Agentic Game Development as a Verifiable Trajectory Data Engine for Scaling World Models

Pengfei Zhou[1,2] **Hexin Wang**[2] **Zhengfeiyang Zhang**[2] **Yixing Ma**[3] **Zhenglin Wan**[1]
Kaipeng Zhang[4] **Wangbo Zhao**[5‡] **Yang You**[1‡]

[1]National University of Singapore, HPC-AI Lab [2]InfRec, Cardinal AI Lab [3]University of California, Berkeley [4]Independent Researcher [5]Hong Kong University of Science and Technology

wangbo.zhao96@gmail.com; yangyou@nus.edu.sg ‡ Corresponding Author

Abstract A common strategy for scaling world models is to train on more crawled video with more compute. We argue that this strategy is inefficient: scaling world models also requires a recursive data engine that offers grounded reward signals. The success of code agents illustrates why this matters. As code is executable, compilers and runtimes can provide high-quality rewards for Reinforcement Learning (RL) post-training of LLMs. By contrast, spatial generation still relies largely on fuzzy proxies such as CLIP scores. These signals are fuzzy and biased, making them hard to support RL post-training. Compared with these, game development provides a missing reward environment for spatial world models. A scene encoded by a game engine is an executable world specification: the engine can efficiently check collision, physics, navigability and bounded playability, while the developer provides the global verification signal by judging whether the scene should be accepted. Game development also provides real-world long-horizon trajectory data for RL post-training. We therefore propose Reinforcement Learning with Human-Engine Verification (RLHEV), a post-training paradigm that combines dense engine signals with implicit human acceptance feedback from the development process. We apply this training objective to our proposed Agentic World Model (AWoMo): a world-building agent that proposes scene edits, observes human-engine verification, and converts accepted or repaired multimodal traces into training data. We evaluate the proposed approach through controlled experiments. On UnitySceneBench, a 200-example Unity asset-edit evaluation, our RLHEV obtains the highest score. In generalization, transfer learning helps with out-of-distribution shifts and gives positive signals in Unreal and Godot cross-engine experiments. AWoMo-augmented training also improves the embodied performance of the policy on R2R, Gymnasium MuJoCo, and D4RL Gym-MuJoCo. Agentic artifacts are released for reproduction: https://github.com/LanceZPF/cardinal-preview.

1. Introduction

Recent world-model work often treats spatial intelligence as a scaling problem: collect more scraped video, train larger models, and spend more compute [5, 6]. Code agents illustrate why this recipe is inefficient. Beyond foundation models pretrained on large-scale corpora, code agents have become one of the most important LLM applications today [82]. *Why are code agents winning the game?*

The key is that code is executable and functional, combining two signals that spatial generation usually separates. First, code can be executed: compilers, tests, and runtimes provide dense, low-cost feedback [36]. Second, the developer provides supplementary verification: a patch that passes tests can still be rejected because it breaks maintainability or fails the product goal in an agentic workflow. The post-training advantage of coding agents comes from this human-compiler dual verification.

Such verification feedback enables reinforcement learning to continuously improve the model after pretraining, incentivizing not only coding ability but also general reasoning and planning [26].

Spatial intelligence is on the other side. The quality of multimodal spatial outputs, such as generated videos or 3D scenes, is usually scored by fuzzy proxies such as JSD [45], FVD [70], CLIP similarity [40, 53, 78], and MLLM-as-judge scores [9, 43, 81]. These proxies are noisy, biased, and gameable [12, 23, 60]. Human judgment is indispensable, but if it is collected only as subjective preference labels over final outputs, it is too expensive and low-bandwidth to support the kind of iterative post-training loop that made code and reasoning scale.

Our central argument is that spatial intelligence is hitting a data-centric version of the Bitter Lesson [63, 87]. Scraping more data can improve coverage, but it does not create a reward engine that automatically provides high-quality supervision. We argue that game development provides such a recursive data engine: it records how humans build worlds, captures engine- and human-generated verification signals throughout the process, and converts these signals into post-training data.

Thus, we propose *Agentic World Model* (AWoMo) for automating game development: a world model embedded in a developer-centered agent workflow that proposes, renders, checks, revises, and learns from world-building traces [14]. AWoMo facilitates storing each prompt, scene program, rendered state, failure, fix, engine check, and human decision as a complete multimodal world-building trajectory. These trajectories enable recursive evolution: better models build more candidate worlds, human-engine verification supplies reward, and accumulated traces train the next model.

> **Game development matters for scaling world models since it can turn world construction into a recursive feedback engine.** Specifically, world models need a loop that observes how worlds are built, checks whether intermediate results work, and turns developer decisions into supervision, whereas game development offers such a loop. Engines can automatically test whether generated worlds are structurally valid, while developers judge whether they are truly qualified. This provides a dual verification signal: engine checks provide dense structural rewards, while developer feedback provides sparse but highly aligned judgments on acceptance. Based on this, we propose AWoMo and train it with RLHEV: Reinforcement Learning with Human-Engine Verification. Under RLHEV, game development trajectories give AWoMo a path similar to coding models: execution exposes local mistakes, humans judge global success, those signals improve the model, and improved models generate better candidates. This creates a possibility of a self-improving loop for world models, while unifying spatial understanding and generation capabilities.

For this argument, game development is regarded as a process by which world knowledge of humans (how objects rest and collide, how spaces connect, what makes a level traversable) is written into executable digital worlds. The built assets are useful, but the development trajectory is worth more: it records what the creator intended, what the state should be, what changed, whether engine checks passed afterward, and whether the human accepted the result. This claim is also falsifiable: if training on development traces and human-engine rewards does not improve OOD generalization, then game development cannot be treated as the missing data engine for world models.

Contributions. The primary contribution of this paper is an argument and a research agenda supported by controlled studies. First, we identify the *verifiability bottleneck* and argue that the slow progress of world model post-training stems from the absence of cheap and reliable reward (§2–§3). Second, we define AWoMo as a developer-centered agentic world model and propose game development as its practical recursive data engine, where executable world specifications and development traces jointly provide low-noise supervision (§4; Appendix A.2). Third, the experimental section first states

