WORLD EDITING: INTERVENING ON EXECUTABLE WORLDS AT INCREASING DEPTH

Anonymous authors
Paper under double-blind review

ABSTRACT

Interactive world models are increasingly capable of generating environments and acting within them, yet deliberately editing an existing executable world remains underexplored. We formulate **world editing** as intervening on an existing world while preserving properties that should remain unchanged, and introduce **intervention depth** as an axis describing how strongly an edit couples world entities, dynamics, and systems. We instantiate this capability through industry-grade game modding and introduce IGMWorld, together with IGMBench, a benchmark of 110 tasks and over 1.1K executable state and behavioral criteria across Minecraft and Terraria. The tasks span property, entity, dynamics, and system interventions and are evaluated through deterministic executability, behavioral, preservation, and visual checks. Frontier coding agents already exhibit substantial world-editing capability: the strongest configuration solves 78.2% of tasks under a strict task-level criterion, while criterion-level performance reaches 94.8%. Reliability generally decreases with intervention depth, and this pattern persists even among tasks with similar numbers of evaluation criteria. Most failed edits still build and load successfully, suggesting that the main difficulty is making the edited world behave as requested. Visual consistency remains a separate weakness, with all evaluated configurations below 50% joint visual pass rate. These results show that world editing is a distinct capability from world generation and interaction, and that executable games provide a practical testbed for studying it.

1 INTRODUCTION

Contemporary research largely studies two relationships between AI systems and interactive worlds: generating worlds (Bruce et al., 2024; Guo et al., 2025; Valevski et al., 2025; Savva et al., 2026; Hu et al., 2026; Marti Monso et al., 2026) and acting within worlds (Fan et al., 2022; Wang et al., 2023a; Hafner et al., 2025; Cheng et al., 2026). Many recent world models formulate generation primarily as predicting or synthesizing visual observations over time. Less studied is whether an AI system can deliberately edit an existing world: changing its entities, behaviors, or underlying systems while preserving unrelated properties. While prior work (Mao et al., 2025b) has used **world editing** for visual or geometric modifications of generated scenes, we use the term for **interventions** on an executable world itself, including its entities, dynamics, and interacting systems. We view world editing as a complementary capability to world generation and gameplay, and as a means of systematically constructing diverse, coherent world variants for training and evaluating future world models and agents.

This formulation raises two questions about the structure of world-editing capability. (1) **Are deeper interventions harder because they require coordinating more of the world, or simply because they contain more requirements?** (2) **When world edits fail, do they fail at basic executability, or after the game runs because the requested behavior is not correctly realized?** We instantiate these questions through industry-grade game modding[1], where agents intervene on commercially released games through their native modding ecosystems. To study these questions, we introduce IGMWorld and IGMBench. IGMWorld provides an executable environment in which agents modify existing game repositories, build and run their edits, and are evaluated on the resulting world rather

[1] A *mod* is a user-created modification of an existing game that changes or extends its assets, content, mechanics, or systems.

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

Table 1: World-editing taxonomy and its realization in game modding.

Level	World Intervention	Game Modding	Examples
L1	**Property Intervention**: Modify properties of existing components without changing their identity or underlying behavior.	**Parameter Editing**: Modify parameters, configurations, or appearance of existing game components.	Increase weapon damage; replace a monster's visual asset.
L2	**Entity Intervention**: Introduce new entities or content while largely preserving existing dynamics.	**Content Editing**: Add new content that conforms to existing mechanics.	Add a crafting recipe or item; add a monster or NPC.
L3	**Dynamics Intervention**: Modify rules governing entity behavior and interaction.	**Mechanic Editing**: Modify game logic governing behaviors and interactions.	Add an air-dash mechanic; add a status-effect system.
L4	**System Intervention**: Modify multiple coupled components and interactions forming a world-level system.	**System Editing**: Modify multiple interacting game subsystems.	Add a skill tree; overhaul the economy system.

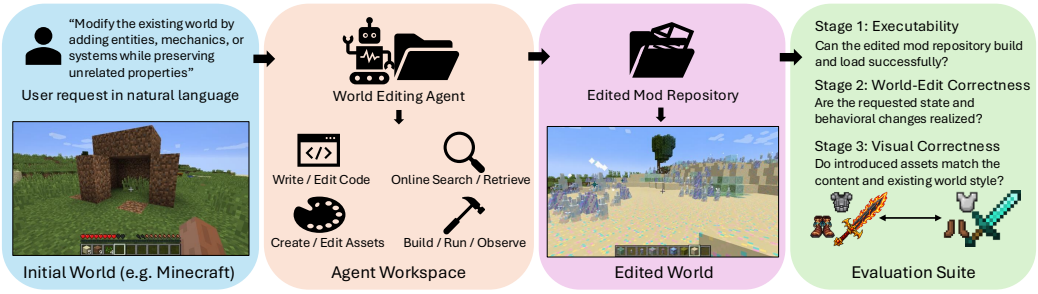


Figure 1: Overview of IGMWorld. A world-editing agent modifies a scaffold mod repository from a natural-language request and produces an edited world, which is evaluated for executability, state and behavioral correctness, and visual consistency with existing in-game assets.

than on code structure alone. IGMBench contains 110 world-editing tasks across Minecraft (Mojang, 2009) and Terraria (Re-Logic, 2011), spanning property, entity, dynamics, and system interventions, with over 1.1K executable state and behavioral evaluation criteria together with regression and visual validation. These games provide mature executable worlds whose entities, dynamics, assets, and systems can be programmatically modified and reproducibly evaluated (Fabric Development Team, 2018; tModLoader Team, 2020). We summarize our contributions as follows:

- We formulate **world editing** as intervention on an existing executable world and define **intervention depth** by the coupling among its entities, dynamics, and systems.
- We operationalize world editing through IGMWorld and IGMBench, comprising 110 tasks and over 1.1K executable evaluation criteria across Minecraft and Terraria, with deterministic state, behavioral, regression, and visual validation.
- We use this testbed to characterize current frontier agents, finding that deeper edits are generally less reliable, most failed edits still build and load successfully but behave incorrectly, and visual integration remains a separate weakness.

2 World Editing as Intervention

We formulate **world editing** as an **intervention** on an existing executable world. Let a world be abstractly represented as $W = (S, T, R, \ldots)$, where S denotes its entities and state space, T its transition dynamics, and R the rules and systems that govern interactions. Given a natural-language editing instruction, a world-editing agent transforms the original world W as

$$W \xrightarrow{\Delta_k} W' = f(W, \Delta_k), \qquad k \in \{\text{property}, \text{entity}, \text{dynamics}, \text{system}\}, \tag{1}$$

where Δ_k denotes an intervention of type k, and W' is the resulting world that realizes the requested intervention while preserving unrelated properties of W. We characterize these interventions by

Table 2: Distribution of tasks and evaluation criteria across games and world-intervention levels. Each task specifies a natural-language world-edit request and multiple state or behavioral criteria. A task succeeds only if the implementation builds and passes all its criteria.

	L1		L2		L3		L4		Total	
Game	Tasks	Criteria	Tasks	Criteria	Tasks	Criteria	Tasks	Criteria	Tasks	Criteria
Minecraft	14	82	15	144	14	169	14	226	57	621
Terraria	13	92	14	162	13	154	13	132	53	540
Total	27	174	29	306	27	323	27	358	110	**1161**

their **depth of intervention**. Property interventions (L1) modify properties of existing world components without changing their identity or underlying behavior; Entity interventions (L2) introduce new entities or content while largely preserving existing world dynamics; Dynamics interventions (L3) introduce or modify rules governing how world entities behave and interact; and System interventions (L4) modify multiple coupled components and interactions that jointly constitute a world-level system. In game-modding terminology, these correspond respectively to parameter editing, content editing, mechanic editing, and system editing.

The levels characterize the semantic scope and coupling of the requested intervention rather than prescribing which implementation components must change. Intervention depth therefore measures how strongly an edit must coordinate multiple parts of the world, not how much code it requires: a system-level edit may be small in implementation yet still need several interacting components to remain consistent. Table 1 summarizes the four levels and representative examples. Games provide a useful testbed because their worlds are both complex and executable: entities, state transitions, rules, assets, and interacting systems are concretely instantiated through code and runtime behavior. Modding further exposes an intervention interface over an already existing world, rather than asking an agent to construct a world from scratch. Agents modify the implementation, but correctness is judged in the resulting executable world. Operationally, each task provides an initial repository, a requested intervention, an executable environment, and an independent validator. IGMWorld provides the execution environment for these edits, while IGMBench provides the collection of task instances. Figure 1 illustrates this workflow.

3 Operationalizing World Editing

Agent Execution Workflow. IGMWorld exposes each task as a reproducible edit $\rightarrow$ build $\rightarrow$ run $\rightarrow$ observe loop. The agent begins with a natural-language editing request and a prepared mod scaffold, and may inspect or modify source code and visual assets; invoke the native build toolchain; launch the game server and use logs to iteratively revise its implementation. The agent is given a fixed time budget of $T = 3600$ seconds, after which the modified scaffold is submitted for evaluation. For each game, IGMWorld provides a reproducible environment with the runtime, modding toolchain, required dependencies, and a fixed-seed world. Tasks within the same game share the same runtime and toolchain setup. For each task, the agent receives the request, its natural-language requirements, the project path, the build command, a generic build-and-load script, and constraints on immutable metadata. Executable verification checks are hidden from the agent. After execution, the modified scaffold is evaluated in a separate validation environment for build, load, and world-edit correctness.

IGMBench Construction. We recruited 11 annotators with prior game-modding experience to author tasks across Minecraft and Terraria at all four intervention levels. Annotators were given the level definitions and representative examples, but were asked to write novel requests. Each task specifies the target game and modding ecosystem, an intervention level, a natural-language request, and evaluation criteria defining a correct implementation. Tasks were reviewed for clarity, feasibility, and level consistency, yielding 110 tasks. Each accepted task is compiled into an instruction view exposed to the agent and a verification view reserved for evaluation. The agent only sees the task description and a natural-language list of state and behavioral requirements. The corresponding executable checks are kept separate and used only during evaluation, so the agent never sees the task-specific grading logic. We instantiate IGMBench in Minecraft and Terraria using their na-

tive modding ecosystems, Fabric (Fabric Development Team, 2018) and tModLoader (tModLoader Team, 2020). The benchmark contains 110 tasks, with 57 in Minecraft and 53 in Terraria, distributed approximately evenly across the four intervention levels (Table 2). In total, IGMBench contains around 1.1K evaluation criteria. The tasks span diverse gameplay domains, including combat, movement, entities, crafting, world generation, progression, and multi-system modifications; Appendix F reports the full distribution. The median number of criteria increases from 8 at L1 to 15 at L4. Because each task requires a native build, game launch, and multimodal validation, IGMBench is kept compact while still covering both games and all four intervention levels.

3.1 Evaluation of Executable World Edits

IGMWorld evaluates world edits along three complementary dimensions: **executability** asks whether the modified world can be built and loaded. **world-edit correctness** asks whether the requested intervention Δ_k is realized in the resulting world W', including targeted checks that unrelated properties of W are preserved. For edits involving visual assets, **visual correctness** asks whether the introduced content is semantically appropriate and visually consistent with host world.

Stage 1: Executability. The submission must compile into a valid mod package and load successfully in the target game without crashing. Build and load checks are strict gates.

Stage 2: World-Edit Correctness. For submissions that build and load successfully, IGMWorld evaluates whether the requested intervention is realized through task-specific state and behavioral checks. State checks query properties of the running game, such as entity attributes, item statistics, recipes, and registrations. Behavioral checks execute controlled in-game actions and verify the resulting state transitions. Where applicable, paired regression checks compare the modified and original worlds to test whether unrelated properties remain unchanged. These checks provide targeted rather than exhaustive evidence of preservation. IGMBench contains 225 regression checks across 30 tasks, concentrated at L1; coverage and pass rates are reported in Appendix B (Table 7). Validators assess observable world behavior rather than source-code structure, so different implementations may satisfy the same intervention. Additional details of the game-specific validation infrastructure are provided in Appendix E.

Stage 3: Visual Consistency. For tasks that introduce or modify visual assets, IGMWorld separately evaluates **semantic consistency**, whether the asset depicts the requested content, and **style consistency**, whether it matches the visual language of the host world. Both checks use TPIPS (Wang et al., 2026b), a text-conditioned extension of learned perceptual metrics such as LPIPS (Zhang et al., 2018), under different text factors. Let x denote the evaluated asset and $\mathcal{I}_c = \{r_i\}$ the reference assets from category c. For a text factor f, let $\mathrm{TPIPS}(x, r_i; f)$ denote the similarity between x and reference r_i, and let $\mathrm{kNN}(x, \mathcal{I}_c; f)$ denote the k nearest category-matched references under the same factor. We define the text-conditioned consistency score $S_f(x; c)$ as

$$S_f(x; c) = \frac{1}{k} \sum_{r_i \in \mathrm{kNN}(x, \mathcal{I}_c; f)} \mathrm{TPIPS}(x, r_i; f). \tag{2}$$

For semantic consistency, the text factor f specifies the requested semantic identity or object class; for style consistency, we use the factor "art style". For each category–factor pair, we calibrate a threshold $\tau_{c,f}$ from the α-quantile of leave-one-out scores over the native reference set $\mathcal{I}_c$. An asset passes the corresponding check when $S_f(x; c) \geq \tau_{c,f}$. Category conditioning avoids comparisons across semantically incompatible assets, while k-NN aggregation accommodates legitimate variation within a category. Appendices D.2 and E provide calibration details, sensitivity analysis, human validation, and comparison with CSD (Somepalli et al., 2024).

