue descriptions. If the verdict is not PASS, SIMCODER performs targeted corrections and re-verifies. The generation episode terminates on PASS or after a maximum of three verification rounds.

Stage 5: Skill authoring (self-evolution). When verifier feedback reveals a failure pattern that recurs across episodes (e.g., commercial buildings colliding due to under-estimated footprints), SIMCODER authors a corrective skill or tool wrapper via `execute_python_script` and writes it to the skill directory. On the next skill-registry refresh, the new skill is indexed and becomes available to all future generations, permanently extending the agent's capability without manual intervention.

C.3 Skill Library Structure

Each skill is a Markdown document with a YAML front-matter header and an optional companion Python utility:

```
---
name: building-placement
version: 1.2
tags: [placement, spacing, buildings]
dependencies: [city-layout]
python_util: building_placement_utils.py
```

```
---
## Summary
Footprint-aware spacing rules for urban building placement ...
## Usage
Call ‘compute_building_grid(n_buildings, style)‘ to get
a list of (location, rotation) placements respecting
minimum inter-building clearance per size category.
```

The five built-in skills cover: **building-placement** (size categories, spacing tables, rotation patterns across 127 building classes), **city-layout** (grid blocks, street-facing rows, mixed-use neighborhoods), **street-furniture** (placement heuristics for trees, benches, lamps, and signs), **weather-and-mood** (lighting and time-of-day presets), and **screenshot-tour** (multi-angle capture patterns for the VLM verifier). Self-authored skills follow the same schema, ensuring seamless integration into the registry.

C.4 Detailed Running Case of SIMCODER

This section provides a detailed running case of SIMCODER editing a container-yard maze scene through iterative tool use, feedback injection, and skill accumulation. The task is to build a navigable container maze near `PlayerStart`. Across four shown rounds, SIMCODER incrementally constructs the maze entrance, extends it into a T-junction, adds a diagonal obstacle, and introduces dead-end formations. The case illustrates three core mechanisms of SIMWORLD STUDIO: external verifier feedback, collision-aware correction, and reusable skill formation.

Running Case Overview: Container-Yard Maze Editing

Run ID. `20260505_013558`
Task. Build a navigable container-yard maze near `PlayerStart` using iterative scene editing.
Model. Claude Sonnet 4.5 via Claude Code SDK.
Rounds shown. 4.
Skills learned in full run. 6.
Final recorded VLM score. 6.5/10 at Round 3.

Each round follows a fixed loop. The agent receives scene context and feedback from the previous round, plans the next spatial edit, invokes scene-editing tools, and emits newly learned skills when a reusable pattern is discovered. After the agent round ends, the pipeline captures screenshots, runs AABB collision checking, performs VLM evaluation in a separate SDK session, and injects the resulting feedback into the next prompt.

Pipeline Loop

Agent side: Claude SDK	Pipeline side: subprocess
1. Call `get_scene_summary`.	Inject VLM and collision feedback from the previous round into the prompt.
2. Apply prior VLM/collision fixes.	After the round ends, call `take_screenshot`.
3. Plan the next maze element.	Run AABB collision checking over spawned actors.
4. Call `spawn_actor` two to three times.	Run VLM evaluation in a separate SDK session.
5. Output JSON with `new_skills`.	Inject screenshot, collision, and VLM results into the next round.

Skill Library: Pre-seeded and Learned Skills

Source	Skill	Description
Pre-seeded	`get_scene_summary`	Get `PlayerStart` location and compact summaries of `Demo_` actors.
Pre-seeded	`search_ue_assets`	Search the UE asset registry by keyword for mesh paths.
Pre-seeded	`spawn_container_row`	Spawn a row of containers with configurable spacing and yaw.
Pre-seeded	`collision_check_demo_actors`	Run AABB collision checking over actors with the `Demo_` prefix.
Round 1 learned	`spawn_maze_l_junction`	L-shaped junction with two parallel walls and one perpendicular blocker.
Round 2 learned	`spawn_perpendicular_blocker`	Container at 90° yaw to close a corridor end.
Round 3 learned	`spawn_tjunction_maze_segment`	T-junction with parallel corridors and a diagonal obstacle.
Later learned	`spawn_maze_t_junction`	T-junction using perpendicular and lengthwise containers.
Later learned	`spawn_maze_dead_end`	U-shaped dead-end with one back wall and two side walls.
Later learned	`spawn_dead_end_u_formation`	Three-container U dead-end with configurable offsets.

Round 1: Maze Entrance Corridor

Agent reasoning

"I will place two parallel lengthwise containers as corridor walls and one perpendicular container as the first turn blocker. Using rotation=[0,0,0] for walls and rotation=[0,0,90] for the turn—this establishes the maze entrance."

Objects spawned.

Actor name	Mesh	Yaw	Purpose
`Demo_MazeWall_Left_XXXX`	`SM_Container12m`	0°	Left corridor wall.
`Demo_MazeWall_Right_XXXX`	`SM_Container12m`	0°	Right corridor wall.
`Demo_MazeTurn_Block_XXXX`	`SM_Container12m`	90°	First turn blocker.

Skill saved

`spawn_maze_l_junction`: spawn an L-shaped junction with two parallel walls and one perpendicular blocker.

VLM verifier feedback

Score: 4.5/10. The verifier finds a good foundation with rotation variety and industrial realism, but the maze is still too sparse. Only three containers are visible, so the structure reads as a simple corridor rather than a recognizable maze. The next round should add branching paths and small navigation obstacles.

Collision check. Not triggered in Round 1 because there are no prior maze actors to conflict with.

Saved Python Utility: `spawn_maze_l_junction`

```
# spawn_maze_l_junction
import unreal, json, time

MESH = "/Game/ContainerYard/Meshes/SM_Container12m.SM_Container12m"
BASE_X, BASE_Y, Z = -3500, -8000, 0
SPACING = 900  # corridor width

subsys = unreal.get_editor_subsystem(unreal.EditorActorSubsystem)
mesh_obj = unreal.load_asset(MESH)

configs = [
    ("Demo_WallL", BASE_X, BASE_Y, 0),
    ("Demo_WallR", BASE_X, BASE_Y + SPACING, 0),
    ("Demo_Turn", BASE_X + 1300, BASE_Y + SPACING // 2, 90),
]

spawned = []
for name, x, y, yaw in configs:
    sfx = hex(int(time.time() * 1000) % 0xFFFF)[2:].upper()
    actor = subsys.spawn_actor_from_class(
        unreal.StaticMeshActor,
        unreal.Vector(x, y, Z)
    )
    if actor:
        actor.set_actor_label(f"{name}_{sfx}")
        comp = actor.get_component_by_class(unreal.StaticMeshComponent)
        if comp:
            comp.set_static_mesh(mesh_obj)
        actor.set_actor_rotation(unreal.Rotator(yaw=yaw), False)
        spawned.append(f"{name}_{sfx}")

print(json.dumps({"spawned": spawned}))
```

Round 2: T-Junction Extension

injected context

VLM Score: 4.5/10. Add branching paths, such as a T-junction or dead-end fork, and include small navigation obstacles.

Agent reasoning

"VLM says density is too low. I will extend the corridor into a T-junction by adding two more lengthwise walls parallel to the existing corridor, plus a perpendicular blocker at the junction to force the turn decision."

Objects spawned.

Actor name	Mesh	Yaw	Purpose
`Demo_WallWest_XXXX`	`SM_Container12m`	0°	West branch wall.
`Demo_WallEast_XXXX`	`SM_Container12m`	0°	East branch wall.
`Demo_TjuncBlock_XXXX`	`SM_Container12m`	90°	T-junction blocker.
`Demo_TjuncBlock_XXXX`	`SM_Container12m`	90°	Retry spawn to confirm junction closure.