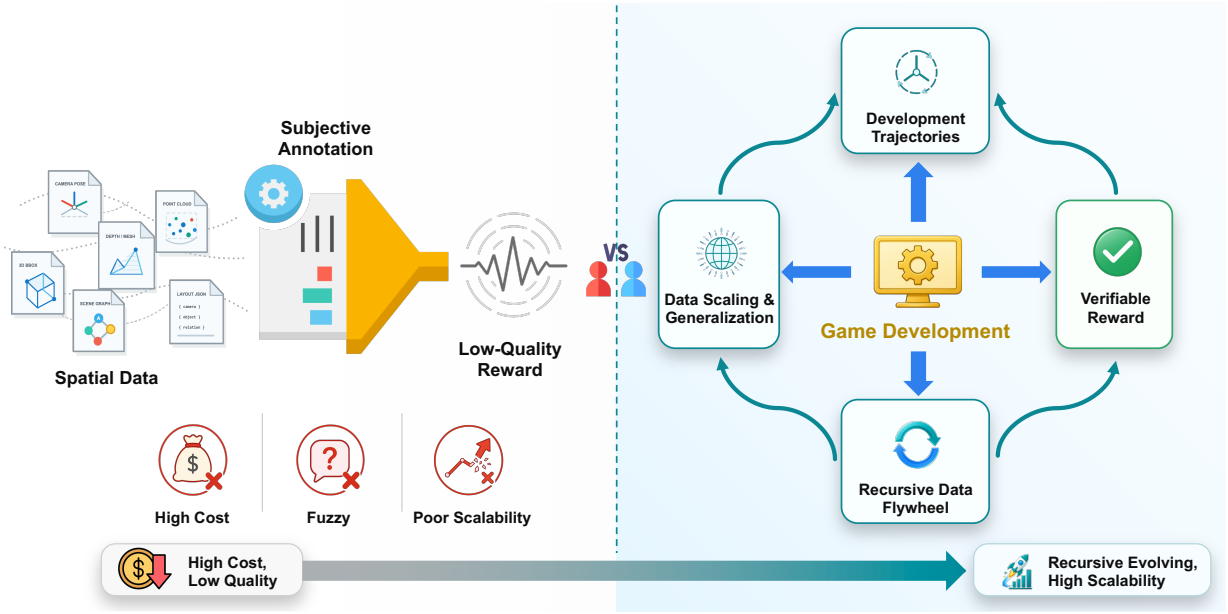

Figure 1 | From subjective reward proxies to a human-engine verification loop. Conventional spatial-data pipelines convert sparse human or model annotations into noisy final-output rewards. A game-development loop can instead pair human intent and feedback with engine execution, development trajectories, rendered evidence, and localized verifier failures, allowing future builders to improve from both grounded checks and developer judgment based on traces rather than final artifacts alone.

a validation roadmap (§5.1) and delivers experimental findings. Experimental results show that full human-engine feedback obtains the strongest result on *UnitySceneBench*, pretraining on source data helps target adaptation in distribution shift, cross-engine transfer also gives positive results on Unreal and Godot [20, 25], and AWoMo-generated environment data improves reported embodied metrics on R2R [3], Gymnasium MuJoCo [66, 68], and D4RL Gym-MuJoCo [22, 66] (§5–§6).

2. The Thesis of Verification

We first separate automatic verification from full task correctness. Engines can provide the former cheaply. The latter still belongs to human intent and acceptance.

Verifiers and verifiable reward. Let $\mathcal{X}$ be the input space and $\mathcal{Y}$ the output space. A *verifier* is an automatic evaluation function

$$V : \mathcal{X} \times \mathcal{Y} \to \mathcal{R}, \tag{1}$$

where $\mathcal{R} \subseteq \mathbb{R}$ denotes the reward space. Given an input x and a candidate output y, the verifier returns a reward $r = V(x, y)$ according to an explicit correctness criterion, such as program execution, theorem checking, or numerical comparison. A task is *automatically verifiable* if such a verifier can be evaluated at substantially lower cost than generating the output.

Efficient verifiers are *grounded*, *reliable*, and *robust*: they can derive rewards from explicit task specifications rather than rely on subjective preference proxies. Reinforcement learning from verifiable reward (RLVR) uses such verifiers to train reasoning models [26, 35, 38].

Human-engine verification. Previous spatial generation rarely has a complete automatic verifier. A game engine can report collider intersections, navmesh connectivity, script errors, or whether

a bounded playtest probe reaches a specified goal. However, it cannot decide by itself whether a generated cutscene has the right mood. We therefore define RLHEV as a new post-training framework. In the RLHEV, engine checks and test suites produce dense structural labels; human acceptance verifies the true qualification. Both signals are stored in a unified protocol of state-action-check-review trajectories for post-training.

The reward abstraction of RLHEV is a constrained selection over candidate outputs:

$$\max_{y \in \mathcal{Y}} \; U_H(x, y, h) - \sum_{i=1}^{n} \lambda_i(h)\, \phi_i(C_i(x, y)) \quad \text{s.t.} \quad G_j(x, y) = 1, \;\; j = 1, \ldots, m, \tag{2}$$

where $x \in \mathcal{X}$ is the task input, $y \in \mathcal{Y}$ is a candidate output such as a scene program or 3D asset, and $h \in \mathcal{H}$ is the human-review context. $U_H(x, y, h)$ is the human-review utility score, typically 0 or 1 for acceptance or rejection. $C_i(x, y) \in \mathbb{R}$ are engine diagnostic checks (collision penetration, etc.), $\phi_i \geq 0$ is a monotone penalty that vanishes when check i is clean, and $\lambda_i(h) \geq 0$ weights check i under context h. $G_j(x, y) \in \{0, 1\}$ $(j = 1, \ldots, m)$ are required hard gates from the engine runtime, where 1 denotes a pass. The invariant is the authority split: the engine offers dense, reproducible verification, while human review supplies the final judgment.

The recursive feedback loop. A verifier enables a closed optimization loop: the model proposes, feedback produces reward, the reward updates the model, and the improved model generates stronger candidates. Figure 1 summarizes the shift from fuzzy final-output rewards to this human-engine loop. The key property is that expensive human judgment can be grounded in cheap executable evidence rendered by a game engine, so humans can focus on final acceptance rather than on every collision or script failure. This mirrors professional game testing, where white-box tests, automated scripts, static checks, telemetry, and engine tooling coexist with black-box QA and human exploration. Self-play systems in Go and chess [58, 59] show how automatic feedback can compound improvement. As gameplay agents mature, this human-engine verification loop may further evolve into a more autonomous cycle, where one agent constructs virtual worlds while another explores, tests, and evaluates playability through gameplay, producing scalable self-improvement with less human intervention [30, 37, 48].

Why unverifiable signal caps data efficiency. Suppose the available reward R_{f} is a fuzzy proxy for ground-truth quality Q^*. Its error can be decomposed as

$$R_{\mathrm{f}}(x, y) - Q^*(x, y) = \varepsilon(x, y) + b(x, y), \tag{3}$$

where ε denotes zero-mean noise and b denotes systematic bias. Reward noise weakens the expected gain from each sample and lowers training efficiency at scale. Bias is more damaging: if R_{f} differs from Q^* along an exploitable direction, optimizing R_{f} can increase proxy reward while decreasing true quality [23, 60]. Scaling compute with such a proxy can therefore amplify the exploit. When the reward is poorly grounded, for learned capabilities the limiting factor is not only data or compute, but the fidelity and authority of the feedback signal. Engine checks improve fidelity for structural properties; human judgments preserve authority for usefulness.