4 Characterizing World-Editing Capability

4.1 Experimental Setup

Agent and Environment setup. We evaluate seven frontier agent configurations spanning proprietary and open-weight models. Proprietary models are evaluated using their standard or provider-recommended coding harnesses: GPT-5.6 Sol and GPT-5.6 Luna with Codex-CLI (OpenAI, 2026a), Gemini 3.5 Flash with Gemini-CLI (Google DeepMind, 2026a), and Claude Opus 4.8 with Claude

Table 3: Criterion Pass Rate (CPR) and World-Editing Success Rate (WSR) on IGMBench, broken down by world-intervention level. CPR measures the fraction of individual state and behavioral criteria satisfied, while WSR measures the fraction of world-editing tasks for which all associated criteria are satisfied. Results are aggregated across Minecraft and Terraria. Visual correctness is evaluated separately in Table 4.

	L1		L2		L3		L4		Overall	
Model	**CPR**	**WSR**	**CPR**	**WSR**	**CPR**	**WSR**	**CPR**	**WSR**	**CPR**	**WSR**
DeepSeek-V4-Pro	92.1	74.1	80.8	24.1	47.7	18.5	47.0	11.1	63.3	31.8
GLM-5.3	97.6	92.6	82.7	62.1	39.8	25.9	39.3	11.1	60.1	48.2
Kimi-K3	98.5	88.9	78.3	65.5	61.0	51.9	73.3	33.3	74.9	60.0
Gemini 3.5 Flash	87.7	77.8	86.4	55.2	83.5	37.0	68.8	59.3	81.0	57.3
Claude Opus 4.8	99.6	92.6	92.8	72.4	94.7	66.7	**89.4**	55.6	93.6	71.8
GPT-5.6 Luna	96.3	92.6	90.2	58.6	90.4	44.4	80.0	**63.0**	88.5	64.5
GPT-5.6 Sol	**99.8**	**96.3**	**93.9**	**86.2**	**98.0**	**81.5**	88.8	48.1	**94.8**	**78.2**

Code (Anthropic, 2026). We therefore compare complete agent configurations rather than model backbones alone. For open-weight models, which do not share a canonical provider-specific coding harness, we evaluate DeepSeek-V4-Pro (DeepSeek-AI et al., 2026), Kimi-K3 (Team et al., 2026), and GLM-5.3 (GLM-5-Team et al., 2026) using Hermes Agent (Nous Research, 2026), a shared coding harness for all three models. All agents have access to Internet search and an optional image-generation tool: Codex-based configurations use gpt-image (OpenAI, 2026b), while the others use nano-banana (Google DeepMind, 2026b) via MCP. Agents decide autonomously whether to use these tools. All configurations also receive the same generic game-specific modding skills and optional sprite post-processing pipeline (Appendix D), without task-specific or evaluator information. Unless otherwise noted, we use provider-recommended inference settings. The prompt is generated from the task specification and includes the request, natural-language requirements, project path, build command, a generic build-and-load script, and immutable-metadata constraints (Figure 11). We provide no task-specific API guidance, worked examples, or executable validation logic. Each run has a wall-clock budget of $T = 3600$ seconds, and each model–task pair is evaluated once. All runs use the pinned environments described in Section 3: Minecraft 1.21.1 with Fabric and Java 21, and the pinned tModLoader release with its bundled .NET runtime for Terraria. Further configuration details are provided in Appendix E (Tables 15 and 16).

Evaluation metrics. We report two metrics for functional world editing. Criterion Pass Rate (CPR) is the fraction of individual state and behavioral evaluation criteria satisfied across tasks. World-Editing Success Rate (WSR) is the fraction of tasks for which all associated state and behavioral criteria are satisfied. Thus, WSR is a strict task-level metric, whereas CPR captures partial correctness within a task. Visual consistency is evaluated separately for tasks involving introduced or modified assets. We report $\mathrm{Pass}_{\mathrm{style}}$, $\mathrm{Pass}_{\mathrm{sem}}$, and $\mathrm{Pass}_{\mathrm{both}}$, denoting the fractions of evaluated assets that pass the style, semantic, and both visual checks, respectively. Visual thresholds follow the calibration procedure in Section 3.1; calibration and sensitivity analysis are provided in Appendix E.4.

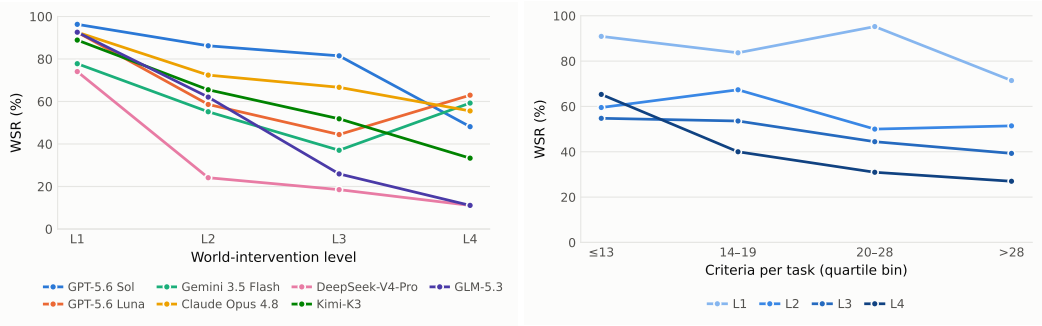
4.2 Reliability Varies with Intervention Depth

Table 3 shows that current frontier agents already exhibit substantial functional world-editing capability. Under the strict World-Editing Success Rate (WSR), the evaluated configurations solve between 31.8% and 78.2% of IGMBench tasks, with GPT-5.6 Sol reaching 78.2%. Criterion Pass Rate (CPR) is substantially higher, reaching 94.8% for GPT-5.6 Sol, indicating that many unsuccessful tasks nevertheless satisfy substantial portions of the requested behavior.

Reliability varies systematically with intervention depth. Every evaluated configuration achieves its highest WSR at L1, while performance is generally lower for deeper dynamics and system interventions, although the relative ordering of L3 and L4 varies across models and games. GPT-5.6 Sol, for example, decreases from 96.3% WSR at L1 to 48.1% at L4, while Claude Opus 4.8 decreases from 92.6% to 55.6%. At deeper levels, CPR remains substantially higher than WSR, indicating that agents often satisfy many requirements without completing the full edit. Because deeper tasks also contain more evaluation criteria, this trend could simply reflect the larger number of requirements

Table 4: Visual quality of agent-produced assets. N counts registered texture assets; vanilla textures and missing files fail all checks. $\text{Pass}_{\text{style}}$, Pass_{sem}, and $\text{Pass}_{\text{both}}$ denote style, semantic, and joint pass rates, computed using the text-conditioned consistency score $S_f(x; c)$ in Equation 2 and the calibrated category-specific thresholds. Vanilla (LOO) provides the in-game reference, with marginal style and semantic pass rates of $1 - \alpha = 0.85$ by construction.

	Minecraft				Terraria			
Model / Source	N	$\text{Pass}_{\text{style}}$	Pass_{sem}	$\text{Pass}_{\text{both}}$	N	$\text{Pass}_{\text{style}}$	Pass_{sem}	$\text{Pass}_{\text{both}}$
Vanilla (LOO)	1989	0.85	0.85	0.80	15456	0.85	0.85	0.78
DeepSeek-V4-Pro	142	0.17	0.23	0.15	113	0.16	0.23	0.11
GLM-5.3	226	0.49	0.53	0.40	73	0.32	**0.37**	**0.29**
Kimi-K3	205	**0.52**	**0.54**	**0.47**	109	0.29	0.35	0.24
Gemini 3.5 Flash	110	0.32	0.36	0.22	104	0.32	0.32	0.22
Claude Opus 4.8	127	0.34	0.42	0.23	110	**0.35**	0.34	0.25
GPT-5.6 Luna	158	0.16	0.20	0.14	110	0.18	0.36	0.12
GPT-5.6 Sol	155	0.23	0.24	0.19	99	0.22	**0.37**	0.18



(a) WSR by intervention level

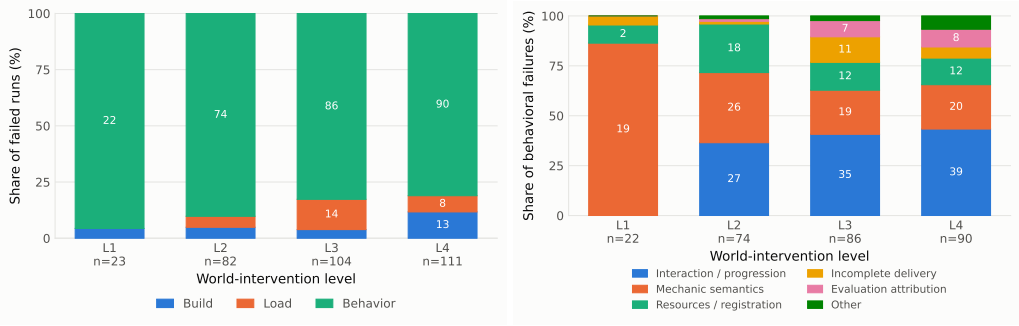
(b) WSR controlled by criterion count

Figure 2: Reliability across intervention depth. Performance differences persist within comparable criterion-count ranges, showing that intervention depth is not reducible to requirement count alone.

that must be satisfied. We therefore stratify tasks by criterion count in Figure 2. Within comparable criterion-count ranges, L1 remains substantially more reliable, while L3 and L4 generally remain below shallower interventions. This suggests that intervention depth captures structure beyond criterion count alone.

4.3 Most Failures Occur After Executability

Most unsuccessful world edits fail after reaching an executable state. Figure 3a shows that behavioral failures account for the large majority of unsuccessful runs at every intervention level, while build and load failures remain comparatively uncommon. Even at L4, where build failures become more frequent, behavioral correctness remains the dominant failure stage. This indicates that the principal bottleneck is not producing a modification that compiles and loads, but realizing the requested behavior in the executable world. The composition of behavioral failures also changes with intervention depth (Figure 3b). L1 failures are almost entirely mechanic semantics, such as a wrong value or formula. From L2 onward, interaction/progression failures, in which new content misbehaves in contact with existing systems, become the largest category and rise to 43% at L4, while resource/registration failures remain at 13–24% and do not grow with depth. The absence of interaction failures at L1 is expected by construction; the more informative pattern is the increase from L2 to L4 while resource/registration failures remain relatively stable. This is consistent with deeper edits requiring coordination across multiple interacting world components rather than a single localized change. Beyond correctness, preservation shows a different pattern. Targeted regression checks show few unintended changes on the tasks where such checks are available (Appendix B, Table 7), but coverage is concentrated at L1. Preservation under deeper interventions is therefore only partially tested and remains a limitation of the current benchmark.



(a) Failure stage by intervention level (b) Behavioral failure domain by level

Figure 3: Failure structure across intervention depth. Behavioral failures dominate across all levels. Among behavioral failures, interaction/progression failures increase from L2 to L4, while resource/registration failures remain relatively stable. Numbers inside bars denote failure counts.



Figure 4: Representative successful world edits in Minecraft (top) and Terraria (bottom) across L2–L4. From left to right, the examples introduce a new entity, add new dynamics, and modify a coupled world-level system.

4.4 How Agents Edit Worlds

Successful edits illustrate how intervention depth changes the structure of the required modification. As shown in Figure 4, entity-level edits can often be realized by introducing localized content that conforms to existing world behavior, while dynamics-level edits require new state transitions or interaction rules. System-level edits additionally coordinate multiple components such as player state, interfaces, progression, and world behavior. These examples illustrate why intervention depth reflects coupling among world components rather than implementation size alone.

Agent trajectories reveal distinct strategies for grounding world edits. Table 11 shows that GPT-5.6 Sol, GPT-5.6 Luna, and Claude Opus 4.8 inspect existing game sources in 91–99% of runs, while

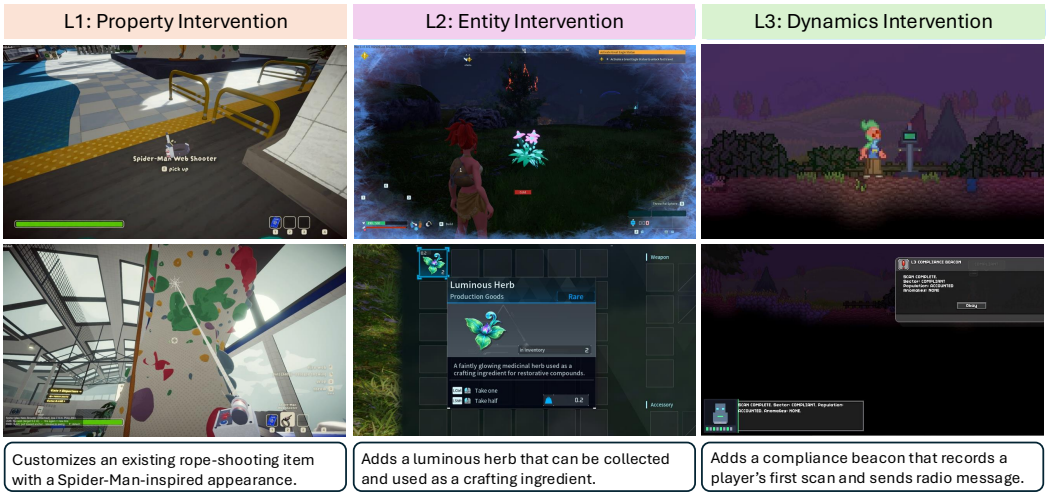


Figure 5: World editing beyond the games included in IGMBench. We show qualitative examples of property, entity, and dynamics interventions in PEAK, Palworld, and Starbound. These examples illustrate that the intervention taxonomy applies to additional executable game worlds, but they are not part of the benchmark.

rebuilding after an initial implementation is common across configurations. Strategies nevertheless differ across harnesses: Gemini relies more heavily on online search, whereas GPT and Claude more often ground their implementations in local game sources, game data, and runtime logs. Visual tasks also expose different asset-construction strategies, including adapting existing assets, using image-generation tools, and programmatic drawing.