Skill saved

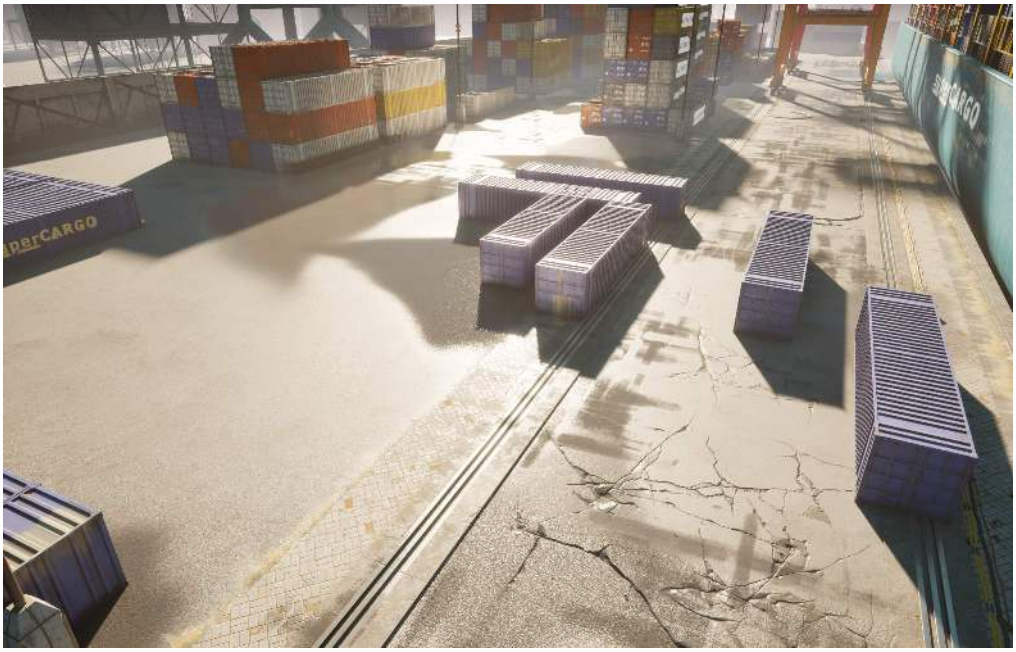
`spawn_perpendicular_blocker`: spawn a container at 90° yaw to close a corridor end or force a turn.

VLM verifier feedback

Score: 5.5/10. The verifier confirms visible rotation variety and a clearer L-turn geometry. However, the scene is still closer to a simple hallway than a branching maze. The next round should add a U-shaped dead end, close the north gap with a perpendicular blocker, and add small props inside corridors.

Collision check. One collision pair is detected between `Demo_MazeWall_01` and `Demo_EntranceWall`. The agent fixes this by deleting the overlapping actor before continuing.

Round 3: Diagonal Obstacle and North Extension



injected context

VLM Score: 5.5/10. Add a U-shaped dead end, close the north gap with a perpendicular blocker, and spawn two to three small props inside corridors.

Agent reasoning

"I will extend north with two more parallel walls, add a 45-degree diagonal container as a navigation obstacle inside the corridor to break straight-line visibility, saving as a T-junction skill for future reuse."

Objects spawned.

Actor name	Mesh	Yaw	Purpose
`Demo_NorthWallWest_XXXX`	`SM_Container12m`	0°	North corridor wall.
`Demo_NorthWallEast_XXXX`	`SM_Container12m`	0°	North corridor wall.
`Demo_DiagObstacle_XXXX`	`SM_Container12m`	45°	Diagonal obstacle inside corridor.

Skill saved

`spawn_tjunction_maze_segment`: spawn a T-junction maze segment with parallel corridors and a diagonal obstacle.

VLM verifier feedback

Score: 6.5/10. The verifier confirms rotation diversity through diagonal container placement and a wider branching footprint. Navigable aisles remain visible and the industrial context is preserved. The scene is still below full maze complexity, so future rounds should add more containers, dead-end paths, and small interior props.

Collision check. No collisions are detected.

Round 4: Dead-End Formations

Injected context

VLM Score: 6.5/10. Add U-shaped dead-end formations and close open gaps.

Agent reasoning

"I will build two U-shaped dead-ends to force the agent into backtracking decisions. This round uses six spawns, which is above the usual limit, but it is useful for increasing maze complexity."

Objects spawned.
Six containers are spawned to form two U-shaped dead-end structures. The formation uses lengthwise walls at 0° yaw and perpendicular blockers at 90° yaw.

VLM verifier feedback

Round 4 VLM feedback is not recorded because of a pipeline screenshot path error.

Collision check. No collisions are detected.

Score Progression

Round	VLM Score	Collisions	Skills Learned	Key Action
1	4.5/10	0	1	Entrance corridor with L-shaped turn.
2	5.5/10	$1 \rightarrow 0$	1	T-junction extension and collision repair.
3	6.5/10	0	1	North extension with 45° diagonal obstacle.
4	—	0	0	Dead-end formations for backtracking behavior.

Evidence of Pipeline Components

Skill reuse. Later rounds explicitly retrieve and reuse earlier skills such as `spawn_perpendicular_blocker` and `spawn_maze_l_junction`, showing that learned scene-editing patterns are persisted across rounds.

VLM-to-agent feedback loop. The VLM feedback is injected into the next prompt as hard context. For example, after Round 1, the agent receives a score of 4.5/10 and the instruction to add branching paths. The next response directly acts on this feedback by extending the corridor into a T-junction.

Collision-aware auto-fix. When the pipeline detects a collision between `Demo_MazeWall_01` and `Demo_EntranceWall`, the next prompt requires the agent to delete or move the overlapping actor before adding new objects. This turns geometric validation into an explicit correction signal.

Rotation diversity. The run gradually increases geometric complexity by using 0°, 45°, 90°, and later 135° yaw placements. This produces corridors, perpendicular blockers, diagonal obstacles, and dead-end formations.

Level Collision Checking

The collision checker runs after every round over actors with the `Demo_` prefix. It performs a lightweight AABB overlap test and injects detected collisions into the next prompt as mandatory correction context.

Level AABB Collision Check

```
# Pipeline-level AABB collision check.
for i in range(len(demo_actors)):
    for j in range(i + 1, len(demo_actors)):
        a, b = demo_actors[i], demo_actors[j]
        if (abs(a.x - b.x) < a.ex + b.ex and
            abs(a.y - b.y) < a.ey + b.ey):
            collision_pairs.append({
                "a": a.label,
                "b": b.label
            })
```