Re-reading the Bitter Lesson. The Bitter Lesson [63] is often summarized as the claim that general methods which scale with compute eventually dominate hand-designed priors. This reading is incomplete for domains such as games, code, and mathematical reasoning [61, 72, 74]. In these settings, scalable learning is paired with evaluators that define success: game rules, program execution, and numerical checkers. The lesson is therefore that the most efficient gain often comes from the feedback channel, instead of the model architecture design itself. Once an efficient and scalable reward mechanism exists, computation can search for candidates that satisfy it. When such reliable

feedback is absent, scaling still improves imitation and perceptual plausibility, but it lacks a robust mechanism for ensuring correctness. This is the central obstacle for spatial intelligence.

The thesis. Progress in a domain is hindered less by the lack of data or compute than by whether the domain has a scalable feedback channel. For closed tasks, this can be a cheap grounded verifier. For open-ended spatial creation, it must be human-engine verification: executable checks for what can be formalized, human judgment for the remain. Domains with strong feedback channels (games, code, mathematics) saw cost-effective, compute-scalable progress. Domains without one (video generation, 3D synthesis, world modeling) saw imitation-bound progress that is judged largely by subjective annotations over final outputs. The question for spatial intelligence is therefore not merely where to get more data, but how to instrument a workflow where efficient executable verification can be found. Section 4 gives one answer: **we already build such verifiers that can help scale world models, and call them game engines.**

3. Why Current Spatial Data Fails to Support Scaling

By the criterion of §2, progress in current spatial world models is constrained by the lack of efficient verifiers for physical and geometric correctness. We illustrate this point with three examples.

Video generation: impressive, but weakly grounded. Modern video models produce realistic clips [5], and interactive variants now generate playable frames in real time [6, 50]. Yet no efficient, reliable verifier decides whether a generated video is physically and geometrically correct. *Is the physics consistent? Do objects persist under occlusion? Is the perspective coherent?* In practice, video generation quality is largely judged by Fréchet Video Distance [31, 70] and human raters. These are fuzzy proxies rather than objective correctness feedback, and they can be costly, noisy, and gameable [23]. With only fuzzy proxies over final outputs, post-training cannot be augmented with deterministic reward, leaving next-frame prediction over scraped videos as the dominant training signal. A model can therefore learn rendering statistics without learning the hidden world state that would make 3D editing and interaction reliable [76]. It has no efficient mechanism for deterministically verifying structural failures in its own output, so gains are paid for with more data and compute [16, 39]. Scaling data with distribution-based objectives alone still leaves common failures such as detail misalignment, violated physical constraints, or incoherent long-horizon dynamics unsolved.

3D generation: little data, no cheap label. Text-to-3D and image-to-3D have advanced quickly [47, 51], and neural scene representations now render in real time [34, 46]. But 3D generation is limited at the source: the largest curated 3D asset corpus holds on the order of 10^7 objects [17, 18], against the 10^9 to 10^{12} images and tokens behind 2D vision-language and language models. The larger problem is again the verifier problem. Some local geometry properties can be checked automatically, but a high-quality 3D asset or scene of *reality* still needs expensive collection, annotation and manual cleanup because there is no cheap joint verifier for physical plausibility, semantics, and task usefulness. Short on both data and reward, 3D generation often distills from 2D priors, borrowing a fuzzy signal from an adjacent unverifiable domain.

World simulators: the annotation wall. Learned simulators of the physical world [27, 28, 83] face the hardest version of the problem. Supervising a model of world simulation needs ground truth about the physical world (depth, geometry, dynamics, contact) at a density and accuracy that is very expensive to annotate. In our view, this is a central constraint on spatial intelligence: the verifiable signal for the real world is hard to scale because it is hard to produce cheaply. Any program that depends on real-world 3D ground truth inherits this limit.

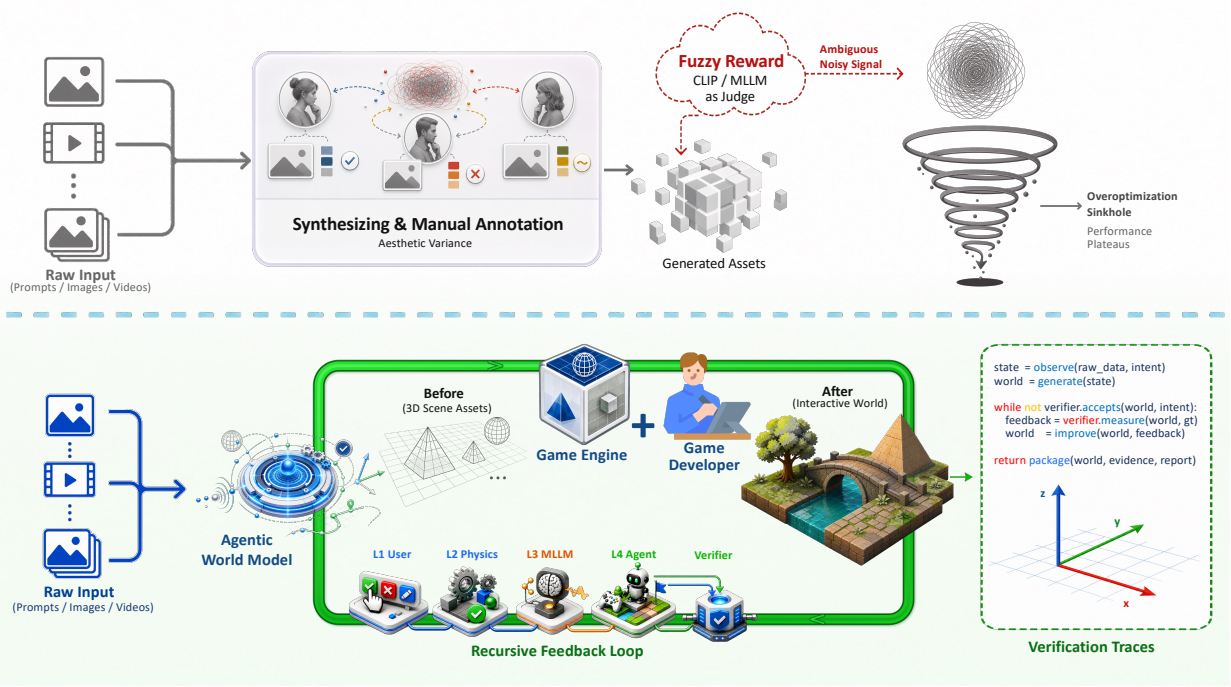

Figure 2 | Fuzzy reward in unverifiable spatial generation versus human-engine feedback in game development. The top pipeline illustrates how model-as-a-judge and human aesthetic variance can produce ambiguous rewards and performance bottleneck. The bottom pipeline turns assets into executable worlds: game engine and internal test suites expose local failures, rendered evidence supports review, and developer supplies the final signal for acceptance.