4.5 Visual Consistency Is a Distinct Bottleneck

Visual integration remains substantially weaker than functional world editing. Table 4 shows that native game assets achieve joint style-and-semantic pass rates of 0.80 in Minecraft and 0.78 in Terraria, whereas every evaluated agent configuration remains below 0.50. The strongest joint pass rates reach 0.47 in Minecraft and 0.29 in Terraria. Qualitative comparisons in Appendix Figures 7 and 8 further illustrate how image-generated and programmatically drawn assets differ from native game art. Notably, visual performance does not follow the ranking observed for functional world editing. Configurations that are weaker on executable state and behavioral correctness can nevertheless perform competitively on visual assets, while the strongest functional agents do not dominate the visual evaluation. This weak correspondence suggests that world editing is multidimensional: successfully modifying the behavior of an executable world does not imply that newly introduced content integrates perceptually with that world.

5 Implications of World Editing

World editing provides a mechanism for constructing controlled variants of an existing executable world while preserving much of its surrounding structure. This makes intervention depth a natural axis for organizing increasingly coupled changes, from localized property edits to modifications of dynamics and interacting systems. Such controlled variation could support future work on agent training, robustness, and generalization, although we do not evaluate these downstream uses here. World editing is also complementary to executable or code-based world models: rather than constructing or inferring a world representation, it asks whether an agent can deliberately intervene on an already instantiated world while maintaining consistency with what should remain unchanged. Although IGMBench quantitatively evaluates Minecraft and Terraria, the formulation is not specific to these games. Figure 5 shows qualitative property, entity, and dynamics interventions in PEAK (Team PEAK, 2025), Palworld (Pocketpair, 2024), and Starbound (Chucklefish, 2016). These examples suggest that world editing is better viewed as a general capability over executable worlds, with games serving as a concrete and verifiable testbed rather than the boundary of the problem.

6 Related Work

World Models and Executable Worlds. World models learn to represent, generate, or simulate an environment so that an agent can predict future states and interact over time (Ha & Schmidhuber, 2018). Recent research extends this idea to interactive 2D and 3D virtual worlds generated from text, images, or other multimodal inputs (Alonso et al., 2024; Bruce et al., 2024; Wang et al., 2025; Cao et al., 2026; Mao et al., 2025b;a; Gao et al., 2026a;b). A substantial portion of this literature focuses specifically on games, training models to predict environment dynamics from pixels, actions, or structured states (Valevski et al., 2025; Savva et al., 2026; Hu et al., 2026; Marti Monso et al., 2026; Wang et al., 2026c; Zhou et al., 2026; Li et al., 2026). Most existing work focuses on generating, predicting, or simulating worlds or their behavior. World editing instead asks how an agent can deliberately intervene on an existing executable world while preserving what should remain unchanged. Recent work on code-based world models represents game or physical dynamics as executable programs, making states, transition rules, and governing mechanisms explicit and controllable (Tang et al., 2024; Lehrach et al., 2025; Serapio et al., 2026; Wang et al., 2026a; Chen et al., 2026). These works primarily construct or infer executable world representations, whereas world editing focuses on intervening on an already instantiated executable world. Our formulation further organizes such interventions by their depth, from localized property changes to modifications that couple multiple dynamics and systems.

Agentic Game Development. Game generation has long been studied as a content-creation problem, ranging from procedural level design (Sarkar & Cooper, 2020; Migdał et al., 2021) to more recent generative model approaches that synthesize games or game assets (Sudhakaran et al., 2023; Wang et al., 2023b; Hu et al., 2024; Li et al., 2025; Sani et al., 2026). In parallel, coding agents have progressed from isolated programming problems to repository-level software engineering, including bug fixing, feature implementation, testing, and large-scale codebase navigation (Jimenez et al., 2024; Yang et al., 2024a;b). Related benchmarks further evaluate agents that interact with software tools and graphical environments (Xie et al., 2024; Huang et al., 2024; Yuan et al., 2026). More recently, several works have applied agentic coding to game development, using code and engine APIs to construct playable games (Chen et al., 2025; Jiang et al., 2026; Zhang et al., 2026; Luo et al., 2026; Chi et al., 2026; Yin et al., 2026). These efforts primarily focus on constructing games or completing game-development tasks. Closest to our setting, StarCharM (Zand Miralvand et al., 2025) studies generative AI for modding a commercial game (Stardew Valley), but as an HCI case study of a single content type without executable evaluation. In contrast, world editing treats modification of an existing executable world as the target capability. Success requires realizing the requested intervention while maintaining behavioral and visual consistency.

7 Conclusion

We introduced **world editing** as the capability to deliberately intervene on an existing executable world while preserving what should remain unchanged. We further proposed **intervention depth** as an axis describing how strongly a requested edit couples world entities, dynamics, and systems. Using executable games as a testbed, we find that current frontier coding agents exhibit substantial world-editing capability, but reliability varies strongly with intervention depth. This trend is not reducible to the number of evaluation criteria alone: deeper interventions remain less reliable within comparable criterion-count ranges. At the same time, most unsuccessful edits produce executable worlds yet fail during behavioral validation, indicating that the main bottleneck lies in realizing the requested world semantics rather than achieving basic build correctness. Visual integration remains a separate weakness, suggesting that functional and perceptual world editing are distinct dimensions of capability. Although IGMBench quantitatively evaluates Minecraft and Terraria, the underlying formulation is broader than game modding. Executable games provide a concrete setting in which interventions can be applied and verified, but the same notion of world editing extends to other executable worlds whose states, dynamics, and interacting systems can be modified. We hope IGMWorld and IGMBench are a step toward treating world editing as a first-class capability alongside world generation and interaction: not only acting within or generating worlds, but deliberately transforming existing ones while maintaining their coherence.

AI USE STATEMENT

In this work, we used generative AI tools to help refine hypotheses, provide feedback on research methodology and experiments, and implement methods. We have not used generative AI tools to generate synthetic datasets, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, or assist with translations. Other required disclosure categories were not applicable to this work. Additionally, we used generative AI tools for language editing and organization of the manuscript. We have reviewed all AI-assisted work. AI-assisted code and analyses were manually inspected and validated through the experimental and evaluation procedures described in this paper, and AI-assisted manuscript content was reviewed and edited by the authors. We take responsibility for the final content of this work, including text, claims, code, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The source code of IGMWorld, the task specifications of IGMBench, and the accompanying evaluation scripts are original works of the authors. All benchmark tasks were authored from scratch. We did not copy, adapt, or redistribute files from existing publicly released mods, except for the official or community modding APIs and toolchains required to build and load mods. Minecraft and Terraria remain commercial works of their respective rights holders. Our release does not include game binaries, proprietary assets, or other copyrighted game files. Users must obtain the games and compatible toolchains independently and are responsible for complying with the applicable licenses and terms of use. The tasks in IGMBench are designed to evaluate benign world-editing capabilities and do not intentionally request harmful or deceptive modifications.

REPRODUCIBILITY STATEMENT

We discuss the implementation details in Appendix E. Code, task specifications, and evaluation scripts will be released publicly under the MIT License upon acceptance. The release will include the pinned environment specifications used in our experiments: Minecraft 1.21.1 with Fabric and Java 21, and the corresponding pinned tModLoader release with its bundled .NET runtime. Each task will provide the initial repository, the instruction view exposed to the agent, and the verification view used by the evaluator. Worlds used for runtime checks are generated with fixed seeds. A third party will be able to reproduce the evaluation pipeline by independently obtaining the required commercial games, building the provided sandbox images, running an agent on the instruction view, and applying the same build, load, state, behavioral, and visual checks. We report model identifiers, providers, access dates, harness configurations, timeout ($t = 3600$ seconds), and other experimental settings in the Appendix. Unless otherwise specified, inference parameters follow provider-recommended defaults at the time of access. We will not redistribute Minecraft, Terraria, or other proprietary game files. Reproducing the experiments therefore requires a lawfully obtained copy of each game and the corresponding Fabric or tModLoader toolchain. Results for proprietary models may vary if providers update the served model endpoints after the reported access dates. Open-weight model results can be reproduced using the released model identifiers and the same harness configurations.

REFERENCES

0x0funky. Agent Sprite Forge. https://github.com/0x0funky/agent-sprite-forge, 2026. GitHub repository, accessed 2026-09-25.

Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in atari, 2024. URL https://arxiv.org/abs/2405.12399.

Anthropic. Claude opus 4.8, 2026. URL https://www.anthropic.com/news/claude-opus-4-8. Accessed: 2026-08-30.

Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Bech-

tle, Feryal Behbahani, Stephanie Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. Genie: Generative interactive environments, 2024. URL https://arxiv.org/abs/2402.15391.

Chenjie Cao, Xuhui Zuo, Zhenwei Wang, Yisu Zhang, Junta Wu, Zhenyang Liu, Yuning Gong, Yang Liu, Bo Yuan, Chao Zhang, Coopers Li, Dongyuan Guo, Fan Yang, Haiyu Zhang, Hang Cao, Jianchen Zhu, Jiaxin Lin, Jie Xiao, Jihong Zhang, Junlin Yu, Lei Wang, Lifu Wang, Lilin Wang, Linus, Minghui Chen, Peng He, Penghao Zhao, Qi Chen, Rui Chen, Rui Shao, Sicong Liu, Wangchen Qin, Xiaochuan Niu, Xiang Yuan, Yi Sun, Yifei Tang, Yifu Sun, Yihang Lian, Yonghao Tan, Yuhong Liu, Yuyang Yin, Zhiyuan Min, Tengfei Wang, and Chunchao Guo. Hy-world 2.0: A multi-modal world model for reconstructing, generating, and simulating 3d worlds, 2026. URL https://arxiv.org/abs/2604.14268.

Dake Chen, Haoyang Zhang, Hanbin Wang, Yunhao Huo, Yuzhao Li, and Junjie Wang. Gamegpt: Multi-agent collaborative framework for game development, 2025. URL https://arxiv.org/abs/2310.08067.

Yiwen Chen, Guosheng Lin, and Chi Zhang. Code world model: Coding agent as world brain, 2026. URL https://arxiv.org/abs/2608.25927.

Xiangchen Cheng, Yunwei Jiang, Jianwen Sun, Zizhen Li, Chuanhao Li, Xiangcheng Cao, Yihao Liu, Fanrui Zhang, Li Jin, and Kaipeng Zhang. Agenticsts: A bounded-memory testbed for long-horizon llm agents, 2026. URL https://arxiv.org/abs/2607.02255.

Wayne Chi, Yixiong Fang, Arnav Yayavaram, Siddharth Yayavaram, Seth Karten, Qiuhong Anna Wei, Runkun Chen, Alexander Wang, Valerie Chen, Ameet Talwalkar, and Chris Donahue. Gamedevbench: Evaluating agentic capabilities through game development, 2026. URL https://arxiv.org/abs/2602.11103.

Chucklefish. Starbound. Video game [PC], 2016. URL https://playstarbound.com.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu,

Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Fabric Development Team. Fabric: A flexible platform-independent mod loader for minecraft, 2018. URL https://fabricmc.net/. Accessed: 2026-08-09.

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.

Zelin Gao, Qiuyu Wang, Yanhong Zeng, Jiapeng Zhu, Ka Leong Cheng, Yixuan Li, Hanlin Wang, Yinghao Xu, Shuailei Ma, Yihang Chen, Jie Liu, Yansong Cheng, Yao Yao, Jiayi Zhu, Yihao Meng, Kecheng Zheng, Qingyan Bai, Jingye Chen, Zehong Shen, Yue Yu, Xing Zhu, Yujun Shen, and Hao Ouyang. Advancing open-source world models, 2026a. URL https://arxiv.org/abs/2601.20540.

Zelin Gao, Qiuyu Wang, Jiapeng Zhu, Jingye Chen, Zichen Liu, Qingyan Bai, Jiahao Wang, Yufeng Yuan, Hanlin Wang, Yichong Lu, Ka Leong Cheng, Haojie Zhang, Jian Gao, Tianrui Feng, Yuzheng Liu, Yao Yao, Yinghao Xu, Xing Zhu, Yujun Shen, and Hao Ouyang. Infinite worlds with versatile interactions, 2026b. URL https://arxiv.org/abs/2607.07534.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang,

Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Google DeepMind. Gemini 3.5 flash: Model card, 2026a. URL https://deepmind.google/models/model-cards/gemini-3-5-flash/.

Google DeepMind. Nano Banana 2: Combining Pro capabilities with lightning-fast speed. https://blog.google/innovation-and-ai/technology/ai/nano-banana-2/, feb 2026b. Accessed: 2026-09-23.

Junliang Guo, Yang Ye, Tianyu He, Haoyu Wu, Yushu Jiang, Tim Pearce, and Jiang Bian. Mineworld: a real-time and open-source interactive world model on minecraft, 2025. URL https://arxiv.org/abs/2504.08388.

David Ha and Jürgen Schmidhuber. World models. 2018. doi: 10.5281/ZENODO.1207631. URL https://zenodo.org/record/1207631.