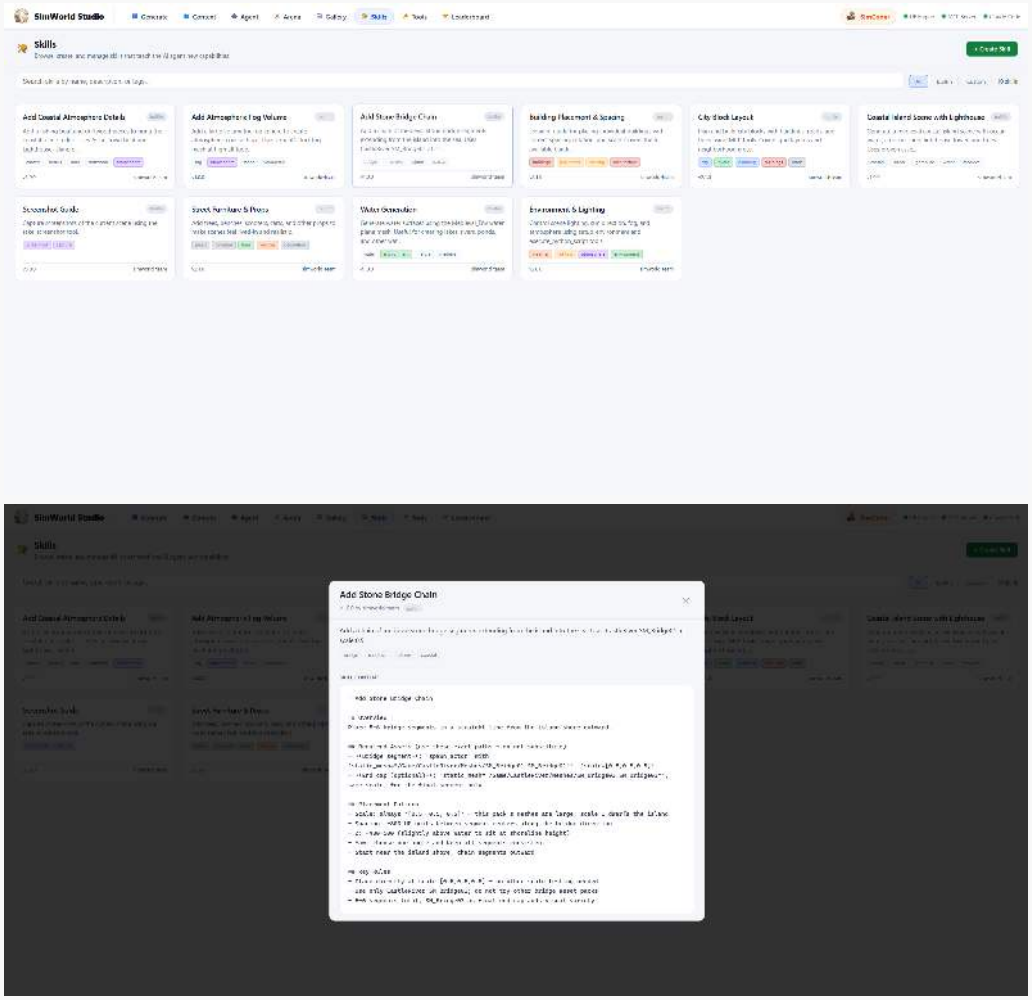
The resulting message is injected into the next round as mandatory correction context:

```
COLLISIONS: N pair(s) detected -> DELETE or MOVE before spawning.
```

C.5 SIMWORLD STUDIO's Graphic User Interface

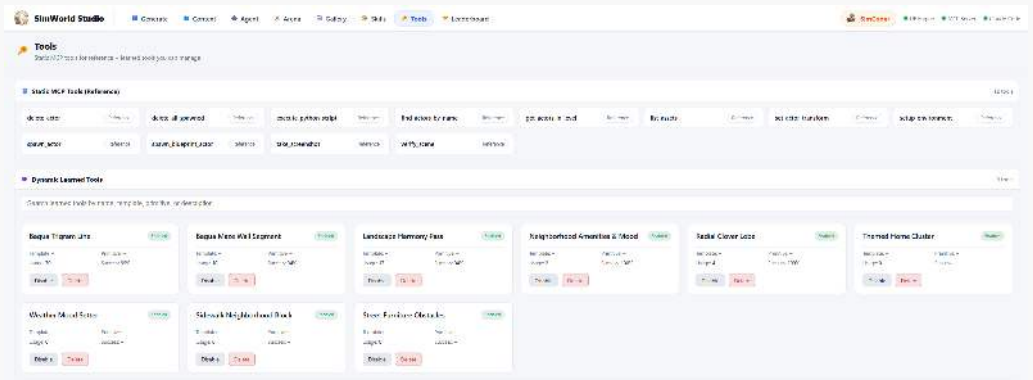
Figure 10 presents representative interface views of SIMWORLD STUDIO. The first panel shows the main studio interface in a light theme, illustrating the integrated workflow across user–agent interaction, UE rendering, asset/backend services, Gym APIs, and embodied-agent monitoring. The remaining dark-theme panels show specialized views for skill management, tool abstraction, and interactive user control inside the generated Unreal Engine environment.

Appendix Case B: Skills Panel



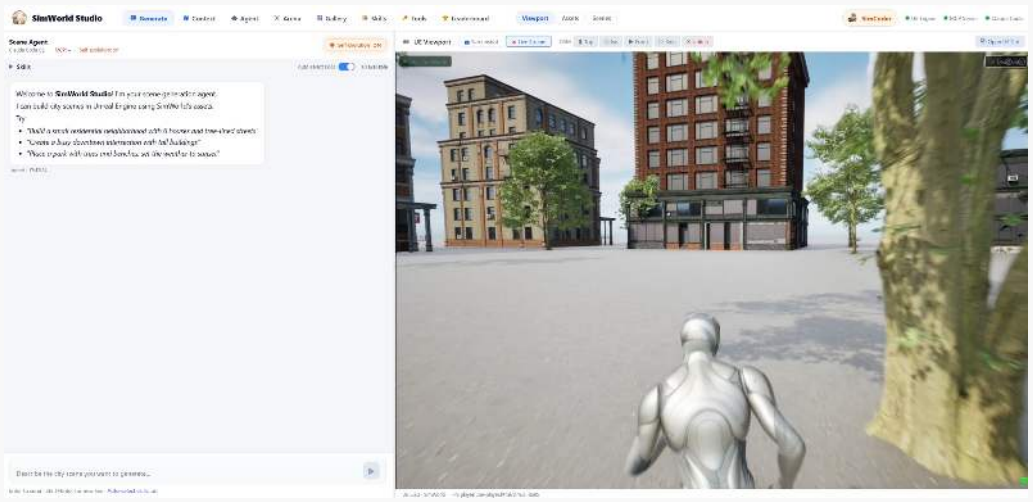
(b) Skills panel. Reusable high-level skills are organized as compositional capabilities that can be browsed, created, and reused by the coding agent across scene-generation episodes. The detailed view further exposes the internal structure of skill `add_stone_bridge_chain`, including its description, callable interface, implementation details, and execution history, enabling the agent to inspect and reuse learned abstractions across tasks.

Appendix Case C: Tools Panel



(c) Tools panel. The interface exposes both static MCP tools and dynamically learned tools, enabling the agent to combine primitive operations with reusable abstractions.

Appendix Case D: Interactive User-Control Mode



(d) Interactive user-control mode. Beyond text-based prompting, users can directly control an embodied character with keyboard input to explore, inspect, and test the generated Unreal Engine environment from an egocentric perspective.

Figure 10: **Representative interface views of SIMWORLD STUDIO.** The light-theme main interface provides an integrated workspace for user–agent interaction, UE scene rendering, asset/backend management, Gym environment APIs, and embodied-agent monitoring. The dark-theme panels further show specialized views for skill management, tool abstraction, and direct embodied interaction, allowing users to move beyond text-only prompting and interact with generated environments through controllable agents.

C.6 SIMWORLD STUDIO Running Configuration

Table 5: SIMWORLD STUDIO Minimal System Requirements.

Component	Requirement
OS	Linux; Ubuntu 20.04+ recommended
GPU	NVIDIA GPU with 8GB+ VRAM; tested on L40S, T4, and A100
NVIDIA Driver	Version 525+ with Vulkan support
Node.js	Version 18+
Python	Version 3.9+
Disk Space	40 GB free; 15 GB download + 21 GB extracted

C.7 Assets, Licenses, and Model Access

Table 6: Existing software, model APIs, and assets used in SIMWORLD STUDIO.