The unverifiability tax. Without a reliable verification channel, the field cannot benefit from the same post-training paradigm that drives rapid progress in code and reasoning. Progress therefore depends heavily on more data, compute, scans, and human annotations, which we refer to as the *unverifiability tax*. A natural response is simply to keep paying it by collecting more videos, 3D scans, and human ratings. Compared to scaling up language models, the cost is even more enormous.

We believe a more cost-effective alternative already exists in game development. Modern game engines provide efficient and grounded verifiers for structural properties of virtual worlds, including physics, collision, navigation, rendering constraints, and bounded playability. During game development, game engines and developers naturally form a dual-verification process. The engine verifies structural correctness, while developers judge whether the scene fulfills the world design. Grounded by engine evidence, human feedback becomes more objective and decisive. Thus, by combining efficient engine verification with developer feedback, game development offers a reliable feedback channel that can support large-scale post-training and recursive self-improvement of spatial intelligence.

4. Game Development as Human-Engine Verification

We argue that the practical feedback already exists in game development. A scene or project encoded for a modern engine (Unity, Unreal, Godot) is an executable specification; the engine is its interpreter, runtime, and partial verifier. And a practical verifier for *game development* is broader than the engine. It is the workflow in which developers define a world blueprint, inspect rendered states, run checks, playtest, critique, accept, or reject the result. Figure 2 contrasts this human-engine loop with fuzzy final-output reward pipelines. AWoMo names this coupled system: model, agent, engine verifier, and

human reviewer organized around world construction.

AWoMo design. Concretely, AWoMo is specified by four interfaces around an omni-modal world model. The *intent interface* receives the task brief, references, and design constraints; the *action interface* emits scene programs, asset edits, tool calls, and repair actions; the *verification interface* records engine checks such as loading, collision, physics stability, navmesh reachability, script execution, and bounded playability probes; and the *review interface* records developer acceptance, rejection, and critique. The execution loop is propose, render, verify, repair, and review: the model proposes an edit, the engine executes it and localizes failures, the agent issues repairs, and the loop terminates with a reviewer decision. Every loop is stored as a structured trace under the protocol introduced below, so the same workflow that builds a world also emits its own training data. Appendix A.1 details the interfaces, the execution loop, the trace design, and the core model architecture.

What an engine checks cheaply. As Figure 3 shows, engine checks are grounded, lower-variance signals that separate a verifier from a proxy: geometry and collision; physics and stability; navmesh reachability; script execution and soft-locks; and bounded reachability or objective-completion probes under specified agents and seeds. They are cheap relative to manual annotation and harder to game than appearance-based metrics: a collision query is not fooled by a plausible texture.

Game development as a rewardable mapping. *Why would a game verifier matter for spatial intelligence?* Because game development encodes shared world knowledge of humans (how things stack, how rooms connect, what is legible) into worlds precise enough to execute. If instrumented as the training data, games, editors, and modding ecosystems can therefore emit a difficulty-graded stream of verifiable spatial tasks whose checks inherit design intent while remaining scalable.

Unified World-Development Protocol (UWDP). A finished asset says what worked; a development trace says why it worked. Most game-development data are hard to reuse for model training because editor saves, assets, screenshots, logs, QA notes, and human decisions live in separate formats. They also lose the reason an edit was accepted or rejected. We propose UWDP: a typed multimodal protocol that turns ordinary game-development work into state-action-check-review traces spanning text briefs, scene programs, rendered evidence, logs, and reviewer decisions.

A compact UWDP instance is

$$u_t = (b, o_t, s_t, a_t, g_t, v_t, h_t, \rho_t), \tag{4}$$

where b is the prompt-based design intent; o_t is a stable object, relation, or scene-region identifier; s_t stores spatial, semantic, physical, and evidence-status fields; a_t is the edit or tool action; g_t is the engine or harness output; v_t is rendered evidence; h_t is the reviewer decision or critique; and ρ_t links repairs, confidence, costs, and residual risks. Table 1 gives an example.

UWDP collection algorithm. The protocol is simple: (1) parse the brief and references into typed objects, relations, evidence tags, and uncertainties; (2) build a proxy or asset edit in the engine; (3) record the engine snapshot, rendered evidence, and harness checks; (4) convert failures into typed repair actions; (5) iterate until the engine test pass and reviewer return *accept*; and (6) store the full trace with supervision. Figure 7 illustrates a trace recorded by our UWDP. The resulting event can supervise next-edit prediction, preference modeling, and RL state-action-reward updates.

Human acceptance protocol. Human review remains the final supervision. UWDP separates engine-checkable fields from human-only fields: brief alignment, visual legibility, and production fit. Human rejection or acceptance is kept as an escalated review and separate labels. This keeps engine feedback dense while preventing the engine from becoming a biased oracle.

The analogy to code. A game scene is a spatial artifact that an engine makes checkable, much as compiler and tests make code checkable. The engine is the compiler and automated playtest is the