Danijar Hafner, Wilson Yan, and Timothy Lillicrap. Training agents inside of scalable world models, 2025. URL https://arxiv.org/abs/2509.24527.

Anthony Hu, Václav Volhejn, Adrien Ramanana Rahary, Chris Mulder, Aditya Makkar, Alyx Liao, Amélie Royer, Manu Orsini, Adam Jelley, Eloi Alonso, Florian Laurent, Fredrik Norén, James Swingos, Jan Hünermann, Kent Rollins, Lucas Hosseini, Matthieu Le Cauchois, Maxim Peter, Pim de Witte, Tim Brown, Vincent Micheli, Moritz Böhle, Gabriel de Marmiesse, Viktoriia Sharmanska, Lucia Specia, Michael Black, and Patrick Pérez. Multiplayer interactive world models with representation autoencoders, 2026. URL https://arxiv.org/abs/2607.05352.

Chengpeng Hu, Yunlong Zhao, and Jialin Liu. Game generation via large language models, 2024. URL https://arxiv.org/abs/2404.08706.

Ian Huang, Guandao Yang, and Leonidas Guibas. Blenderalchemy: Editing 3d graphics with vision-language models, 2024. URL https://arxiv.org/abs/2404.17672.

Yilei Jiang, Jinyuan Hu, Qianyin Xiao, Yaozhi Zheng, Ruize Ma, Kaituo Feng, Jiaming Han, Tianshuo Peng, Kaixuan Fan, Manyuan Zhang, and Xiangyu Yue. Opengame: Open agentic coding for games, 2026. URL https://arxiv.org/abs/2604.18394.

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?, 2024. URL https://arxiv.org/abs/2310.06770.

Klaus Krippendorff. Computing krippendorff's alpha-reliability, 2011. URL https://api.semanticscholar.org/CorpusID:59901023.

Wolfgang Lehrach, Daniel Hennes, Miguel Lazaro-Gredilla, Xinghua Lou, Carter Wendelken, Zun Li, Antoine Dedieu, Jordi Grau-Moya, Marc Lanctot, Atil Iscen, John Schultz, Marcus Chiam, Ian Gemp, Piotr Zielinski, Satinder Singh, and Kevin P. Murphy. Code world models for general game playing, 2025. URL https://arxiv.org/abs/2510.04542.

Ruihuang Li, Caijin Zhou, Shoujian Zheng, Jianxiang Lu, Jiabin Huang, Comi Chen, Junshu Tang, Guangzheng Xu, Jiale Tao, Hongmei Wang, Donghao Li, Wenqing Yu, Senbo Wang, Zhimin Li, Yetshuan Shi, Haoyu Yang, Yukun Wang, Wenxun Dai, Jiaqi Li, Linqing Wang, Qixun Wang, Zhiyong Xu, Yingfang Zhang, Jiangfeng Xiong, Weijie Kong, Chao Zhang, Hongxin Zhang, Qiaoling Zheng, Weiting Guo, Xinchi Deng, Yixuan Li, Renjia Wei, Yulin Jian, Duojun Huang, Xuhua Ren, Junkun Yuan, Zhengguang Zhou, Jiaxiang Cheng, Bing Ma, Shirui Huang, Jiawang Bai, Chao Li, Sihuan Lin, Yifu Sun, Yuan Zhou, Joey Wang, Qin Lin, Tianxiang Zheng, Jingmiao Yu, Jihong Zhang, Caesar Zhong, Di Wang, Yuhong Liu, Linus, Jie Jiang, Longhuang Wu, Shuai Shao, and Qinglin Lu. Hunyuan-game: Industrial-grade intelligent game creation model, 2025. URL https://arxiv.org/abs/2505.14135.

Zhen Li, Zian Meng, Shuwei Shi, Mingliang Zhai, Jiaming Tan, Chuanhao Li, and Kaipeng Zhang. From pixels to states: Rethinking interactive world models as game engines, 2026. URL https://arxiv.org/abs/2607.14076.

Tongxu Luo, Rongsheng Wang, Jiaxi Bi, Chenming Xu, Zhengyang Tang, Jianlong Chen, Juhao Liang, Ke Ji, Shuqi Guo, Yuhao Du, Fan Bu, Wenyu Du, Xiaotong Zhang, Kyle Li, Shaobo Wang, Linfeng Zhang, Yuxuan Liu, Xin Lai, Chenxin Li, Yiduo Guo, Zhexin Zhang, Xinyuan Wang, Tianyi Bai, Ziniu Li, and Benyou Wang. Gamecraft-bench: Can agents build playable games end-to-end in a real game engine?, 2026. URL https://arxiv.org/abs/2606.17861.

Xiaofeng Mao, Zhen Li, Chuanhao Li, Xiaojie Xu, Kaining Ying, Tong He, Jiangmiao Pang, Yu Qiao, and Kaipeng Zhang. Yume-1.5: A text-controlled interactive world generation model, 2025a. URL https://arxiv.org/abs/2512.22096.

Xiaofeng Mao, Shaoheng Lin, Zhen Li, Chuanhao Li, Wenshuo Peng, Tong He, Jiangmiao Pang, Mingmin Chi, Yu Qiao, and Kaipeng Zhang. Yume: An interactive world generation model, 2025b. URL https://arxiv.org/abs/2507.17744.

Diego Marti Monso, Francesco Sacco, and Edward Hu. How to train a frontier-level world model, jul 2026. URL https://next-state.github.io/open-dreamer/.

Piotr Migdał, Bartłomiej Olechno, and Błażej Podgórski. Level generation and style enhancement – deep learning for game development overview, 2021. URL https://arxiv.org/abs/2107.07397.

Mojang. Minecraft. Video game, 2009. Java edition released in 2009; official full release in 2011.

Nous Research. Hermes agent: The agent that grows with you. https://github.com/nousresearch/hermes-agent, 2026. GitHub repository, accessed 2026-09-25.

OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition. OpenAI Blog, July 2026a. URL https://openai.com/index/gpt-5-6/. Accessed: 2026-08-10.

OpenAI. Introducing ChatGPT Images 2.0. https://openai.com/index/introducing-chatgpt-images-2-0/, apr 2026b. Accessed: 2026-09-23.

Pocketpair. Palworld. Video game, 2024. URL https://www.palworldgame.com/.

PrismarineJS. Mineflayer: Create minecraft bots with a javascript api. https://github.com/PrismarineJS/mineflayer, 2011. GitHub repository, accessed 2026-09-25.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

Re-Logic. Terraria. Video game, 2011. URL https://www.terraria.org.

Samin Mahdizadeh Sani, Max Ku, Nima Jamali, Matina Mahdizadeh Sani, Paria Khoshtab, Wei-Chieh Sun, Parnian Fazel, Zhi Rui Tam, Thomas Chong, Edisy Kin Wai Chan, Donald Wai Tong Tsang, Chiao-Wei Hsu, Ting Wai Lam, Ho Yin Sam Ng, Chiafeng Chu, Chak-Wing Mak, Keming Wu, Hiu Tung Wong, Yik Chun Ho, Chi Ruan, Zhuofeng Li, I-Sheng Fang, Shih-Ying Yeh, Ho Kei Cheng, Ping Nie, and Wenhu Chen. Imagenworld: Stress-testing image generation models with explainable human evaluation on open-ended real-world tasks. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=bld9g6jFh9.

Anurag Sarkar and Seth Cooper. Towards game design via creative machine learning (gdcml), 2020. URL https://arxiv.org/abs/2008.13548.

Georgy Savva, Oscar Michel, Daohan Lu, Suppakit Waiwitlikhit, Timothy Meehan, Dhairya Mishra, Srivats Poddar, Jack Lu, and Saining Xie. Solaris: Building a multiplayer video world model in minecraft. *arXiv preprint arXiv:2602.22208*, 2026.

Tyrone Serapio, Arjun Prakash, Haoyang Xu, Kevin Wang, and Amy Greenwald. Distilling game code world model generation into lightweight large language models, 2026. URL https://arxiv.org/abs/2605.24375.

Gowthami Somepalli, Anubhav Gupta, Kamal Gupta, Shramay Palta, Micah Goldblum, Jonas Geiping, Abhinav Shrivastava, and Tom Goldstein. Measuring style similarity in diffusion models, 2024. URL https://arxiv.org/abs/2404.01292.

Shyam Sudhakaran, Miguel González-Duque, Claire Glanois, Matthias Freiberger, Elias Najarro, and Sebastian Risi. Mariogpt: Open-ended text2level generation through large language models, 2023. URL https://arxiv.org/abs/2302.05981.

Hao Tang, Darren Key, and Kevin Ellis. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment, 2024. URL https://arxiv.org/abs/2402.12275.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, M. C., Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y. Charles, H. S. Che, Guanduo Chen, Guangyu Chen, Guanzheng Chen, Huarong Chen, Jia Chen, Jianlong Chen, Jun Chen, Kexin Chen, Peng Chen, Ruijue Chen, Wentao Chen, Xin Chen, Yang Chen, Yanru Chen, Yifei Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Dazhi Cheng, Yean Cheng, Jialei Cui, Jingbing Cui, Anqi Dai, Jiaqi Deng, Hao Ding, Rui Ding, Shaofeng Ding, Mengfan Dong, Mengnan Dong, Yuhao Dong, Yuxin Dong, Angang Du, Chenzhuang Du, Dikang Du, Jusen Du, Yulun Du, Yu Fan, Jing Feng, Qiulin Feng, Yichen Feng, Kelin Fu, Qiang Fu, Fuxuan Gao, Hongcheng Gao, Jingyue Gao, Tong Gao, Weijia Gao, Shangyi Geng, Jie Gong, Linhu Gong, Shengao Gong, Xiaochen Gong, Qizheng Gu, Yicheng Gu, Shuhao Guan, Haiqing Guo, Shiqi Guo, Xiang Guo, Zhengyan Guo, Beixi Hao, Wenxin Hao, Xiaoru Hao, Dailan He, Haotian He, Lehan He, Qi He, Weiran He, Xinran He, Xinyi He, Yibo He, Yunjia He, Chao Hong, Tiange Hong, Hao Hu, Jiaxi Hu, Ruikun Hu, Weiming Hu, Yangyang Hu, Zhenxing Hu, Liang Hua, Jinbin Huang, Ke Huang, Ruiyuan Huang, Siying Huang, Weixiao Huang, Yan Huang, Zhengjie Huang, Zhiqi Huang, Yulong Hui, Chaobo Jia, Yutong Jiang, Zhejun Jiang, Zuoyou Jiang, Wenyi Jin, Xinyi Jin, Yu Jing, Huanjun Kong, Guokun Lai, Aidi Li, Cheng Li, Chengyuan Li, Cong Li, Fang Li, Guanyu Li, Haoyang Li, Jia Li, Junxiong Li, Lei Li, Letian Li, Lincan Li, Weihong Li, Wentao Li, Xintong Li, Yang Li, Yishen Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zhaoxi Li, Zheming Li, Zhengxiao Li, Zhiyuan Li, Jiawei Lin, Xiaohan Lin, Yibo Lin, Zichao Lin, Ziyan Lin, Bill Liu, Boxiao Liu, Chuan Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yipeng Liu, Zhengying Liu, Zhiheng Liu, Enzhe Lu, Haoyu Lu, Linqiang Lu, Tingzhan Lu, Zhiyuan Lu, Aotian Luo, G. Luo, Junyu Luo, Yifan Luo, B. Lyu, Wenzhou Lyu, Shaoguang Mao, Yuan Mei, Xin Men, Minqing Ni, Yixuan Niu, Siyuan Pan, Shujun Peng, Zhangyang Qi, Ruoyu Qin, ZeChao Qin, Zeyu Qin, Haiquan Qiu, Jianxin Qiu, Jiezhong Qiu, Bowen Qu, Yuhao Qu, Zeyu Shang, Youbo Shao, Han Shen, Jincheng Shi, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Wingchun Siu, Pengwei Song, Xiaoxi Song, Jianlin Su, Yunfeng Su, Zhaochen Su, Lin Sui, Jingsong Sun, Junyao Sun, Shaoning Sun, Shuzhe Sun, Tongyu Sun, Yujun Sun, Yunpeng Tai, Chuning Tang, Heyi Tang, Sirui Tang, Zecheng Tang, Chaoran Tian, Rongpeng Tian, Yu Tian, Wei Tu, Chensi Wang, Chuang Wang, Chunjie Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Hao Wang, Hao Wang, Huaqing Wang, Hui Wang, Jiayi Wang, Jinglong Wang, Jinhong Wang, Jiuzheng Wang, Linian Wang, Shaobo Wang, Shenzhi Wang, Shuyi Wang, Si Wang, Siyuan Wang, Tianfu Wang, Wenjue Wang, Xingran Wang, Xinmei Wang, Xinyuan Wang, Xusheng Wang, Yalin Wang, Yangkun Wang, Yao Wang, Yaoyu Wang, Yejie Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhenhao Wang, Zhongsheng Wang, Zifan Wang, Chu Wei, Ming Wei, Shouxin Wei, Zichen Wen, Fan Wu, Haoning Wu, Rucong Wu, Wenhao Wu, Xiaoxue Wu, Yingcong Wu, Yongqi Wu, Yuxin Wu, Zijian Wu, Xinglang Xian, Chenxuan Xiang, Yuye Xiang, Bocheng Xiao, Chenjun Xiao, Xin Xiao, Jin Xie, Xiaotong Xie, Yifeng Xie, Zhe Xie, Bowei Xing, Yiming Xiong, Baosheng Xu, Boyu Xu, Jiale Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Qingtao Xu, Shuyao Xu, Suting Xu, Tiantian Xu, Tianxiang Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ye Xu, Yueni Xu, Ziyao Xu, Haonan Xue, Junjie Yan, Yaoyao Yan, Fan Yang, Guangyao Yang, Hao Yang, Junwei Yang, Ruoyu Yang, Wenjie Yang, Xiaofei Yang, Xinyu Yang, Yi Yang, Yiling Yang, Ying Yang, Yuchen Yang, Zhen Yang, Zhilin Yang, Zian Yang, Zuhao Yang, Haotian Yao, Dan Ye, Haoran Ye, Wenjie Ye, Zhanbo Ye, Bohong Yin, Haoxiang Yin, Xietong Yin,