Asset / Software	Source / Owner	Version / Access	License / Terms
Unreal Engine	Epic Games	UE5.3.2	Unreal Engine EULA
SimWorld asset library	SimWorld[91]	Project asset library	Follows the original SimWorld asset license / terms
Qwen3.5-2B/9B/27B	Qwen Team	Hugging Face model cards	Apache-2.0 / model card terms
Claude Opus / Sonnet APIs	Anthropic	API model snapshots used in experiments	Anthropic API terms and model documentation
Claude Code	Anthropic	Agentic coding framework	Anthropic product / API terms
Gymnasium and Python packages	Upstream open-source projects	Listed in repository environment file	Respective open-source licenses

We do not scrape personal data or release human-subject data. Generated scenes are created from prompts and licensed engine/assets. Users of the released code are expected to comply with the licenses and terms of the underlying engine, model providers, and asset libraries.

D Additional Related Work

3D scene and world generation. *Generative 3D methods* synthesize explicit 3D scene representations such as object layouts, meshes, or 3D Gaussian splats. Data-driven indoor methods model object arrangements through autoregressive, diffusion-based, or scene-graph-conditioned generation [54, 69, 96, 9], while 2D-prior-based methods construct scenes through depth estimation, inpainting, and multi-view fusion [31, 15, 93, 98]. *Video-based world models* instead treat world generation as conditioned video synthesis, producing controllable visual experiences without exposing editable 3D assets, persistent scene graphs, or physics-simulator state [10, 88, 51, 33]. *Procedural and agentic pipelines* generate simulation-ready environments through procedural generators or LLM/VLM-driven agents that combine layout planning, asset retrieval or synthesis, placement, and iterative validation [16, 60, 89, 35, 83, 55, 41, 100, 47]. SimWorld Studio targets high-fidelity open-ended world generation in Unreal Engine 5, provides a standardized gym-like interface for embodied-agent training, and incorporates a persistent verifier-driven skill library for continual improvement of the coding agent.

Tool-augmented coding agents. Large language models have evolved from passive code generators into agents that reason, invoke tools, execute code, and interact with external environments. Recent work has developed this paradigm through interleaved reasoning and acting, large-scale API invocation and orchestration, executable code actions, and reflective feedback [90, 63, 58, 76, 13, 11, 66]. Evaluation has shifted from function-level code synthesis [12, 7] to compositional library use [105], repository-level patch generation with executable test harnesses [39, 95, 87], and holistic agent evaluation across operating systems and interactive environments [45, 85]. Another line develops reusable agent infrastructure, including executable skill libraries, generated tools, role-specialized agents, agent–computer interfaces, sandboxed harnesses, and multi-agent coordination frameworks for reproducible verifier-guided iteration [72, 11, 79, 32, 80, 86, 77, 82]. SimWorld Studio extends tool-augmented coding agents beyond software repair and digital automation to verified game-engine construction of simulation-ready embodied environments.

Self-evolving agents. Recent work on self-evolving agents studies closed-loop systems that autonomously improve from environmental feedback, self-generated curricula, and agent–environment co-evolution [27, 23]. In language and software domains, key mechanisms include verbal self-reflection, experience accumulation, self-editing code, and automated search over agent designs [66, 50, 101, 92, 34, 61, 99, 97]. In web and general interactive settings, agents improve through failure-driven online curricula, multi-turn reinforcement learning, or iterative co-evolution with world models [57, 30, 24, 82, 18]. In embodied and robotic settings, self-evolution appears as adaptive training loops that expand an agent's skills and training distribution through automatic curricula, executable skill libraries, and generated tasks, rewards, or environments [72, 21, 78, 49, 84, 44, 48]. SimWorld Studio is distinct in that it self-evolves the 3D world generator itself, guided by executable scene verifiers and downstream embodied-agent usability feedback, rather than primarily improving the agent, policy, or task curriculum.

Agent–environment co-evolution. In unsupervised environment design, environments are generated, selected, or edited near the agent's capability frontier to induce automatic curricula [17, 37, 53], with open-ended and multi-agent variants jointly adapting environments, solvers, or co-players [74, 75, 62]. LLM-based methods move co-evolution into code space, using language models to generate and iteratively revise environment configurations, terrains, rewards, or task programs from agent feedback or learnability signals [94, 49, 21, 44]; GenEnv [28] further co-trains the agent and a generative environment policy through RL difficulty-aligned curriculum. A parallel world-model line uses learned simulators for VLA and RL policy improvement and iteratively co-refines the world model and policy from imagined rollouts, real interactions, or policy failures [38, 65, 46, 29, 70]. R-Zero demonstrates an analogous Challenger–Solver loop in which task generation and reasoning ability co-evolve without human-curated data [36]. SimWorld Studio differs by generating simulation-ready 3D UE5 worlds via code and adapting future worlds from downstream embodied-agent feedback.

E Case Study 1

E.1 Full Results by Difficulty Level

Table 7 reports per-metric performance broken down by difficulty level (easy/medium/hard) for all four LLM backbones and three generation settings.

Table 7: **Scene generation quality across LLM backbones, settings, and difficulty levels.** All metrics $\in [0, 1]$ (↑). Bold = best per column within each setting. E/M/H = easy/medium/hard; Avg = mean across difficulties. Metric colors: quantity, physical validity, semantic, aesthetic.

		S1: Text-to-Scene							**S2: Image+Text-to-Scene**							**S3: Scene Editing**						
		Rule-Based				*VLM*			*Rule-Based*				*VLM*			*Rule-Based*				*VLM*		
LLM	**Diff.**	*Count*	*Diversity*	*No Collision*	*Gravity*	*Fidelity*	*Aesthetics*	**Avg**	*Count*	*No Collision*	*Gravity*	*In-Bounds*	*Fidelity*	*Style*	**Avg**	*Preserve*	*Edit Count*	*No Collision*	*Coherence*	*Edit Compl.*	*Layout*	**Avg**
Qwen 9B	E	.00	.70	.60	.80	.30	.20	.43	.80	.50	.70	1.0	.20	.20	.57	1.0	.00	.60	.30	.30	.30	.42
	M	.00	.50	.30	.50	.20	.10	.27	.60	.30	.50	1.0	.10	.10	.43	1.0	.00	.30	.23	.20	.20	.32
	H	.00	.30	.20	.43	.10	.10	.19	.40	.20	.40	1.0	.10	.10	.37	1.0	.00	.20	.17	.20	.10	.28
	Avg	.00	.50	.37	.58	.20	.13	.36	.60	.33	.53	1.0	.13	.13	.45	1.0	.00	.37	.23	.23	.20	.34
Qwen 27B	E	.50	.60	1.0	1.0	.60	.50	.70	1.0	.95	1.0	1.0	.50	.37	.80	1.0	.00	.99	.50	.20	**.60**	.55
	M	.00	.46	.99	1.0	.40	.30	.53	.70	.80	.90	1.0	.37	.30	.68	1.0	.00	.99	.47	.10	.50	.51
	H	.00	.32	.98	1.0	.30	.30	.48	.50	.50	.60	1.0	.27	.23	.52	1.0	.00	.99	.43	.10	.40	.49
	Avg	.17	.46	.99	1.0	.43	.37	.59	.73	.75	.83	1.0	.38	.30	.67	1.0	.00	.99	.47	.13	.50	.52
Sonnet 4	E	1.0	**1.0**	1.0	1.0	.70	.50	.87	1.0	**.99**	**1.0**	1.0	.63	.47	.85	1.0	1.0	.99	.50	**.90**	.50	**.82**
	M	.00	**.80**	**1.0**	1.0	.50	.30	.60	.80	**.85**	.90	1.0	.50	.37	.74	1.0	1.0	.98	.40	**.60**	.40	.73
	H	.00	.60	**1.0**	1.0	**.40**	**.30**	.55	.60	.60	**.70**	1.0	.37	.27	.59	1.0	1.0	.97	.30	**.50**	.30	.68
	Avg	.33	**.80**	**1.0**	1.0	.53	.37	.70	.80	.81	.87	1.0	.50	.37	.73	1.0	1.0	.98	.40	**.67**	.40	.74
Opus 4	E	**1.0**	.80	1.0	1.0	**.80**	**.60**	**.87**	**1.0**	.99	1.0	1.0	**.80**	**.60**	**.90**	1.0	1.0	.99	**.60**	.70	.60	.82
	M	**1.0**	.72	.99	1.0	**.70**	**.50**	**.82**	**.80**	.90	.95	1.0	**.60**	**.47**	**.79**	1.0	1.0	.98	**.50**	.50	**.53**	**.75**
	H	**.25**	**.64**	.98	1.0	.40	.30	**.60**	**.70**	**.70**	.80	1.0	**.47**	**.33**	**.67**	1.0	1.0	.97	**.40**	.30	**.47**	**.69**
	Avg	**.75**	.72	.99	1.0	**.63**	**.47**	**.77**	**.83**	**.86**	**.92**	1.0	**.62**	**.47**	**.79**	1.0	1.0	.98	**.50**	.50	**.53**	**.75**