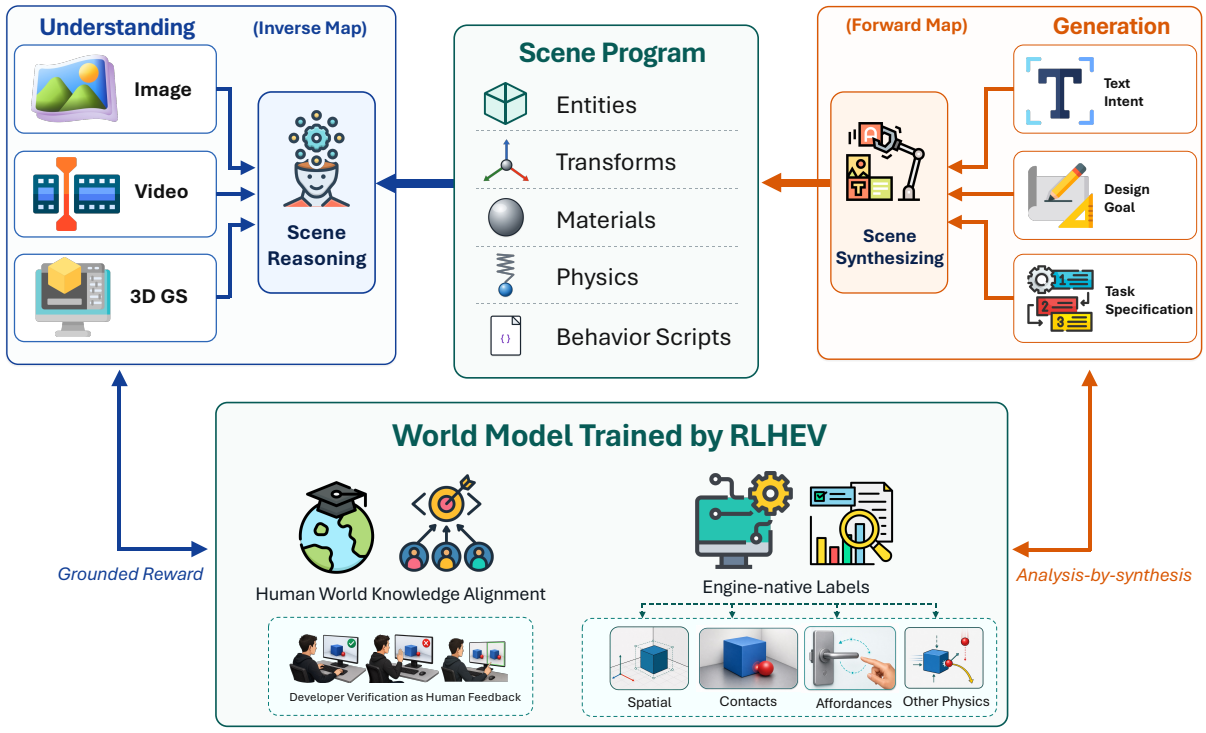

Figure 3 | A shared executable scene-program representation for understanding and generation. Game creation projects human world understanding, design intent, constraints, and playable experience into the model's understanding and generation capacity. Generation maps text intent, design goals, or task specifications to the scene program; understanding reads the same object in reverse. The engine checks it through rendering, physics rollouts, and temporal consistency, while humans judge whether the resulting world preserves intent and workflow value.

test suite. Code agents are not trusted by passing tests alone; they are trusted when delivering results align with the user's intent and review. Game development should be treated the same way. The engine signal is partial, but grounded enough to provide dense reward while keeping reward hacking measurable (§5); human review keeps the optimization anchored to actual usefulness.

A ladder of reward. The loop can begin with cheap checks and climb toward richer ones: *validity* (loads, manifold), *physical plausibility* (stable rollouts), *functional correctness* (navigable, objective achievable), and *playability* (agent and human players can successfully complete the game with target behavior). Procedural generation and automated playtesting already instantiate parts of this ladder [15, 44, 54, 62]. Unlike the static CLIP score, this framework can keep adding bounded engine probes and human criteria as the model improves.

The cost asymmetry. In the real world, a unit of verifiable 3D signal is among the most expensive data in machine learning. In the engine it is cheap by comparison, because the same computation that runs a world can grade it. Human feedback remains costly, but it becomes more efficient: a developer reviews rendered candidates after automatic filters. This inverts the economics of §3: labels are expected to become byproducts of regular development, inspection, and game testing.

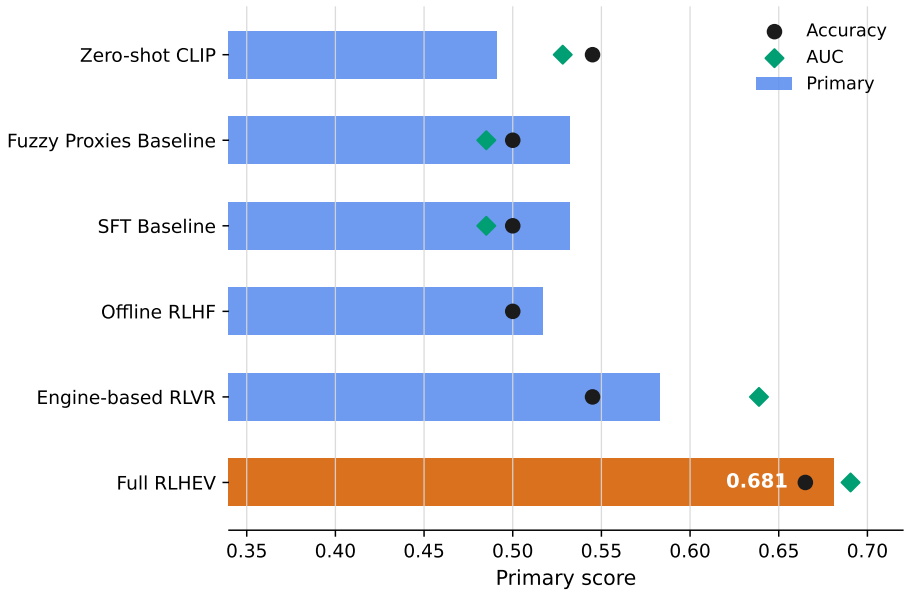

Figure 4 | Unity asset classification evaluation on *UnitySceneBench*. Bars represent the primary score, and markers denote accuracy and AUC. This figure reports the best performance for each method across eight random seeds. Full RLHEV obtains the highest best-of-eight primary score (0.681), with 0.665 accuracy, 0.665 balanced accuracy, 0.733 F1, and 0.690 AUC.

5. Experiments on Human-Engine Verification and Generalization

This section combines the validation roadmap with controlled experiments. The roadmap states the falsifiable claims first; the following subsections report pilot results of human-engine verification, generalization, and embodied diagnostic runs that test those claims.

5.1. Validation Roadmap and Falsifiers

To evaluate the proposed approach, we first design oracles with measurable explicit falsifiers. We then report completed controlled studies of *UnitySceneBench*, generalization, and embodied diagnostic runs, while providing findings for future studies.

Oracles.

P1 (Structural efficiency). At matched compute, reinforcement learning against engine-verifiable structural rewards gives better quality for validity, physics, and playability constraints than training against fuzzy proxies (CLIP, MLLM-as-judge scores, prior preference). *Falsifier:* fuzzy-reward training matches or beats verifiable-reward training at equal compute.

P2 (Improvement from human-engine). A generator can improve through game development trajectories of propose, render, verify, human-review, repair, and retrain, as engine checks local failures and humans judge global acceptance. *Falsifier:* adding human acceptance to engine traces does not improve task usefulness, sample efficiency, or robustness over engine-only traces.

P3 (Generalization). Capabilities learned against human-engine verified trajectories can generalize when the source and target share executable structure and compatible human goals. The transfer learning test is designed as: pretrain on source world, evaluate first on held-out within-family OOD shifts and then on harder shifts in different engines. *Falsifier:* no positive OOD transfer gain against

controlled baselines after pre-training on source data.

5.2. Experimental Setup

We evaluate AWoMo in a human-engine workflow: the world model generates or edits game assets inside an agentic harness; engine checkers and a human reviewer evaluate the outputs; and the resulting signals are used for post-training. For brevity, detailed experiment settings are described in Appendix B.

UnitySceneBench. The multimodal understanding experiment evaluates Unity asset classification for edited asset candidates [69] (Appendix B.2). The multimodal inputs include the text edit prompt, Unity asset/context features, reference-image features, and structured layout payloads. We refer to our benchmark as *UnitySceneBench*. The primary metric is

$$\text{Primary} = 0.45\,\text{balanced accuracy} + 0.25\,\text{accuracy} + 0.20\,\text{F1} + 0.10\,\text{AUC}. \tag{5}$$

Baselines and rewards. Zero-shot CLIP [53] is a strict CLIP-score classifier. The other main baselines are Fuzzy Proxies Baseline, SFT Baseline, Offline RLHF, and Engine-based RLVR (Appendix B.2). Human feedback is represented by accept/reject labels from the human-review channel (Appendix B.1); engine feedback comes from Unity-derived checks. The full model uses only human+engine reward, 0.65 human + 0.35 engine.