Chengzhen Yu, Haozhen Yu, Longhui Yu, Shengnan Yu, Shuying Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Tongtian Yue, Wei Yue, Yang Yue, Dunyuan Zha, Haobing Zhan, B. H. Zhang, Dehao Zhang, Fei Zhang, Hao Zhang, Haoyuan Zhang, Huanyu Zhang, Jiapei Zhang, Jiaxuan Zhang, Jin Zhang, Kaiyi Zhang, Miaozhen Zhang, Puqi Zhang, Qinglei Zhang, Rong Zhang, Rui Zhang, Shaoshuai Zhang, Shiyi Zhang, Xiaobin Zhang, Xiaoyun Zhang, Y. Zhang, Yangkun Zhang, Ye Zhang, Yichi Zhang, Yikun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Zijing Zhang, Bin Zhao, Chenguang Zhao, Feifan Zhao, Jinglun Zhao, Jinxiang Zhao, Shuai Zhao, Wenshuo Zhao, Xiangyu Zhao, Xuanle Zhao, Yikai Zhao, Zijia Zhao, Haozhi Zheng, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Haofeng Zhong, Lei Zhong, Longguang Zhong, M. Zhou, Qiankang Zhou, Runjie Zhou, Ruozhang Zhou, Xinyu Zhou, Yiqiao Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yangjunfeng Zhu, Yuxuan Zhu, Zhen Zhu, Chen Zhuang, Weiyu Zhuang, and Xinxing Zu. Kimi k3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

Team PEAK. Peak. Video Game, June 2025. URL https://store.steampowered.com/app/3527290/PEAK/. Windows via Steam.

tModLoader Team. tmodloader: A mod to make and play terraria. https://github.com/tModLoader/tModLoader, 2020. Open-source modification tool for Terraria.

Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion models are real-time game engines, 2025. URL https://arxiv.org/abs/2408.14837.

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, 2023a. URL https://arxiv.org/abs/2305.16291.

Hanyang Wang, Yimo Cai, Weiliang Chen, Jiawei Chi, Haowen Sun, Qiyu Dai, Yi-Hsin Hung, Xingzhuo Guo, Jinshan Ren, Runmao Yao, Ziwei Liu, Mingsheng Long, Yueqi Duan, Jun Gao, Jiangran Lyu, Fangfu Liu, and Jialong Wu. Code as worlds: Agentic discovery of executable world representations for physical reasoning, 2026a. URL https://arxiv.org/abs/2608.27549.

Ruoyao Wang, Graham Todd, Eric Yuan, Ziang Xiao, Marc-Alexandre Côté, and Peter Jansen. Bytesized32: A corpus and challenge task for generating task-specific world models expressed as text games, 2023b. URL https://arxiv.org/abs/2305.14879.

Sheng-Yu Wang, Yotam Nitzan, Aaron Hertzmann, Jun-Yan Zhu, Eli Shechtman, Alexei A. Efros, and Richard Zhang. The many senses of visual similarity: A text-prompted image perceptual metric. *arXiv preprint arXiv:2607.18237*, 2026b.

Zhenwei Wang, Yuhao Liu, Junta Wu, Zixiao Gu, Haoyuan Wang, Xuhui Zuo, Tianyu Huang, Wenhuan Li, Sheng Zhang, Yihang Lian, Yulin Tsai, Lifu Wang, Sicong Liu, Puhua Jiang, Xianghui Yang, Dongyuan Guo, Yixuan Tang, Xinyue Mao, Jiaao Yu, Junlin Yu, Jihong Zhang, Meng Chen, Liang Dong, Yiwen Jia, Chao Zhang, Yonghao Tan, Hao Zhang, Zheng Ye, Peng He, Runzhou Wu, Minghui Chen, Zhan Li, Wangchen Qin, Lei Wang, Yifu Sun, Lin Niu, Xiang Yuan, Xiaofeng Yang, Yingping He, Jie Xiao, Yangyu Tao, Jianchen Zhu, Jinbao Xue, Kai Liu, Chongqing Zhao, Xinming Wu, Tian Liu, Peng Chen, Di Wang, Yuhong Liu, Linus, Jie Jiang, Tengfei Wang, and Chunchao Guo. Hunyuanworld 1.0: Generating immersive, explorable, and interactive 3d worlds from words or pixels, 2025. URL https://arxiv.org/abs/2507.21809.

Zile Wang, Zexiang Liu, Jiaxing Li, Kaichen Huang, Baixin Xu, Fei Kang, Mengyin An, Peiyu Wang, Biao Jiang, Yichen Wei, et al. Matrix-game 3.0: Real-time and streaming interactive world model with long-horizon memory. *arXiv preprint arXiv:2604.08995*, 2026c.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. URL https://arxiv.org/abs/2404.07972.

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, 2024a. URL https://arxiv.org/abs/2405.15793.

John Yang, Carlos E. Jimenez, Alex L. Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R. Narasimhan, Diyi Yang, Sida I. Wang, and Ofir Press. Swe-bench multimodal: Do ai systems generalize to visual software domains?, 2024b. URL https://arxiv.org/abs/2410.03859.

Lei Yin, Wentao Cheng, Zhida Qin, Tianyu Huang, Yidong Li, and Gangyi Ding. Autoue: Automated generation of 3d games in unreal engine via multi-agent systems, 2026. URL https://arxiv.org/abs/2603.07106.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, Dunjie Lu, Haikong Lu, Qi Zhen, Xinyuan Wang, Jiaqi Deng, Yuhao Yang, Cheng Chen, Boyuan Zheng, Alex Su, Xiao Yu, Hao Zou, Saaket Agashe, Xing Han Lu, Manpreet Kaur, Zhengyang Qi, Vincent Sunn Chen, Frederic Sala, Dayiheng Liu, Junyang Lin, Zhou Yu, Yu Su, Siva Reddy, Xin Eric Wang, Peng Qi, Tianbao Xie, and Tao Yu. Osworld 2.0: Benchmarking computer use agents on long-horizon real-world tasks, 2026. URL https://arxiv.org/abs/2606.29537.

Hamid Zand Miralvand, Mohammad Ronagh Nikghalb, Mohammad Darandeh, Abidullah Khan, Ian Arawjo, and Jinghui Cheng. Democratizing game modding with genai: A case study of starcharm, a stardew valley character maker. *Proceedings of the ACM on Human-Computer Interaction*, 9 (6):475–509, October 2025. ISSN 2573-0142. doi: 10.1145/3748612. URL http://dx.doi.org/10.1145/3748612.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018. URL https://arxiv.org/abs/1801.03924.

Wenyu Zhang, Guoliang You, Tianlun, Haotian Zhao, Tianshu Zhu, Haoran Wang, Xiaoxuan Tang, Mingyang Dai, Jingnan Gu, Daxiang Dong, and Jianmin Wu. Webgamebench: Requirement-to-application evaluation for coding agents via browser-native games, 2026. URL https://arxiv.org/abs/2605.17637.

Pengfei Zhou, Hexin Wang, Zhengfeiyang Zhang, Yixing Ma, Zhenglin Wan, Kaipeng Zhang, Wangbo Zhao, and Yang You. Agentic game development as a verifiable trajectory data engine for scaling world models, 2026. URL https://arxiv.org/abs/2608.25518.

LIMITATIONS

Verification of open-ended world modifications. Even with task-specific state and behavioral criteria, no finite evaluator can exhaustively verify an arbitrary world modification. Some unintended side effects may remain outside the tested scenarios, particularly for deeper interventions involving multiple interacting systems. Broader verification would require stronger regression testing and more extensive runtime coverage, including long-horizon or multiplayer scenarios for interaction-dependent failures.

Generalization across executable worlds. We instantiate IGMBench in Minecraft and Terraria using Fabric and tModLoader. Although these ecosystems are mature and support a broad range of interventions, other games expose different APIs, asset pipelines, runtime interfaces, and modding constraints. Figure 5 illustrates that the same intervention taxonomy can be applied to additional games, including PEAK, Palworld, and Starbound, but these examples are not part of IGMBench and are not included in the quantitative evaluation. Quantitative evaluation across a broader range of executable worlds remains future work, as each environment requires ecosystem-specific execution and validation infrastructure.

Beyond state, behavior, and visual consistency. Our evaluation primarily targets executable world state, behavior, and visual assets. Other modalities that contribute to a complete game experience, such as audio, animation, narrative, are not explicitly evaluated. Extending the benchmark to these dimensions is non-trivial: tasks involving such modalities are less naturally captured by deterministic state checks, and reliable automatic evaluation would require additional modality-specific infrastructure and validation protocols.

Single-trial agent evaluation. Each model–task pair is evaluated once due to the cost of native game execution and multimodal validation. Consequently, our results characterize observed configuration-level performance rather than estimating per-task success probabilities across repeated stochastic runs. Repeated evaluation would provide stronger estimates of variance and reliability, but would substantially increase experimental cost.

A APPENDIX

A.1 MORE REPRESENTATIVE EXAMPLES

This section provides additional qualitative examples of failed world edits, together with visual comparisons of agent-generated assets. Figure 6 shows representative failure cases across intervention levels. Figures 7 and 8 compare image-generated and programmatically drawn assets against native game textures.

B DETAILED RESULTS

We provide detailed functional, visual, regression, and efficiency results in Tables 5, 6, 7, and 8, respectively.

C FAILURE STUDY AND AGENT BEHAVIORS

Failure stages and domains. We further analyze unsuccessful and invalid runs to characterize where world-editing attempts break down. Most failed submissions progress past compilation and loading, with failures instead appearing during behavioral validation (Table 9). Across the audited runs, interaction/progression and mechanic semantics are the most common observed failure domains (Figure 9; Table 10). Figure 9 further decomposes these broad domains into more specific failure types, showing that unsuccessful world edits span both high-level behavioral errors and lower-level integration failures. Because many cases remain unresolved with respect to root cause, these categories describe where failures manifest rather than necessarily identifying the underlying agent-side defect. Where the cause can be established confidently, recurring errors are often integration-related, including outdated resource conventions, registration-lifecycle mistakes, missing assets, compilation failures, and incomplete delivery.

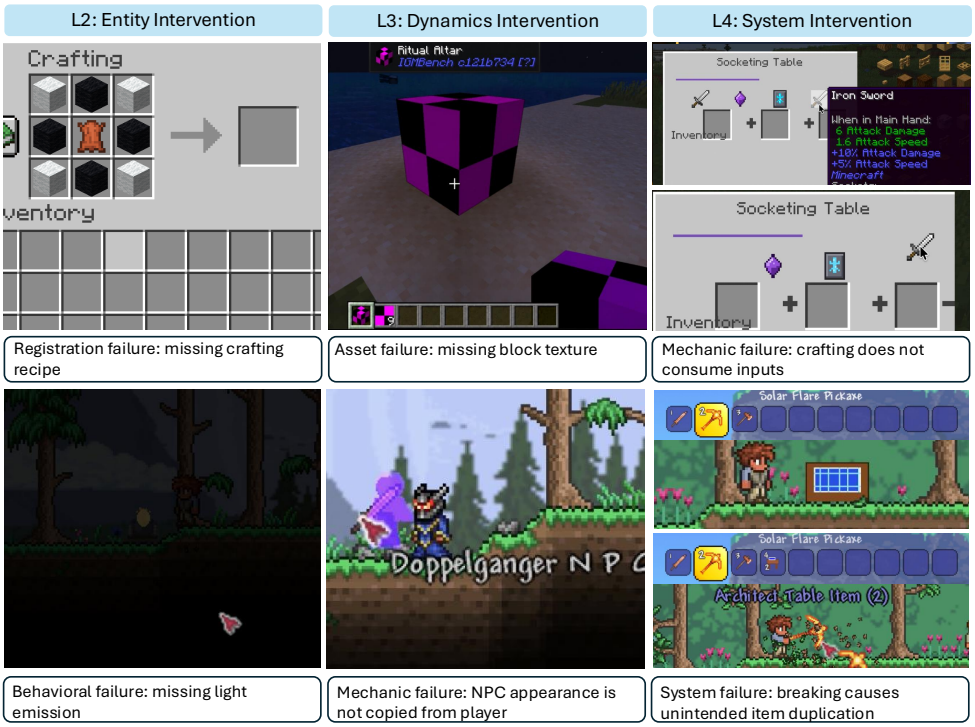


Figure 6: Representative failure cases across intervention levels in Minecraft (top) and Terraria (bottom). Columns show entity (L2), dynamics (L3), and system (L4) interventions. Failures include missing registration or assets, incorrect mechanic execution, and unintended system-level behavior.



Figure 7: Minecraft item textures grouped by generation method: **left**, image-generated; **middle**, native game assets; **right**, programmatically drawn. Image-generated assets tend toward smoother shading and higher contrast, while programmatic assets are generally flatter and more geometric than native Minecraft textures. Each panel shows 36 sampled item textures.