E.2 Detailed Metric Specifications

Table 8 provides the complete metric inventory across all three evaluation settings.

Table 8: Complete metric inventory. "R" = Rule-based, "V" = VLM-as-Judge. **Bold** indicates metrics novel to this work.

Metric	**Type**	**S1**	**S2**	**S3**	**Adapted From**
CNT (Object Count)	R	✓	✓	—	SceneEval, Holodeck
DIV (Diversity)	R	✓	—	—	**Novel**
COL (Collision Rate)	R	✓	✓	✓	SAGE, VULCAN, SceneEval
GRAV (Gravity Validity)	R	✓	✓	—	SceneEval, VULCAN
OOB (In-Bounds Rate)	R	✓	—	—	SceneEval
PRES (Preservation)	R	—	—	✓	Adapted
ECNT (Edit Count)	R	—	—	✓	**Novel**
PF (Prompt Fidelity)	V	✓	✓	—	Code2Worlds
SRF (Spatial Fidelity)	V	✓	—	—	SceneEval
LAES (Aesthetics)	V	✓	—	—	WorldCraft, SAGE
ILC (Image Correspondence)	V	—	✓	—	Adapted
STY (Style Consistency)	V	—	✓	—	Adapted
EC (Edit Completeness)	V	—	—	✓	**Novel**
SC (Scene Coherence)	V	—	—	✓	**Novel**
LQ (Layout Quality)	V	—	—	✓	**Novel**

VLM scoring rubric. All VLM metrics use a 0–10 integer rubric with anchored descriptors: **10** = perfect realization; **7** = most elements correct, minor deviations; **5** = partial match, significant deviations; **3** = vaguely related, major elements missing; **0** = no relationship. Scores are normalized to $[0, 1]$ by dividing by 10 for aggregation. The VLM receives the original prompt, six multi-angle screenshots, and the scene graph as context.

Rule-based metric details. **Collision detection** uses axis-aligned bounding box (AABB) overlap tests on all non-environment actor pairs where both actors have extent > 100 units. **Gravity checking** tests whether each actor's bounding box bottom face lies within ± 200 units of the ground plane ($z = 0$). **In-bounds checking** tests $|x|, |y| \leq 9500$ units (the 190m $\times$ 190m ground plane).

E.3 Ablation Results

Table 9: Ablation study on verification loop and self-evolving skill accumulation across four models. All methods use the same minimal system prompt without domain-specific hints. **Bold** = best per model–difficulty group. Green = improvement ≥ 0.10 over baseline; red = degradation ≥ 0.10.

Model	Method	Task	CNT↑	DIV↑	COL↑	PF↑	SRF↑	LAES↑	Tools
Qwen3.5-9B	Baseline	Easy	**1.00**	**1.00**	**1.00**	0.40	0.40	0.30	8
		Mid	**1.00**	**0.90**	0.92	0.40	0.40	0.40	21
		Hard	0.00	0.40	**1.00**	0.00	0.00	0.00	10
	+ Verify	Easy	**1.00**	0.43	**1.00**	**0.60**	**0.60**	**0.40**	8
		Mid	**1.00**	0.85	0.94	0.40	0.30	0.20	21
		Hard	**0.25**	0.54	**1.00**	0.20	0.20	0.30	129
	+ Self-Evolve	Easy	**1.00**	**1.00**	**1.00**	0.50	0.40	0.30	8
		Mid	**1.00**	0.86	0.92	**0.50**	0.40	**0.50**	22
		Hard	0.00	**1.00**	**1.00**	**0.10**	**0.10**	**0.20**	27
Qwen3.5-27B	Baseline	Easy	**1.00**	**1.00**	0.80	0.50	0.30	0.30	8
		Mid	**1.00**	**0.81**	0.99	0.40	0.40	0.30	22
		Hard	0.25	**0.50**	0.88	0.20	0.20	0.30	26
	+ Verify	Easy	**1.00**	**1.00**	0.80	0.50	0.50	0.30	18
		Mid	**1.00**	0.77	0.99	**0.50**	**0.60**	**0.50**	23
		Hard	**0.25**	0.38	**0.99**	**0.40**	**0.40**	**0.50**	37
	+ Self-Evolve	Easy	**1.00**	**1.00**	0.80	0.40	0.40	0.30	8
		Mid	**1.00**	**0.81**	**1.00**	**0.50**	0.50	0.40	22
		Hard	**0.25**	**0.50**	0.95	0.30	0.30	0.40	31
Sonnet 4	Baseline	Easy	**1.00**	**1.00**	0.67	0.40	0.20	0.20	10
		Mid	**1.00**	**0.79**	**1.00**	0.60	0.50	0.50	27
		Hard	**0.50**	**0.44**	**0.99**	0.40	0.30	0.30	60
	+ Verify	Easy	0.00	**1.00**	**1.00**	0.40	0.30	0.30	21
		Mid	**1.00**	0.65	**1.00**	**0.60**	**0.60**	0.50	29
		Hard	0.25	0.36	**0.99**	**0.60**	**0.50**	**0.60**	73
	+ Self-Evolve	Easy	**1.00**	**1.00**	0.80	**0.50**	**0.50**	**0.40**	10
		Mid	**1.00**	0.78	**1.00**	0.50	0.50	0.50	26
		Hard	0.25	0.47	**0.99**	0.50	0.40	**0.50**	58
Opus 4	Baseline	Easy	**1.00**	**1.00**	0.80	0.40	0.40	0.30	11
		Mid	**1.00**	**0.85**	0.91	0.30	0.20	0.20	23
		Hard	0.25	0.37	**1.00**	0.40	0.30	0.30	89
	+ Verify	Easy	**1.00**	0.88	**1.00**	**0.70**	**0.80**	**0.50**	12
		Mid	**1.00**	0.77	0.98	**0.60**	**0.70**	**0.60**	24
		Hard	0.25	0.31	**1.00**	**0.60**	**0.50**	**0.60**	97
	+ Self-Evolve	Easy	**1.00**	**1.00**	0.80	0.00	0.40	0.40	11
		Mid	**1.00**	0.76	**1.00**	**0.60**	0.50	0.50	28
		Hard	**0.50**	**0.42**	**1.00**	0.50	0.50	0.50	62

Metrics: CNT = object count accuracy, DIV = asset diversity, COL = collision avoidance, PF = prompt fidelity (VLM), SRF = spatial relation fidelity (VLM), LAES = layout aesthetics (VLM). All $\in [0, 1]$. *Tools* = total MCP tool calls across all rounds. Self-evolving runs tasks sequentially (easy→mid→hard), accumulating skills between tasks. Verification uses up to 3 harness-driven fix rounds per task.