5.3. Evaluating Understanding and Generation on UnitySceneBench

Understanding-side evaluation. Figure 4 reports the best observed asset classification performance for each method across eight random seeds, where learned methods use 720 raw training instances. Zero-shot CLIP for classification is shown as a fixed reference. Under this best-of-eight view, Full RLHEV reaches the highest asset classification performance, beating the strongest non-full baseline by +0.098 primary score and +0.120 accuracy/balanced accuracy. Appendix B.2 reports the exact method definitions, evaluation protocol, and scaling details.

Scaling and generation-side evaluation. We next vary the amount of *UnitySceneBench* training data and use the same test splits. The base run evaluates our RLHEV and baselines across different training budgets and eight random seeds per method, with training budgets up to 720 instances. Figure 4 shows the best full-budget run for each method, while Figure 5(a) shows the mean and variance across all eight runs. As shown in Figure 5, the generation-side task tests the same scaling question under a generation manner. In the range up to 640 training instances, Full RLHEV reaches 0.8106 generation quality; with the full 720-instance training budget, Full RLHEV reaches 0.8197 generation quality versus 0.7934 for Engine-based RLVR.

5.4. OOD Generalization and Cross-Engine Transfer

Generalization evaluation. We next test whether the workflow transfers across data distributions and engines (Appendices B.3 and B.4). The benchmark covers Unity distribution transfer [69] and Unity-to-Unreal/Godot cross-engine transfer [20, 25]. The main score is the normalized MLLM-as-a-judge score in $[0, 1]$, supplemented by proxy and engine metrics. We note that cross-engine asset quality has no engine-native scalar that is directly comparable across Unity, Unreal, and Godot outputs, so a rubric-based judge is currently the only practical way to quantify relative quality on a common scale. The judge is therefore used in a restricted role: it follows the same accept/reject rubric as the human-review channel, every reported judge score is reviewed and verified by human inspection

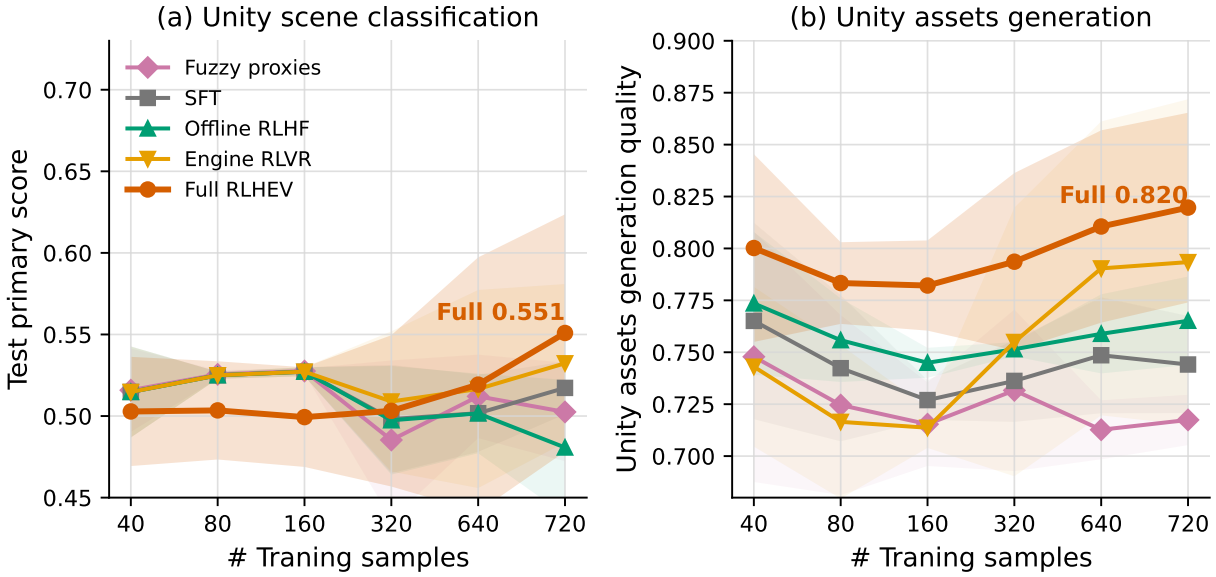

Figure 5 | Scaling with different numbers of training samples and tested on *UnitySceneBench*. (a) reports the primary score for Fuzzy Proxies Baseline, SFT Baseline, Offline RLHF, Engine-based RLVR, and Full RLHEV on the test set, as mean±std over eight seeds. (b) Unity assets generation uses the same training budgets and reports mean±std over eight seeds per method and budget.

(Appendix B.3), and it serves as an audited evaluation instrument rather than a training reward, which is the failure mode criticized in §1. We compare two conditions: target-only scratch trains only on the target-domain training split, while target-adapted transfer starts from a source-trained checkpoint and then adapts on the target-domain training split. Figure 6(a) reports the comparison.

Pretrained on the source improves the Unity distribution shift from 0.25 to 0.75 relative to training from scratch on the target distribution. Cross-engine transfer gives a smaller but still positive signal: Unity-to-Unreal rises from 0.25 to 0.35 and Unity-to-Godot from 0.15 to 0.35 relative to scratch, with lower loss and higher proxy scores.

5.5. Embodied Generalization Diagnostics

Embodied generalization. We also test whether AWoMo improves the performance of embodied policy trained through generative environment-data augmentation (Appendix B.5). The finalized benchmark covers R2R [3], Gymnasium MuJoCo [66, 68], and D4RL Gym-MuJoCo [22, 66]. It compares the original baseline, a naive augmentation baseline, and AWoMo-augmented training. Figure 6(b) reports direction-normalized improvement over the original baseline.

AWoMo-augmented training improves over the original baseline on all three reported metrics: +0.79% on R2R success rate, +9.96% on Gymnasium MuJoCo rollout return, and +48.43% on D4RL Gym-MuJoCo normalized score. This supports agentic environment-data generation as an effective method for improving policy in diagnostic embodied settings.