Ecosystem-specific integration errors. Several recurring failures arise from incorrect assumptions about the target game or modding ecosystem rather than from misunderstanding the requested world edit. For example, multiple agents use outdated resource conventions that produce otherwise plausible implementations but prevent the intended content from being registered at runtime. Similar failures arise from incorrect initialization order, loader/API assumptions, or resource layout. These cases distinguish world-edit reasoning from ecosystem integration: an agent may identify the correct entity, mechanic, or system-level change but still fail to realize it in the running game because its implementation does not conform to the target ecosystem.

Development behavior and debugging. Trajectory analysis further shows that unsuccessful runs do not necessarily correspond to less agent effort. For several strong configurations, failing runs contain



Figure 8: Terraria item textures grouped by generation method: **left**, image-generated; **middle**, native game assets; **right**, programmatically drawn. Image-generated assets more closely reproduce Terraria's dense and outlined visual style, while programmatic assets remain visibly simpler and more geometric. Each panel shows sampled item textures from the same size range.

Table 5: Performance on IGMBench across games and world-intervention levels. Build Success Rate (BSR) measures the percentage of submissions that build and load successfully. Criterion Pass Rate (CPR) measures the percentage of individual state and behavioral criteria satisfied, with criteria associated with non-building or non-loading submissions counted as failed. World-Editing Success Rate (WSR) measures the percentage of tasks that build and load successfully and satisfy all associated state and behavioral criteria. WSR columns are shaded, and the highest reported value in each column is shown in bold. All values are percentages.

(a) Terraria

	L1			L2			L3			L4			Overall		
Model	BSR	CPR	WSR	BSR	CPR	WSR	BSR	CPR	WSR	BSR	CPR	WSR	BSR	CPR	WSR
Open-weight															
DeepSeek-V4-Pro	**100**	94.9	76.9	**100**	88.4	42.9	61.5	33.0	23.1	69.2	66.1	23.1	83.0	68.9	41.5
GLM-5.3	**100**	**100**	**100**	92.9	85.0	64.3	84.6	13.5	23.1	46.2	12.0	0.0	81.1	53.0	47.2
Kimi-K3	**100**	99.4	92.3	85.7	71.5	64.3	84.6	41.8	61.5	84.6	69.3	53.8	88.7	67.8	67.9
Proprietary															
Gemini 3.5 Flash	92.3	86.1	76.9	**100**	90.5	71.4	**100**	88.8	46.2	**100**	**98.9**	84.6	**98.1**	90.6	69.8
Claude Opus 4.8	**100**	**100**	**100**	92.9	93.0	78.6	**100**	**99.4**	**84.6**	**100**	96.2	69.2	**98.1**	**97.0**	**83.0**
GPT-5.6 Luna	**100**	97.2	92.3	92.9	92.2	64.3	92.3	90.1	69.2	**100**	97.9	**92.3**	96.2	93.7	79.2
GPT-5.6 Sol	**100**	**100**	**100**	**100**	**97.0**	**85.7**	**100**	97.0	76.9	92.3	90.0	61.5	**98.1**	96.3	81.1

(b) Minecraft

	L1			L2			L3			L4			Overall		
Model	BSR	CPR	WSR	BSR	CPR	WSR	BSR	CPR	WSR	BSR	CPR	WSR	BSR	CPR	WSR
Open-weight															
DeepSeek-V4-Pro	**100**	85.6	71.4	93.3	63.8	6.7	78.6	65.2	14.3	85.7	34.9	0.0	89.5	56.0	22.8
GLM-5.3	**100**	92.1	85.7	93.3	77.6	60.0	85.7	71.4	28.6	92.9	56.8	21.4	93.0	69.4	49.1
Kimi-K3	**100**	96.4	85.7	**100**	**93.8**	66.7	**100**	83.7	42.9	**100**	75.9	14.3	**100**	84.1	52.6
Proprietary															
Gemini 3.5 Flash	**100**	91.4	78.6	93.3	77.4	40.0	85.7	77.1	28.6	85.7	49.5	35.7	91.2	68.5	45.6
Claude Opus 4.8	**100**	98.6	85.7	**100**	92.4	66.7	92.9	89.2	50.0	**100**	84.9	**42.9**	98.2	89.2	61.4
GPT-5.6 Luna	**100**	94.2	**92.9**	**100**	85.7	53.3	**100**	90.7	21.4	85.7	68.2	35.7	96.5	81.8	50.9
GPT-5.6 Sol	**100**	**99.3**	**92.9**	**100**	86.7	**86.7**	**100**	**99.2**	**85.7**	**100**	**88.1**	35.7	**100**	**92.8**	**75.4**

more tool use, probing, and log inspection than successful runs, suggesting that difficult tasks often trigger longer debugging and repair loops rather than immediate breakdowns. Other configurations exhibit the opposite pattern, with failed runs showing less rebuilding and source inspection. Across observable trajectories more generally, agents also adopt different information-gathering strategies (Table 11). GPT-5.6 Sol, GPT-5.6 Luna, and Claude Opus 4.8 predominantly inspect local game sources, data files, and runtime logs, whereas Gemini 3.5 Flash relies much more heavily on online search. Rebuilding after an initial implementation is common across all four observable configurations, indicating that world editing typically follows an iterative edit–build–inspect–repair workflow rather than a single-shot generation process.

Table 6: Detailed visual-quality results by game. N counts registered texture assets. *Custom* is the fraction using custom artwork rather than a vanilla texture or missing file; non-custom assets fail all visual checks. $S - \tau_c$ denotes the margin over the category-specific TPIPS threshold, with positive values indicating a pass; *Both Pass* requires both style and semantic checks to pass.

Model / Source	N	**Custom** ↑	**Style** $S{-}\tau$ ↑	$\text{Pass}_{\text{style}}$ ↑	**Sem.** $S{-}\tau$ ↑	Pass_{sem} ↑	$\text{Pass}_{\text{both}}$ ↑
Minecraft (item, block, entity)							
Vanilla (LOO)	1989	–	+0.041	0.85	+0.062	0.85	0.80
DeepSeek-V4-Pro	142	1.00	-0.026	0.17	-0.029	0.23	0.15
GLM-5.3	226	0.94	+0.001	0.49	+0.003	0.53	0.40
Kimi-K3	205	0.94	-0.005	0.52	-0.000	0.54	0.47
Gemini 3.5 Flash	110	0.95	-0.009	0.32	-0.011	0.36	0.22
Claude Opus 4.8	127	0.96	-0.010	0.34	-0.011	0.42	0.23
GPT-5.6 Luna	158	0.97	-0.027	0.16	-0.030	0.20	0.14
GPT-5.6 Sol	155	0.97	-0.016	0.23	-0.018	0.24	0.19
Terraria (items and projectiles as sprites; tiles and NPCs as frames from vanilla sheets)							
Vanilla (LOO)	15456	–	+0.026	0.85	+0.041	0.85	0.78
DeepSeek-V4-Pro	113	0.90	-0.031	0.16	-0.030	0.23	0.11
GLM-5.3	73	0.74	-0.012	0.32	-0.003	0.37	0.29
Kimi-K3	109	0.92	-0.015	0.29	-0.011	0.35	0.24
Gemini 3.5 Flash	104	1.00	-0.013	0.32	-0.011	0.32	0.22
Claude Opus 4.8	110	0.94	-0.011	0.35	-0.008	0.34	0.25
GPT-5.6 Luna	110	0.85	-0.019	0.18	-0.004	0.36	0.12
GPT-5.6 Sol	99	0.96	-0.021	0.22	-0.014	0.37	0.18

Table 7: Regression preservation on IGMBench. A regression check verifies that a property not targeted by the requested edit remains unchanged, using a paired comparison between the modded world and a mod-free world under the same environment. IGMBench contains 225 regression checks across 30 tasks (14 Minecraft and 16 Terraria), primarily at L1. We report pass rates over checks reached by each configuration; runs that fail to build or load do not execute regression checks.

Model	**Regression pass rate**
Open-weight	
DeepSeek-V4-Pro	99.1%
GLM-5.3	96.9%
Kimi-K3	99.6%
Proprietary	
Gemini 3.5 Flash	98.2%
Claude Opus 4.8	98.7%
GPT-5.6 Luna	97.3%
GPT-5.6 Sol	**100.0%**

Table 8: Cost efficiency of evaluated agents across games and intervention levels. We report the median processed input tokens and agent runtime, where M denotes millions of tokens and m denotes minutes. The lowest value in each column is bolded.

	Minecraft								Terraria							
	L1		L2		L3		L4		L1		L2		L3		L4	
Model	Token	Time	Token	Time	Token	Time	Token	Time	Token	Time	Token	Time	Token	Time	Token	Time
Open-weight																
DeepSeek-V4-Pro	1.3M	8m	**2.5M**	14m	3.8M	19m	4.4M	19m	0.4M	2m	1.8M	10m	2.7M	13m	5.4M	20m
GLM-5.3	2.0M	11m	12.7M	27m	11.3M	27m	12.9M	31m	4.3M	15m	10.9M	25m	11.3M	27m	10.6M	27m
Kimi-K3	1.1M	14m	4.8M	44m	6.6M	47m	6.8M	38m	2.3M	18m	4.5M	25m	9.5M	32m	9.6M	35m
Proprietary																
Gemini 3.5 Flash	1.7M	16m	2.8M	26m	2.8M	30m	**2.4M**	**13m**	0.7M	4m	1.8M	18m	**1.9M**	14m	5.3M	**13m**
Claude Opus 4.8	**0.4M**	**3m**	**2.5M**	11m	**2.7M**	14m	4.3M	18m	**0.1M**	**1m**	**0.9M**	**6m**	**1.9M**	**10m**	4.6M	19m
GPT-5.6 Sol	0.8M	**3m**	5.2M	**10m**	5.1M	**13m**	6.2M	16m	0.2M	2m	2.6M	9m	3.2M	14m	**3.9M**	19m
GPT-5.6 Luna	2.1M	6m	5.7M	13m	8.1M	15m	6.9M	16m	0.5M	3m	3.2M	8m	5.5M	14m	7.1M	19m

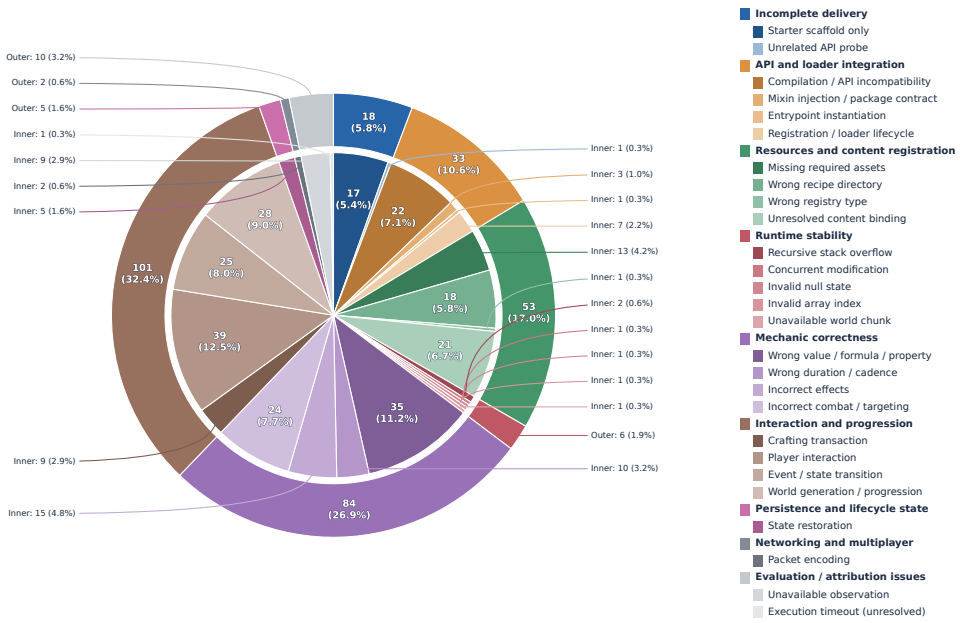


Figure 9: Failure modes in agent-produced world edits. The outer ring shows broad categories and the inner ring shows specific failure types; labels report counts and percentages of categorized failures.

Table 9: Observed failure stage by model across audited unsuccessful runs. Counts indicate whether failures occur during build, load, or behavioral validation.

Model	Build	Load	Behavior
DeepSeek-V4-Pro	2	13	60
GLM-5.3	11	3	44
Kimi-K3	5	1	39
Gemini 3.5 Flash	4	2	42
Claude Opus 4.8	0	2	27
GPT-5.6 Luna	0	4	35
GPT-5.6 Sol	0	1	25

Table 10: Observed failure domains across audited unsuccessful runs. These categories describe where failures are observed and do not always identify the underlying root cause.

Observed domain	Count
Interaction / progression	101
Mechanic semantics	84
Resources / registration	53
API / loader	33
Incomplete delivery	18
Evaluation attribution	18
Runtime crash	6
Persistence	5
Networking	2

Table 11: Agent development behavior across all IGMBench runs. Values are the percentage of runs exhibiting each behavior. Trajectory-level behaviors are reported only for harnesses whose execution traces are observable.