F Case Study 2: Experimental Details

F.1 Compute Setup for Case Study 2

For Case Study 2, we use an 8×H100 server to run the embodied-agent rollout pipeline. We host eight SIMWORLD STUDIO instances in parallel, each responsible for executing a subset of the navigation episodes. The instances are synchronized with the agent-inference workers for observation collection, action execution, reward computation, and trajectory logging. With this setup, processing the 1.2K-episode training set requires approximately 8 hours per full pass.

F.2 Experiment Design

Environment generation. Training environments are generated by SIMCODER using text-to-scene prompts spanning five urban archetypes: downtown intersections, residential neighborhoods, industrial districts, commercial avenues, and mixed-use blocks. Each environment is generated on a fixed $190\text{m} \times 190\text{m}$ ground plane and automatically exported as a Gymnasium environment via the SIMWORLD STUDIO embodied interface (§2.1). To study the effect of environmental diversity on downstream learning, we vary the number of distinct training environments from 1 to 30 while holding the total training-episode budget fixed at 200 episodes.

Episode generation. Navigation episodes are generated automatically from each environment's UE5 NavMesh via the following procedure:

1. **NavMesh computation.** UE5's NavMesh is built from the static geometry of the generated scene, producing a walkable region that inherits all obstacles and building footprints automatically.
2. **Start–goal sampling.** Start and goal positions are sampled uniformly from the walkable region. For PointNav, the goal is a 2D coordinate; for ObjectNav, the goal is a semantic category drawn from objects present in the scene.
3. **Path computation.** The geodesic shortest path between start and goal is computed via NavMesh A* search, recording the path waypoints and length L^*.
4. **Filtering.** An episode is retained if: (i) the path is fully reachable, (ii) the path length satisfies $3\text{m} \leq L^* \leq 20\text{m}$, and (iii) for ObjectNav, the target object is visible from at least one waypoint along the shortest path.

We generate 1,200 training episodes distributed across all training environments and 329 held-out test episodes on unseen SIMWORLD STUDIO-generated maps.

Agent observation and action space. At each step the agent receives a structured observation tuple: an RGB image (224×224), bearing to goal (degrees), geodesic distance to goal (meters), and elapsed step count. The discrete action space consists of four primitives: `move_forward` (0.25 m), `turn_left` ($15°$), `turn_right` ($15°$), and `stop`. An episode terminates on `stop` or after 40 steps.

Training protocol. Rather than gradient-based policy optimization, the agent learns in a training-free manner by accumulating and retrieving episodic memory (Appendix F.4) across training episodes. For each training episode the agent attempts navigation; the full action–observation trajectory and outcome are stored in the memory module at three granularities (step, trajectory, task). During test-time evaluation, the feedback signal is removed and the agent relies solely on retrieved memory to condition its policy.

Cross-benchmark evaluation. Beyond held-out SIMWORLD STUDIO test environments, all agents are evaluated on SimWorld-MMNav [104], an independently constructed navigation benchmark spanning three difficulty tiers (easy, medium, hard) in unseen UE5 environments. This evaluation directly measures whether navigation knowledge acquired in SIMWORLD STUDIO-generated worlds transfers beyond the training distribution.

F.3 Metric Definitions

We report four complementary metrics covering task success, path efficiency, and trajectory fidelity.

Notation. Let N denote the number of evaluation episodes. For episode i, let $P_i = (p_1, \ldots, p_T)$ be the executed trajectory and P_i^* the geodesic shortest-path reference, with lengths L_i and $L_i^* = d_i^0$ respectively. Let d_i be the geodesic distance from the agent's final position to the goal and $\delta = 1.0\,\text{m}$ the success threshold.

Success Rate (SR).

$$\mathrm{SR} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(d_i < \delta). \tag{1}$$

Binary indicator of task completion. The primary metric reported in the main paper.

Success weighted by Path Length (SPL).

$$\mathrm{SPL} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(d_i < \delta) \cdot \frac{L_i^*}{\max(L_i,\, L_i^*)}. \tag{2}$$

Penalizes inefficient paths; a successful agent that takes a much longer route than necessary scores below 1.

SoftSPL.

$$\mathrm{SoftSPL} = \frac{1}{N}\sum_{i=1}^{N} \max\!\left(0,\, 1 - \frac{d_i}{d_i^0}\right) \cdot \frac{L_i^*}{\max(L_i,\, L_i^*)}. \tag{3}$$

A continuous relaxation of SPL that rewards partial progress toward the goal rather than requiring binary success.

Normalized Dynamic Time Warping (nDTW).

$$\mathrm{nDTW} = \frac{1}{N}\sum_{i=1}^{N} \exp\!\left(-\frac{\mathrm{DTW}(P_i,\, P_i^*)}{\eta \cdot |P_i^*|}\right), \tag{4}$$

where $\mathrm{DTW}(\cdot,\cdot)$ is the dynamic time warping alignment cost, $|P_i^*|$ the number of waypoints in the reference path, and η a normalization constant ($\eta = 5$ in our experiments). nDTW measures trajectory fidelity beyond endpoint success, rewarding agents that closely follow the reference path.

F.4 Hierarchical Memory Design

The memory module extends the Generative Agents [52] and ExpeL [101] frameworks with *multi-level* updates that distill experience at step, trajectory, and task granularities, allowing strategies to be retrieved at the appropriate scope during inference.

Three-level memory structure.

- **Step-level memory (L1).** Stores fine-grained action–observation pairs within a single episode. Primarily used for within-episode self-correction (e.g., detecting a position revisit and triggering a recovery maneuver).
- **Trajectory-level memory (L2).** After each episode, the full trajectory is summarized into a compact natural-language record capturing navigation strategy, key decision points, and outcome. L2 records are indexed by environment features (obstacle density, path length tier) to enable retrieval of relevant past trajectories in new episodes.
- **Task-level memory (L3).** Periodically distills patterns across multiple L2 records into abstract navigation principles (e.g., "in dense urban environments, maintain a 45° offset from the goal bearing to avoid building corners"). L3 knowledge generalizes across environments and is injected into the system prompt as a high-priority context prefix.

Memory update. After each training episode, the module performs three sequential updates:

1. **L1 flush.** Step-level records from the completed episode are compressed and stored.
2. **L2 summarization.** An LLM summarizer condenses the episode trajectory and outcome into an L2 record, tagging it with environment metadata.
3. **L3 distillation.** Every 10 episodes, an LLM distiller identifies recurring patterns across recent L2 records and appends new L3 principles, replacing outdated ones.

Memory retrieval. At the start of each inference episode, the agent retrieves the top-k most relevant L2 records (by environment similarity) and the current L3 knowledge base. These are formatted into a structured memory block injected into the agent's context before the first action, giving the agent a head start grounded in prior experience.

F.5 Observation Modality Ablation

Table 10 reports the ablation over observation modalities. RGB-D consistently outperforms depth-only or text-only inputs across model scales, confirming that complementary geometric (depth) and semantic (RGB) cues both contribute to navigation performance. Text-only (bearing + distance scalars) provides a surprisingly competitive baseline for larger models but degrades sharply at smaller scales, suggesting that visual grounding becomes more important as in-context reasoning capacity decreases. Three models is trained on 100 episodes training set and tested on 30 episode unseen test set.

Table 10: **Ablation on observation modalities across model scales.** Each Qwen3.5 model is evaluated on ObjectNav with three vision configurations (RGB, Depth, RGB+Depth) and a text-only baseline where the agent receives goal distance and bearing scalars but no image. Metrics are success rate (SR) and SoftSPL (SS), both in $[0, 1]$ (↑). Best values within each (model scale, split, metric) group are highlighted in **bold**; tied values share the bold marker; tied zeros are not bolded.