6. Discussion

These pilot studies support a focused claim: human-engine verification is a practical feedback source for training agentic world models. On *UnitySceneBench*, combining human acceptance with engine

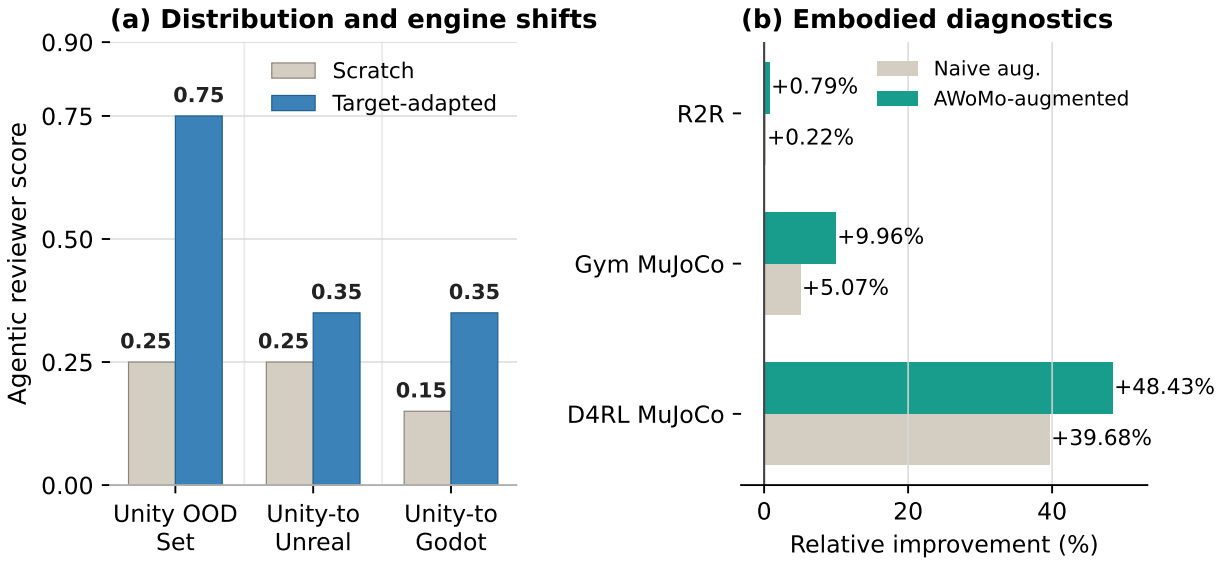

Figure 6 | Generalization and embodied diagnostic evaluations. (a) Generalization under distribution and engine shifts. Bars report the MLLM-as-a-judge score in $[0, 1]$ for training from scratch and target-adapted from a pretrained checkpoint. Pretrained on source data and adapted on the target distribution achieve significant gain; the cross-engine settings show smaller but measurable positive signal. (b) Embodied diagnostic evaluation on R2R, Gymnasium MuJoCo, and D4RL Gym-MuJoCo. Values are relative improvements over the original baseline after normalizing scores (positive is better for all benchmarks). Naive aug. denotes standard data augmentation, whereas AWoMo-augmented denotes the policy performance trained with AWoMo-generated data.

checks gives the best asset classification performance in the best-of-eight full-budget breakdown, reaching a 0.681 primary score with 0.665 accuracy, while the scaling curve reports seed-averaged robustness separately. As for generalization, pretrained on source data raises the performance on OOD distributions from 0.25 to 0.75, with positive cross-engine gains on Unreal (0.25 to 0.35) and Godot (0.15 to 0.35). In embodied diagnostics, AWoMo-augmented training improves the reported metrics by +0.79% on R2R, +9.96% on Gymnasium MuJoCo, and +48.43% on D4RL Gym-MuJoCo.

The strongest gains appear where the policy is trained on human-engine dual verification; current experiment results are positive but still diagnostic, requiring larger scaling studies to verify generalization capability. The next step is to connect playable artifacts, held-out target-engine checks, held-out human acceptance, and game-agent playtesting into a tighter loop; recursive self-improvement becomes possible. Appendix C.1 gives the detailed limitation analysis and future path.

7. Related Work

Neural game engines and interactive video world models. Genie, GameNGen, and iVideoGPT show that learned models can produce action-conditioned interactive rollouts from video-like data [6, 71, 79]. Newer systems such as GameFactory and GameGen-X extend this direction toward open-world and controllable game-video generation [8, 88]. This is the strongest contrasting route to ours [2, 56]. We view these systems as important interactive world models: they can respond to actions, but they typically do not expose the explicit state, scripts, collision semantics, or localized failure traces needed for recursive verification.

Structured and executable 3D generation. A closely related line of work represents 3D worlds as executable programs rather than opaque media artifacts. WorldCoder-Bench evaluates browser-native 3D worlds through executed Three.js programs and hidden runtime-state contracts [42]. VoxelCodeBench, 3DCodeBench, P3D-Bench, Code2Worlds, and PhysForge further examine code-based, parametric, dynamic, or physics-grounded 3D generation [24, 84, 85, 89, 90]. These works support our central distinction: visual plausibility is not the same as executable correctness. However, we argue that to generalize to more realistic scenes, 3D-native understanding and generation capabilities still need to be improved, which motivates the game-development data engine proposed in this paper.

Game development as an agentic verifier and data source. Another related line treats natural-language game specifications and real development workflows as executable environments for agents. WorldCoder learns code world models through interaction and uses them for planning and transfer [64], and agentic world modeling frames world models around agent decisions, environment dynamics, and self-revision under evidence [14]. GameDevBench studies real development tasks involving code changes, multimodal assets, shaders, animation, and gameplay logic [13]. GameGen-Verifier decomposes natural-language game specifications into keypoints, injects runtime states, and checks bounded behavior assertions [32]. Agent2World uses adaptive multi-agent testing to generate executable symbolic world models and keeps repair trajectories for training [29]. Our paper differs by making the game development itself the proposed data engine: the target object is not only a passed game or a score, but the object-linked edit/check/fix trace that can train the next builder.

Games as verifiable supervision and open-ended curricula. Games have long served as platforms for evaluating and training agents and embodied models, from ALE, Unity ML-Agents, and MineDojo to LLM/VLM-based systems such as Voyager and BALROG [4, 21, 33, 49, 73]. More recent work uses game rules or executable transitions as verifiable training signals: Game-RL uses game code and rules to synthesize multimodal verifiable data for VLM reasoning [67], while Game Code World Model distillation and RLVR-World use executable rules or transition checks as rewards [57, 80]. Related curriculum methods such as UED/PAIRED and POET evolve environments to expose agent weaknesses and improve generalization [19, 75]. We share this recursive-data intuition, but shift the focus from playing in worlds to building them: scene programs, verifier outputs, repairs, and accepted or rejected edits become the supervision for scalable world-model training.

8. Conclusion

Spatial intelligence needs scalable feedback, not just more data. Game development provides a practical source of that feedback for supporting the post-training of RLHEV: human intent and acceptance combined with engine checks, rendered evidence, and edit/check/review/fix traces. This points to an agentic world model that learns inside the workflow that builds, tests, repairs, and accepts worlds. The next future-work milestone is recursive self-improvement: agentic world models build executable worlds, game agents test and stress them, and the playtest results become training signal for the next generation of stronger world models. This is why game development matters for scaling world models: it provides an executable feedback loop in which world construction, verification, and learning can reinforce each other.

References

[1] Niket Agarwal, Arslan Ali, Jon Allen, Martin Antolini, Adeline Aubame, Alisson Azzolini, Junjie Bai, Maciej Bala, Yogesh Balaji, Josh Bapst, et al. Cosmos 3: Omnimodal world models for physical ai. *arXiv preprint arXiv:2606.02800*, 2026.