	GPT-5.6 Sol	**GPT-5.6 Luna**	**Gemini 3.5 Flash**	**Claude Opus 4.8**
Read game sources	97%	99%	52%	91%
Read vanilla data	56%	52%	24%	37%
Searched the Web	9%	9%	72%	3%
Used provided skills	100%	96%	93%	58%
Rebuilt after initial build	85%	90%	83%	76%
Ran game server	15%	19%	0%	6%
Ran own probe / test	30%	46%	13%	16%
Read logs	66%	80%	27%	49%
Kept a todo list	25%	65%	0%	0%

D Ablations

D.1 Use of Agent Skills

We provide all evaluated agents with the same set of host-provided, game-specific skills as part of the IGMWorld execution environment. These skills contain generic modding knowledge rather than task-specific solutions or evaluator information. In particular, *IGMWorld-minecraft-modding* summarizes common Fabric registration patterns, client/server-side conventions, and asset and model paths, while *IGMWorld-terraria-modding* provides corresponding tModLoader class and registration patterns. For visual asset construction, agents also have access to *agent-sprite-forge* (0x0funky, 2026), an external pipeline for generating sprite sheets on a keyable background, chroma-keying them locally, extracting frames, and exporting assets with transparency. We also observe that, even without *agent-sprite-forge*, agents are often able to post-process their own generated outputs and produce assets with transparent backgrounds, suggesting that the pipeline provides convenience rather than an essential capability. The skills are made available uniformly, while agents decide autonomously whether and how to use them. This is reflected in the observed trajectories: provided skills are used in 58–100% of runs across the proprietary configurations with observable execution traces (Table 11). We include these skills in the default configuration to provide the same generic ecosystem guidance across runs and reduce incidental friction from modding-specific conventions, while withholding task-specific implementation hints and evaluator information.

In this ablation, the no-skill condition removes both game-specific modding skills and *Agent Sprite Forge*, while leaving all other tools and inference settings unchanged. To measure their effect, we rerun GPT-5.6 Luna without the host-provided skills while keeping the remaining configuration fixed. Table 12 shows that providing skills increases overall WSR from 57.3% to 64.5%, with the largest gain at L4 (33.3% to 63.0%). The effect is not uniform across intervention levels: L1 WSR is unchanged, L2 improves with skills, while L3 is higher without skills. Overall CPR is nearly unchanged, at 88.5% with skills and 89.2% without skills. This suggests that the skills primarily affect the reliability of completing all requirements of a task rather than criterion-level correctness in isolation. The skills do not provide a corresponding improvement in visual quality. As shown in Table 13, the no-skill condition achieves slightly higher style, semantic, and joint visual pass rates in both Minecraft and Terraria. This indicates that the game-modding skills mainly support functional implementation and ecosystem integration, while visual asset quality remains governed by a different set of agent choices and tools. We therefore use the skill-equipped configuration as the default setting in the main experiments and report the no-skill configuration as a controlled ablation.

Table 12: Effect of host-provided skills on GPT-5.6 Luna. We report Criterion Pass Rate (CPR) and World-Editing Success Rate (WSR) across intervention levels, pooled over both Minecraft and Terraria. All values are percentages.

	L1		**L2**		**L3**		**L4**		**Overall**	
Configuration	CPR	WSR	CPR	WSR	CPR	WSR	CPR	WSR	CPR	WSR
Luna (w/ skills)	96.3	92.6	90.2	58.6	90.4	44.4	80.0	63.0	88.5	64.5
Luna (w/o skills)	97.8	92.6	89.8	44.8	93.3	59.3	78.5	33.3	89.2	57.3

Table 13: Effect of host-provided skills on visual asset quality for GPT-5.6 Luna. We report art-style, semantic, and joint visual pass rates for Minecraft and Terraria.

Game	**Configuration**	$\text{Pass}_{\text{style}}$	Pass_{sem}	$\text{Pass}_{\text{both}}$
Minecraft	Luna (w/ skills)	0.16	0.20	0.14
	Luna (w/o skills)	0.22	0.36	0.18
Terraria	Luna (w/ skills)	0.18	0.36	0.12
	Luna (w/o skills)	0.21	0.38	0.19

D.2 Alternative Visual Asset Metrics

CSD as Style Metric. We additionally evaluate CSD (Somepalli et al., 2024) as an alternative representation for measuring visual style consistency. Given a generated asset x and category-matched reference set $\mathcal{I}_c$, we embed each image using the pretrained CSD encoder f_{CSD}, a CLIP (Radford et al., 2021)-based style embedding, and compute the mean cosine similarity to the k nearest references:

$$S_{\mathrm{CSD}}(x; c) = \frac{1}{k} \sum_{r_i \in \mathrm{kNN}(\mathbf{z}_x, \mathcal{I}_c)} \cos(\mathbf{z}_x, \mathbf{z}_i), \tag{3}$$

where $\mathbf{z}_x = f_{\mathrm{CSD}}(x)/\|f_{\mathrm{CSD}}(x)\|_2$ and $\mathbf{z}_i$ is defined analogously.

We calibrate category-specific thresholds using the same leave-one-out protocol as the main TPIPS evaluator. To compare the underlying representations independently of threshold calibration, we additionally measure their threshold-free agreement with human style judgments. Nine raters with prior gaming experience evaluated 40 blind-sampled agent-produced textures and judged whether each asset matched the game's art style. As shown in Table 14, TPIPS conditioned on the factor "art style" achieves an AUC of 0.66 against the majority human judgment, compared with 0.54 for CSD. We therefore use TPIPS for style consistency in the main evaluation pipeline and retain CSD as an alternative-metric comparison. The detail of human visual study is discussed in Appendix E.4.

Table 14: Agreement with majority human judgments of whether agent-produced textures match the game's art style, measured by threshold-free AUC. Nine raters with prior gaming experience evaluated 40 blind-sampled textures; unsure responses were excluded before majority voting.

	CSD	**TPIPS art style**
Majority vote	0.54	**0.66**

E Detail Setup

E.1 Agent and Environment Configuration

Table 15: Agent configurations evaluated on IGMBench. Each configuration specifies a backbone model, coding harness, inference effort, and image-generation backend. All models support approximately 1M-token context windows.

Model	Harness	Effort	Image Tool	Released
Open-weight				
DeepSeek-V4-Pro (DeepSeek-AI et al., 2026)	Hermes Agent	high	Nano Banana	2026-08
GLM-5.3 (GLM-5-Team et al., 2026)	Hermes Agent	high	Nano Banana	2026-08
Kimi-K3 (Team et al., 2026)	Hermes Agent	high	Nano Banana	2026-07
Closed-source				
Gemini 3.5 Flash (Google DeepMind, 2026a)	Gemini-CLI	high	Nano Banana	2026-05
Claude Opus 4.8 (Anthropic, 2026)	Claude Code	xhigh	Nano Banana	2026-05
GPT-5.6 Luna (OpenAI, 2026a)	Codex-CLI	xhigh	GPT-Image	2026-07
GPT-5.6 Sol (OpenAI, 2026a)	Codex-CLI	xhigh	GPT-Image	2026-07

Table 16: The two game docker environments used at IGMWorld.

	Minecraft	Terraria
Game / server	Minecraft 1.21.1	tModLoader v2026.05.3.0
Mod loader	Fabric loader 0.19.3	tModLoader v2026.05.3.0
Build tool	Gradle 8.12	tML-build
Runtime	Java 21	bundled .NET 8.0
Mod API	Fabric API 0.116.14	tModLoader API
Probe channel	RCON+mineflayer	tModLoader ModSystems

E.2 IGMBench Setup

We show the agent prompt and task specification, also the evaluation sample from IGMBench in Figure 11, Figure 12, and Figure 13.

E.3 Evaluating World-Edit Correctness

Each behavior is described by seven fields: actor, action, subject, provenance, context, timing, and outcome. These specify who performs or causes the behavior, what action occurs, what it affects, its required causal history, the surrounding conditions, when it occurs, and its observable result. For example, "a player harvesting a mature crop in Spring has a 20% chance of receiving a Spring token" identifies the player as the actor, harvesting as the action, the mature crop as the subject, and Spring as the context. The outcome is a 20% token drop chance per eligible harvest. Provenance requires evidence that the token came from the harvest rather than being supplied by the test. No additional timing constraint is specified. Each test has five components: fixture, trigger, measurement, acceptance rule, and prerequisite gates. The fixture establishes the actor, subject, and context. The trigger performs the required action. The measurement records the observable result, and the acceptance rule defines what counts as a pass. Prerequisite gates establish the conditions needed for a valid measurement. Timing requirements govern execution and observation, while provenance requirements govern setup and evidence. Compound requirements are split into separate checks for claims such as item registration, crafting, activation, cooldown, and appearance. For behaviors that modify a measurable property of the game world, tests compare that property before and after the required action. Other requirements are tested by inspecting properties directly or examining event traces. The components above define what each check must establish, while their implementation depends on the game's available interfaces. As shown in Table 17, the evaluation remains tractable at benchmark scale, with an overall combined p95 of 508.1 s per task and approximately 6 hours for

a full benchmark run with five parallel workers. We next describe how Minecraft and Terraria each perform test execution and measurement.

Table 17: Evaluation runtime per task, reported as the 95th percentile in seconds. Measurements were collected on a machine with two AMD EPYC 7302 processors (32 cores total) and 503 GiB of RAM, excluding build and orchestration overhead. Using the overall combined p95 of 508.1 s gives a conservative sequential budget of 15.52 hours for all 110 tasks. In practice, evaluating the full benchmark with five parallel workers takes approximately 6 hours.

Game	Phase	L1	L2	L3	L4	All levels
Minecraft	Fresh-world load	46.2	44.4	45.2	44.2	44.8
	Behavior checks	236.1	582.8	421.1	1,738.1	632.9
	Combined	269.6	615.9	463.8	1,770.2	670.2
Terraria	Fresh-world load	35.3	35.1	34.5	35.7	35.2
	Behavior checks	80.9	79.0	68.0	68.0	78.3
	Combined	116.0	110.5	102.5	101.4	112.1
Overall	Fresh-world load	44.4	43.6	43.8	42.9	43.7
	Behavior checks	125.0	165.9	378.4	663.2	475.7
	Combined	164.6	207.8	418.2	698.4	508.1

E.3.1 Minecraft Deterministic Checks Design

Execution and Observation Interface. Minecraft tests use Remote Console (RCON), Mineflayer (PrismarineJS, 2011), and a Fabric client (Fabric Development Team, 2018). RCON sets up fixtures and queries server state. Mineflayer performs ordinary player actions. The Fabric client handles actions that depend on client entrypoints, custom input handling, the candidate mod's client code, or graphical interfaces. Evaluator-owned helpers support test setup and observation.

Runtime Subject and Interface Discovery. Mineflayer discovers registry identifiers through command completion, removes duplicates, and excludes baseline identifiers. It converts identifiers and search terms into tokens, then selects candidates containing all required tokens and no excluded tokens. The Fabric client reads its loaded item registry directly. It matches substrings after lowercasing and removing non-alphanumeric characters, with optional namespace restrictions and filters. Both methods require exactly one match.When command syntax is unspecified, the evaluator reconstructs paths from the command tree and matches them against task-specific regular expressions. Named argument placeholders are filled with test-supplied values and checked against supported argument constraints. Resolution requires exactly one executable invocation, without expanding redirects. Discovery identifies how to invoke an interface; subsequent measurements establish whether it produces the required behavior.

Gameplay Actions and Measurements RCON sets up the actor, subject, environment, and required resources. Mineflayer or the Fabric client then performs the player action. RCON queries or client observations measure changes, final states, or events, which the evaluator checks against the acceptance rule. For requirements relative to vanilla gameplay, the evaluator first measures the same behavior without the candidate mod under matched test conditions.

Timing and Transient Effects. Timing requirements specify when to act and observe. Where supported, the evaluator advances or monitors server ticks to control delays and observation windows. Client-dependent actions also require confirmation that the interaction occurred. Short-lived status effects may expire or lose duration before a query. For supported checks, an evaluator-owned server helper records the maximum remaining duration observed within a monitoring interval. The fixture starts without the target effect to distinguish a new application from an existing effect. Cooldown checks attempt the action before and after the expected cooldown expires and check the outcome. Other timing checks use timed observations or event traces to assess duration and ordering.

Custom Input and GUI Interaction. The Fabric client performs actions involving client-side mod code, custom key bindings, or graphical interfaces. Each test specifies the interaction procedure. For clickable widgets, the evaluator matches active, visible controls by label using a task-specific regular expression, then calls the screen's existing input handling method at the selected widget's center. For custom-drawn interfaces, evaluator inject evaluator-owned hooks into supported rendering methods

to record text, item identities, and coordinates. The evaluator clicks matching text or a nearby item icon. Text-item associations rely on geometric constraints. The evaluator inspects container slots through the current screen renderer's method, distinguishing them from player-inventory slots and reading their contents and insertion rules. To insert an item, it uses slot-click operations to move a test-specified hotbar stack into the first empty container slot that accepts it. To collect output, it transfers a matching stack from a slot that disallows insertion to an empty, test-specified hotbar slot. For key-driven interactions, the evaluator matches task-specific terms against registered key-binding names, labels, and categories. It then generates a press notification for the assigned key, allowing the candidate mod to process the input through its registered handler.

Probabilistic Mechanics. There are four main ways in testing requirements entailing stochastic requirement like "a player harvesting a mature crop in Spring has a 20% chance to receive the Spring token"

1. Monte Carlo-style sampling. The evaluator repeats an eligible event for a task-dependent number of trials and compares outcome counts against predefined acceptance bounds. Qualitative requirements instead use occurrence checks or comparisons.
2. Direct parameter inspection. The evaluator reads registered parameters, such as spawn weights, from runtime server state.
3. Two-point boundary testing. For supported threshold-based decisions, the evaluator forces random inputs immediately below and at the required threshold, then inspects the outcomes.
4. Bounded enumeration. The evaluator executes all supported random-choice paths within its enumeration limits and weights their observed outcomes to calculate an expectation.

E.3.2 Terraria Deterministic Checks Design

Execution and Observation Interface. Terraria tests use an evaluator mod from tModLoader (tModLoader Team, 2020) to setup fixtures, triggers and measurer. The measurements are then emitted to an external verification engine to compare against the acceptance rules.