Observation	Qwen3.5-27B				Qwen3.5-9B			
	Seen		*Unseen*		*Seen*		*Unseen*	
	SR↑	**SS↑**	**SR↑**	**SS↑**	**SR↑**	**SS↑**	**SR↑**	**SS↑**
RGB	.636±.487	.775±.227	**.833**±.408	**.721**±.373	**.136**±.347	**.279**±.332	0	.070±.113
Depth	**.659**±.479	**.784**±.198	.667±.516	.694±.356	.045±.211	.194±.308	0	.179±.255
RGB + Depth	.614±.493	.696±.308	.667±.516	.614±.386	.068±.255	.202±.290	**.167**±.408	**.223**±.331
Text only	.455±.504	.693±.267	.500±.548	.619±.369	.068±.255	.235±.276	0	.154±.162

G Case Study 3: Experimental Details

G.1 Structured Prompt Evolution

The embodied agent in Case Study 3 optimizes its navigation policy via a structured variant of GEPA [1]. Standard GEPA rewrites the entire system prompt at each update step, which risks overwriting effective strategies already in place. Our structured variant instead maintains an ordered list of prioritized decision rules $\mathcal{R}_t = [r_1, r_2, \ldots, r_k]$ that are injected verbatim into the system prompt in fixed priority order.

Rule format. Each rule r_i is a concise, actionable navigation heuristic expressed in natural language. Representative examples:

- *"If the bearing to the goal is within $\pm 30^\circ$ and no obstacle is visible, prefer* `move_forward`*."*
- *"If the same grid cell has been visited three or more times in the last ten steps, execute two consecutive* `turn_right` *actions to escape the loop."*
- *"When the reported distance to goal drops below 1.5 m, issue* `stop` *immediately."*

Rules are ordered by precedence: more specific rules (low-level reactive behaviors) appear first and override more general rules when both apply. A later rule takes effect only when no earlier rule fires.

Update mechanism. After each training episode, failure modes $\mathcal{F}_t$ are extracted from the trajectory and passed to the rule-synthesis LLM, which generates new rules $\Delta\mathcal{R}(\mathcal{F}_t)$ that address observed failures without contradicting existing rules:

$$\mathcal{R}_{t+1} = \mathcal{R}_t \cup \Delta\mathcal{R}(\mathcal{F}_t). \tag{5}$$

New rules are appended at the end of the list. A pruning step removes rules that have not been activated in the preceding 20 episodes. The list is capped at 30 active rules to prevent prompt bloat.

Failure mode taxonomy. The rule-synthesis prompt presents the full action–observation trajectory alongside a structured failure taxonomy:

1. **Directional errors.** Turning when a clear forward path exists; misalignment between bearing and action.
2. **Loop behavior.** Revisiting the same position repeatedly; oscillating between two nearby cells.

3. **Goal proximity failure.** Stopping too far from the goal; overshooting and failing to recover.
4. **Obstacle avoidance failure.** Repeated collisions with the same obstacle type (e.g., building corner).
5. **Timeout.** Reaching the 500-step budget without issuing `stop`.

For each detected failure, the LLM synthesizes one to three targeted rules. The synthesis prompt also receives the current rule list $\mathcal{R}_t$ to enforce non-contradiction.

G.2 Difficulty Curriculum Parameterization

SIMCODER controls episode difficulty via three axes: geodesic path length, initial heading offset (angular deviation from the goal direction at episode start), and dynamic obstacle density (fraction of navigable area occupied by movable obstacles).
Table 11 lists the eight difficulty levels (L0–L7) with their exact parameter ranges. Levels are designed so that each axis increases monotonically: longer paths demand extended planning, larger heading offsets require deliberate reorientation before progress can begin, and higher obstacle density increases path variability and collision risk.

Table 11: **Difficulty level parameterization.** Path length is the geodesic distance from start to goal; heading offset is the initial angular deviation from the goal direction; obstacle density is the fraction of navigable area occupied by dynamic obstacles.

Level	Path Length (cm)	Heading Offset (°)	Obstacle Density
L0	400– 800	0– 15	0.00
L1	600– 1000	0– 30	0.05
L2	800– 1400	0– 45	0.10
L3	1000– 1800	0– 60	0.15
L4	1200– 2200	0– 90	0.20
L5	1500– 2600	0– 120	0.25
L6	2000– 3000	0– 150	0.30
L7	2500– 3500	0– 180	0.35

Mastery-gating thresholds. SIMCODER advances to the next level only when the agent's rolling success rate $\bar{S}_t$—averaged over the most recent 5 epochs—exceeds a level-specific threshold τ_ℓ:

$$\ell_{t+1} = \ell_t + \mathbf{1}\big[\bar{S}_t \geq \tau_{\ell_t}\big]. \tag{6}$$

Thresholds decrease with difficulty to maintain challenge as tasks grow harder: $(\tau_0, \ldots, \tau_7) = (0.80,\ 0.75,\ 0.70,\ 0.65,\ 0.60,\ 0.55,\ 0.50,\ 0.45)$. This mastery-gating scheme mirrors the zone-of-proximal-development principle [71]: the agent is exposed to the next challenge only once it has demonstrated reliable competence at the current level.

G.3 Training and Evaluation Protocol

Co-evolution loop. Each training epoch proceeds in four steps:

1. **Episode generation.** SIMCODER generates 20 navigation episodes at the current difficulty level ℓ_t using the SIMWORLD STUDIO embodied interface.
2. **Agent rollout.** The embodied agent attempts all 20 episodes; action–observation trajectories and outcomes are recorded.
3. **Policy update.** The prompt-evolution module updates $\mathcal{R}_t$ from extracted failure modes.
4. **Difficulty advance.** The rolling success rate $\bar{S}_t$ is computed; if $\bar{S}_t \geq \tau_{\ell_t}$, SIMCODER advances to ℓ_{t+1}.

The benchmark evaluation on SimWorld-MMNav is run every 5 epochs. Training runs for 25 epochs total (500 episodes).

Baseline conditions.

- **Co-evolving environment + learning.** The full co-evolution loop as described above. SIMCODER adapts difficulty; the agent accumulates rules.

- **Fixed-difficulty environment + learning.** The same prompt-evolution mechanism, but SIMCODER holds difficulty fixed at L3 (medium) throughout. Isolates the contribution of the adaptive curriculum from the rule-learning mechanism.
- **No learning.** A fixed system prompt with no rule accumulation and no curriculum adaptation. Serves as the zero-shot LLM baseline.

All conditions use Qwen3.5-9B and the same initial system prompt with an empty rule list.

Evaluation. Final performance is measured on SimWorld-MMNav [104], a held-out benchmark of 329 episodes spanning easy, medium, and hard difficulty tiers in independently authored UE5 environments unseen during training. We report Success Rate (SR) as the primary metric.

H Prompt Examples

Text-to-Scene (Hard). *"Design a full residential neighborhood. Build two parallel streets with buildings on both sides: place 3 buildings along the north side and 3 buildings along the south side of each street (total 6 buildings in two rows). Connect the streets with a cross road. Line all roads with trees on both sides. Create a central park between the two streets with tables, couches, and trash bins. Add a fire hydrant at every street intersection. Place scooters and carts parked along the roads. Mark one intersection with road cones and road blockers as a construction zone."*

Image+Text-to-Scene (Hard). *"Build a dense urban block matching this aerial photo: multiple buildings arranged in a grid pattern with streets between them, trees lining the streets, and vehicles and street furniture throughout."* (Accompanied by a real aerial photograph of a city block.)