Runtime Subject Discovery. The evaluator mod reads loaded content registries directly and locates the relevant subjects using a name-matching process similar to Minecraft's.

Probabilistic Mechanics. Supported stochastic checks use fixed-budget repeated trials, following the Monte Carlo-style sampling described for Minecraft. Terraria's crafting checks also examine joint outcomes within each trial. For example, if ingredients A and B each independently have a 50% chance of being saved, the four combinations of saving and consuming them should each occur with probability 25%.

E.4 Evaluating Visual Correctness

Our visual evaluation measures whether agent-produced assets both depict the requested content and remain compatible with the visual distribution of the host game. As described in the main paper, we use category-conditioned TPIPS with separate text factors for semantic identity and art style. For each asset category, the pass threshold is calibrated from leave-one-out scores over original game assets, accounting for differences in visual variability across categories. We use $\alpha = 0.15$ and $k = 5$ throughout the main experiments. Here, we validate these choices through a preliminary human study and sensitivity analysis.

Human calibration of the pass threshold. We conducted a preliminary human perceptual study to guide the choice of α. Figure 10 shows the annotation interface. Nine raters evaluated 40 agent-produced textures without access to the automatic metric scores. For each texture, raters judged whether the asset would visually fit alongside the shown vanilla game art and whether it represented a recognizable and plausible object of the requested type. Table 18 compares the majority human judgments with the automatic style and semantic verdicts across candidate values of α. Because the human labels are skewed toward acceptance, we use balanced accuracy rather than raw agreement. The setting $\alpha = 0.15$ achieves the highest mean balanced accuracy across the two checks. We use this setting throughout the main evaluation. This study is intended as a calibration sanity check

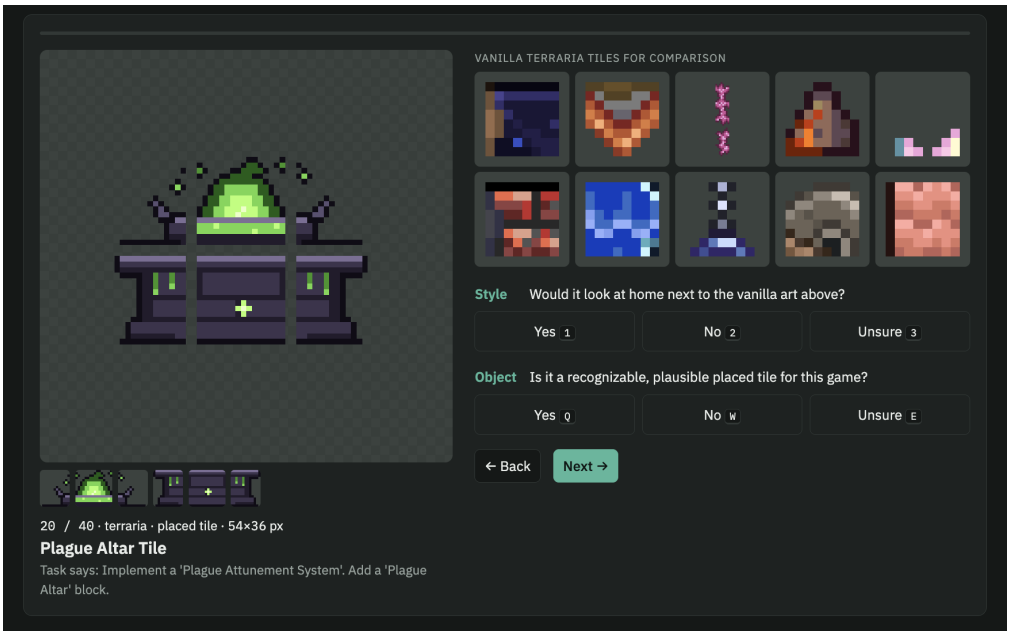


Figure 10: Interface used for the preliminary human perceptual study. Raters judged whether agent-produced assets were stylistically compatible with the shown vanilla references and semantically plausible for the requested object category.

Table 18: Calibration of the pass threshold α using the preliminary human study. Nine raters judged 40 agent-produced textures blind to metric scores. *Acc.* is the fraction of majority-accepted textures passed by the metric, *Rej.* the fraction of majority-rejected textures rejected by the metric, and *Bal.* their mean. *Vanilla* is the fraction of original game assets passing both checks. The shaded row is the setting used in the main experiments.

	Art Style vs. style votes			**Semantic** vs. object votes				
α	Acc.	Rej.	Bal.	Acc.	Rej.	Bal.	**Mean Bal.** ↑	Vanilla
0.01	0.92	0.09	0.51	0.90	0.22	0.56	0.53	0.98
0.02	0.88	0.27	0.58	0.80	0.33	0.57	0.57	0.97
0.05	0.73	0.45	0.59	0.63	0.33	0.48	0.54	0.92
0.10	0.58	0.64	0.61	0.53	0.56	0.54	0.58	0.85
0.15	0.38	0.82	0.60	0.47	0.78	0.62	0.61	0.78
0.20	0.27	0.91	0.59	0.40	0.78	0.59	0.59	0.72
0.25	0.15	0.91	0.53	0.30	1.00	0.65	0.59	0.67

rather than a definitive perceptual benchmark, as the sample is small and inter-rater agreement is modest (Krippendorff's alpha (Krippendorff, 2011) equals 0.25 for style and 0.20 for object.)

Sensitivity to the number of neighbours. We additionally evaluate sensitivity to the neighbourhood size k. For each value of k, the category-specific threshold is recalibrated using the same leave-one-out procedure, so differences reflect only the choice of neighbourhood size. As shown in Table 19, the resulting texture rankings and pass/fail decisions remain stable over a broad range of k. Relative to $k = 5$, Spearman rank correlation remains at least 0.94 for all tested values, while at least 88% of evaluated textures retain the same verdict across the full range. Model-level ordering is also largely preserved. We therefore use $k = 5$ as a middle setting that avoids relying on a single nearest neighbour while limiting the influence of less relevant references in smaller categories.

Table 19: Sensitivity to the neighbour count k at $\alpha = 0.15$ over 1,760 scored agent textures in sprite-like categories. Thresholds are recalibrated by leave-one-out for each k. *Pass* is the agent pass rate, *Rank corr.* is the Spearman correlation of texture rankings with $k = 5$, *Same verdict* is the fraction retaining the same pass/fail decision, and *Model order* is the Spearman correlation of model-level pass rates with those at $k = 5$. The shaded rows indicate the setting used in the main experiments.

Metric	k	**Pass**	**Rank corr.**	**Same verdict**	**Model order**
Art Style	1	0.32	0.948	0.91	0.95
	3	0.32	0.993	0.97	1.00
	5	0.33	1.000	1.00	1.00
	10	0.37	0.989	0.94	0.98
	20	0.42	0.969	0.88	1.00
Semantic	1	0.37	0.943	0.88	0.93
	3	0.39	0.993	0.95	0.90
	5	0.41	1.000	1.00	1.00
	10	0.45	0.988	0.94	0.98
	20	0.49	0.960	0.88	0.98

Prompt Sample Provided to Agent for Mod Development

```
You are implementing a game mod inside a prepared, local mod
↪  project scaffold.

Task:
Add a Fire Sword item with registry id benchmod:fire_sword and 8
↪  attack damage. Melee attacks with it set the target on fire.
↪  Right-clicking shoots a fireball projectile. The sword has a
↪  shaped crafting recipe in a sword shape: a diamond on top, a
↪  diamond in the middle, and a Blaze Rod as the handle at the
↪  bottom (one vertical column). It appears in the Combat creative
↪  tab.

Requirements - the mod is evaluated against these:
- A Fire Sword item with id benchmod:fire_sword exists, named 'Fire
↪  Sword', and can be obtained from the Combat creative tab.
...
- The Fire Sword has a custom fire-themed item texture (orange/red
↪  flame colors) rather than a reused vanilla texture.

Rules:
- The mod project is in /workspace. Implement the task by editing
↪  it.
- Check that your code compiles with: cd /workspace/mod && gradle
↪  --no-daemon build
...
- Unit tests are yours to use and are not graded; test-only
↪  dependencies are
  ignored by the grader, so they will not break your submission.
- When the mod builds, loads, and implements the task, you are
↪  done.
```

Figure 11: Example prompt provided to the agent for an example world-editing task. The prompt specifies the requested modification, evaluation-facing behavioral requirements, project location, build command, and generic execution rules.

Sample Task Specification from IGMBench

```
{
  "uuid": "...",
  "tldr": "Adds a fire sword that sets what it hits alight and
  ↪  throws fireballs on right-click.",
  "game": "minecraft",
  "framework": "minecraft_fabric",
  "level": "L2",
  "instruction": "...",
  "expected_behaviors": [
    "A Fire Sword item with id benchmod:fire_sword exists, named
    ↪  'Fire Sword', and can be obtained from the Combat creative
    ↪  tab.",
    ...,
    "The Fire Sword has a custom fire-themed item texture
    ↪  (orange/red flame colors) rather than a reused vanilla
    ↪  texture."
  ]
}
```

Figure 12: Sample task specification from IGMBench. The task specification records benchmark metadata, the intervention level, the natural-language request, and the evaluation criteria describing the desired world edit.

Sample Evaluation Specification from IGMBench

```
{
  "verify": {
    "setup": [
      {
        "rcon": "kill @e[type=minecraft:item]"
      },
      {
        "rcon": "setblock 0 -59 0 minecraft:chest"
      },
      ...,
      {
        "rcon": "time set midnight"
      },
      {
        "bot": {
          "op": "physics",
          "enabled": false
        }
      },
      ...,
      {
        "rcon": "scoreboard players operation #fireball_damage
        ↪  firesword -= #fireball_after firesword"
      }
    ],
    "checks": [
      ...,
      {
        "name": "the fired fireball damages a target it hits",
        "rcon": "scoreboard players get #fireball_damage
        ↪  firesword",
        "rule": {
          "min": 1
        },
        "kind": "positive",
        "depends_on": [
          "benchbot fixture is present after setup"
        ],
        "tags": [
          "BOT_RCON_NOW"
        ]
      }
    ]
  }
}
```

Figure 13: Sample evaluation specification from IGMBench. The verification view defines the controlled game setup and executable checks used to determine whether the requested world edit is correctly realized.

F IGMBENCH DIVERSITY

Figure 14 summarizes the overall task-genre distribution in IGMBench, while Figure 15 shows the corresponding distributions for Minecraft and Terraria.

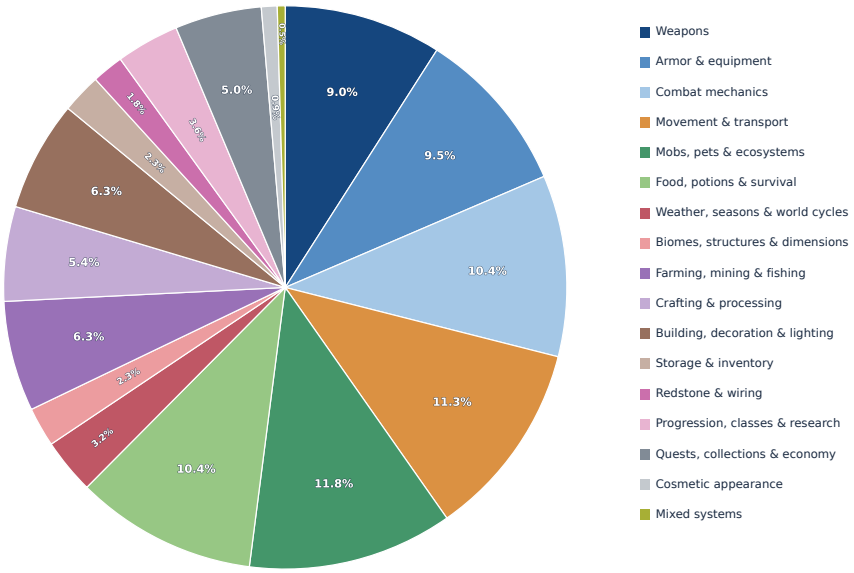
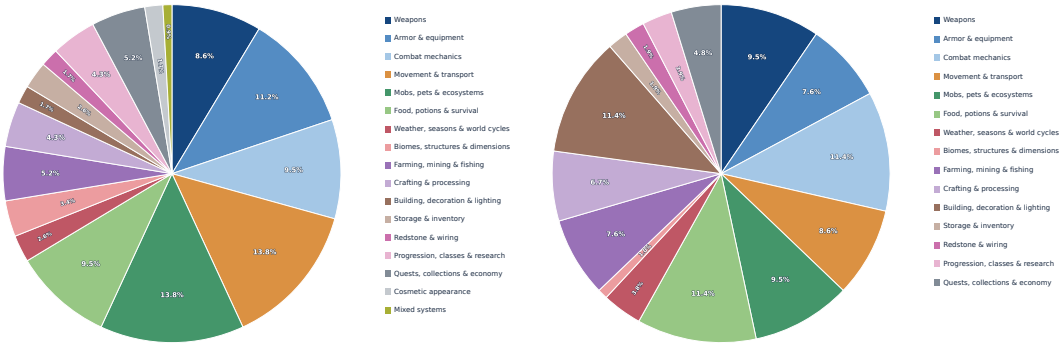


Figure 14: Task genre distribution across all 110 tasks in IGMBench. Percentages indicate the share of tasks in each genre.



(a) Minecraft (b) Terraria

Figure 15: Task genre distributions in IGMBench by game. (a) Minecraft (57 tasks); (b) Terraria (53 tasks). Percentages are computed within each game.