Scene Editing (Hard). *"Expand the plaza into a larger district: add 2 new buildings on the north side. Add roads connecting the new buildings to the existing ones. Plant 6 more trees to line the new roads. Create a marketplace area with 3 tables and 2 carts near the center. Add road cones and road blockers to mark a construction zone near the new buildings. Place fire hydrants at each road intersection and trash bins along the sidewalks."*

I Qualitative Examples

The qualitative examples in this appendix use the model snapshots available at the time of figure generation. They are provided only as visual examples and are not included in the quantitative comparisons in Table 1.

I.1 Text-to-Scene

Prompt P1

Build a downtown city-block scene at a 4-way cross intersection in Unreal Engine 5, set during the morning. The aesthetic, building heights, density, and how lively each side feels are completely your call - make it feel like a real downtown intersection.

It needs to be a proper 4-way cross, not a T-intersection, with one road running along the X axis and another cutting through it along Y. Make sure the seam where they meet is clean. Nothing should overlap the road or sidewalk - buildings stay outside the road footprint, cars stay in their lanes, pedestrians stay on sidewalks or in crosswalks. Run everything in a single script call, then take one screenshot, then stop.

Everything else is yours. How long each road is, which quadrants get tall glass towers versus shorter brick mid-rises, whether buildings are packed tight or have occasional plazas and setbacks - that's your composition. Same goes for sidewalk dressing like trees, café tables, and benches, how many people are out, car patterns, and whether the two roads feel like the same neighborhood or two different ones. Try to get specific and make real decisions rather than just defaulting to whatever looks average - the more deliberate the choices, the more the scene has a point of view.

The skill library is staged and ready to use. Before writing the script, briefly sketch out what kind of downtown this is, what the focal points are, and how the density and lighting support the mood - then write it.

Claude Opus 4.7 | Qwen 3.5-27B | Qwen 3.5-9B

Figure 11: **Qualitative Example P1.** Output scenes generated by three model backbones given the same downtown city-block intersection prompt.

I.2 Image+Text-to-Scene

Prompt P1

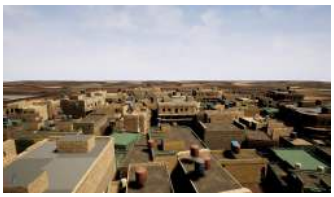
Create a large Middle Eastern desert village that matches the attached reference image. The reference shows a dense settlement of low-rise buildings spread across desert terrain - your job is to recreate that feeling using the tools available.

A few things are fixed: start by establishing the desert ground, then build the village with somewhere between 150 and 200 houses, then set up the lighting. The village builder already handles clutter internally so don't add random props on top of that. Pick your own random seed.

Everything else is a judgment call based on what you see in the reference. How many houses, how tightly packed they are, how organic or grid-like the layout feels - those are yours to decide. Think about whether rooftops should have water tanks and antennas, whether some houses get compound walls, and whether there's debris and clutter visible between buildings. Beyond the village itself, think about where dirt roads cut through, where fences sit, how many water tanks and market tents are scattered around the edges, and where they'd naturally fall. Study the reference and let it guide those decisions rather than just picking middle-of-the-road values for everything.

Reference Image

Claude Opus 4.7



Qwen 3.5-27B

Qwen 3.5-9B

Figure 12: **Qualitative Example P1.** Top: prompt and reference image. Bottom: rendered UE5 screenshots from each model backbone.

I.3 Scene Editing

Prompt P1

Build a two-sided residential street in the current scene, which already has six starting buildings and six trees. Keep one existing building, remove the others, then fill out both sides using exactly two building types, the kept one plus one other medium sized building of your choosing. Check the scene first, then remove the unwanted buildings, road is already laid but lay the sidewalk along it, and build outward. The north side incorporates the kept building with additional buildings extending west and east; the south side gets its own row facing north. Once buildings are placed, add pedestrians on both sides, vehicles in the road, and finish with a cloudy sky. Street length between 18,000 and 28,000, road width 800–1,100, sidewalk width 400–700, road center Y between -2,500 and -1,700. North row: 3 to 6 buildings on each side of the existing house. South row: 8 to 14. Pedestrians: 40 to 70 per sidewalk. Vehicles: 10 to 16. Two building types only.

Original Scene

Claude Opus 4.7

Qwen 3.5-27B

Qwen 3.5-9B

Figure 13: **Qualitative Example P1.** Top: editing prompt and the original scene prior to modification. Bottom: rendered UE5 screenshots showing each model's edited scene, built on top of the same starting configuration.

Prompt P2

Build a two-sided residential street in the current scene, which already has six starting buildings and six trees. Remove all of them, keep one, and build out both sides using a fixed pool of four small 2-to-3 storey buildings, mixed randomly per tile. Check the scene first, remove the unwanted buildings, lay the sidewalk along the road, then build outward. North side incorporates the kept building with rows extending east and west; south side gets its own row facing north. Add pedestrians on both sides, vehicles in the road, and finish with a cloudy sky. Pick real values: street length 18,000–28,000, road width 800–1,100, sidewalk width 400–700, road center Y between -2,500 and -1,700. North row: 3 to 6 buildings each side. South row: 8 to 14. Pedestrians: 40 to 70 per sidewalk. Vehicles: 10 to 16.

Original Scene

Claude Opus 4.7

Qwen 3.5-27B

Qwen 3.5-9B

Figure 14: **Qualitative Example P2.** Top: editing prompt and the original scene prior to modification. Bottom: rendered UE5 screenshots showing each model's edited scene, built on top of the same starting configuration.

I.4 Iterative Scene Development

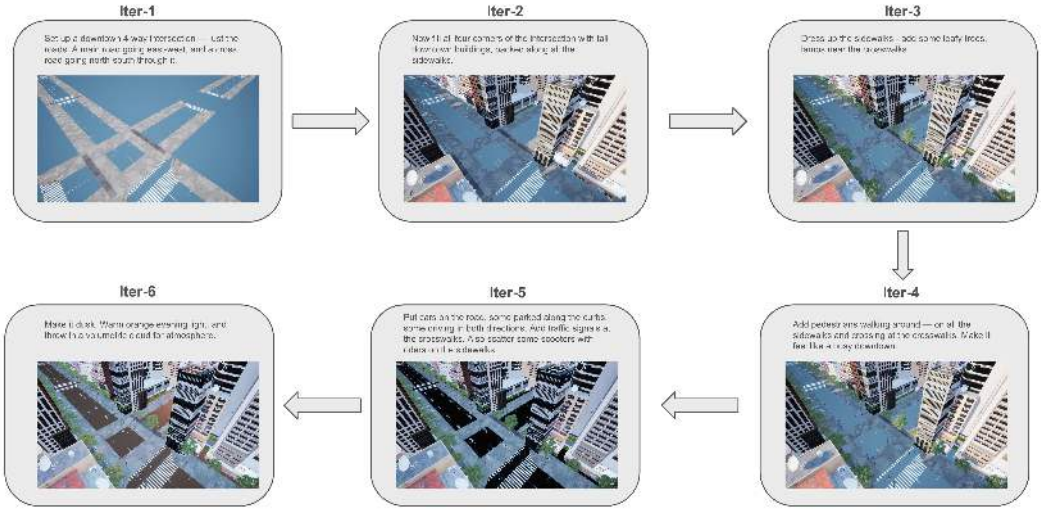


Figure 15: **Iterative scene development over six steps.** Starting from a bare 4-way road intersection (Iter-1), the scene is progressively enriched through a sequence of natural language editing instructions: tall downtown buildings are added at each corner (Iter-2), sidewalks are dressed with trees and lamps (Iter-3), pedestrians populate the crosswalks and sidewalks (Iter-4), cars, scooters, and traffic signals are introduced (Iter-5), and finally the lighting is shifted to dusk with warm orange tones and volumetric clouds (Iter-6). Each iteration modifies only what is specified without rebuilding the scene from scratch, demonstrating the platform's ability to support compositional, instruction-driven scene development.