[2] Alibaba Token Hub. Happy Oyster: An Open-Ended World Model for Real-Time World Creation and Interaction. https://happyoyster.cn/, 2026.

[3] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3674–3683, 2018.

[4] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[5] Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Leo Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, et al. Video generation models as world simulators. *OpenAI Blog*, 1(8):1, 2024.

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

[7] Chameleon Team. Chameleon: Mixed-modal early-fusion foundation models. *arXiv preprint arXiv:2405.09818*, 2024.

[8] Haoxuan Che, Xuanhua He, Quande Liu, Cheng Jin, and Hao Chen. GameGen-X: Interactive open-world game video generation. In *The Thirteenth International Conference on Learning Representations*, 2025.

[9] Dongping Chen, Ruoxi Chen, Shilin Zhang, Yaochen Wang, Yinuo Liu, Huichi Zhou, Qihui Zhang, Yao Wan, Pan Zhou, and Lichao Sun. MLLM-as-a-judge: Assessing multimodal LLM-as-a-judge with vision-language benchmark. In *Proceedings of the 41st International Conference on Machine Learning*, pages 6562–6595. 2024.

[10] Shizhe Chen, Pierre-Louis Guhur, Makarand Tapaswi, Cordelia Schmid, and Ivan Laptev. Think global, act local: Dual-scale graph transformer for vision-and-language navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16537–16547, 2022.

[11] Xiaokang Chen, Zhiyu Wu, Xingchao Liu, Zizheng Pan, Wen Liu, Zhenda Xie, Xingkai Yu, and Chong Ruan. Janus-pro: Unified multimodal understanding and generation with data and model scaling. *arXiv preprint arXiv:2501.17811*, 2025.

[12] Zerui Cheng, Stella Wohnig, Ruchika Gupta, Samiul Alam, Tassallah Abdullahi, João Alves Ribeiro, Christian Nielsen-Garcia, Saif Mir, Siran Li, Jason Orender, Seyed Ali Bahrainian, Daniel Kirste, Aaron Gokaslan, Carsten Eickhoff, and Ruben Wolff. Position: Benchmarking is broken - don't let AI be its own judge. In *Advances in Neural Information Processing Systems*, 2025. Position Paper Track.

[13] 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. In *ICML 2026 Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD)*, 2026.

[14] Meng Chu, Xuan Billy Zhang, Kevin Qinghong Lin, Lingdong Kong, Jize Zhang, Teng Tu, Weijian Ma, Ziqi Huang, Senqiao Yang, Wei Huang, Yeying Jin, Zhefan Rao, Jinhui Ye, Xinyu Lin, Xichen Zhang, Qisheng Hu, Shuai Yang, Leyang Shen, Wei Chow, Yifei Dong, Fengyi Wu, Quanyu Long, Bin Xia, Shaozuo Yu, Mingkang Zhu, Wenhu Zhang, Jiehui Huang, Haokun Gui, Runyi Li, Chenyu Tang, Dong Huang, Xuhang Chen, Rui Liu, Chengzu Li, Shiyi Du, Xu Huang, Haoxuan Che, Long Chen, Qifeng Chen, Wenya Wang, Wenxuan Zhang, Xiaojuan Qi, Yang Deng, Yanwei Li, Mike Zheng Shou, Zhi-Qi Cheng, See-Kiong Ng, Ziwei Liu, Philip Torr, and Jiaya Jia. Agentic world modeling: Foundations, capabilities, laws, and beyond. *arXiv preprint arXiv:2604.22748*, 2026.

[15] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, pages 2048–2056. 2020.

[16] Justin Cui, Jie Wu, Ming Li, Tao Yang, Xiaojie Li, Rui Wang, Andrew Bai, Yuanhao Ban, and Cho-Jui Hsieh. Lol: Longer than longer, scaling video generation to hour. *arXiv preprint arXiv:2601.16914*, 2026.

[17] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, Eli VanderBilt, Aniruddha Kembhavi, Carl Vondrick, Georgia Gkioxari, Kiana Ehsani, Ludwig Schmidt, and Ali Farhadi. Objaverse-XL: A universe of 10m+ 3d objects. In *Advances in Neural Information Processing Systems*, pages 35799–35813, 2023.

[18] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13142–13153, 2023.

[19] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *Advances in Neural Information Processing Systems*, pages 13049–13061, 2020.

[20] Epic Games. Unreal Engine 5.8 documentation. https://dev.epicgames.com/documentation/unreal-engine/unreal-engine-5-8-documentation, 2026.

[21] 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. In *Advances in Neural Information Processing Systems*, pages 18343–18362, 2022.

[22] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[23] Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning*, pages 10835–10866. 2023.

[24] Yipeng Gao, Lei Shu, Genzhi Ye, Xi Xiong, Ameesh Makadia, Meiqi Guo, Laurent Itti, and Jindong Chen. 3dcodebench: Benchmarking agentic procedural 3d modeling via code. *arXiv preprint arXiv:2606.01057*, 2026.

[25] Godot Engine Contributors. Godot Docs – 4.7 branch. https://docs.godotengine.org/en/stable/, 2026.

[26] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081): 633–638, 2025.

[27] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pages 2450–2462, 2018.

[28] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025.

[29] Mengkang Hu, Bowei Xia, Yuran Wu, Ailing Yu, Yude Zou, Qiguang Chen, Shijian Wang, Jiarui Jin, Kexin Li, Wenxiang Jiao, et al. Agent2world: Learning to generate symbolic world models via adaptive multi-agent feedback. *arXiv preprint arXiv:2512.22336*, 2025.

[30] Sihao Hu, Tiansheng Huang, Gaowen Liu, Ramana Rao Kompella, Fatih Ilhan, Selim Furkan Tekin, Yichang Xu, Zachary Yahn, and Ling Liu. A survey on large language model-based game agents. *arXiv preprint arXiv:2404.02039*, 2024.

[31] Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, et al. Vbench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21807–21818, 2024.

[32] Chaobo Jia, Ruipeng Wan, Ting Sun, Weihao Tan, Borui Wan, Yuxuan Tong, Guangming Sheng, and Hong Xu. Gamegen-verifier: Parallel keypoint-based verification for llm-generated games via runtime state injection. *arXiv preprint arXiv:2605.07442*, 2026.

[33] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

[34] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):139:1–139:14, 2023.

[35] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025.

[36] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022.

[37] Zizhen Li, Chuanhao Li, Yibin Wang, Jianwen Sun, Yukang Feng, Jiaxin Ai, Fanrui Zhang, Mingzhu Sun, Yifei Huang, and Kaipeng Zhang. MeepleLM: A virtual playtester simulating diverse subjective experiences. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18678–18722, 2026.

[38] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024.

[39] Jie Liu, Gongye Liu, Jiajun Liang, Ziyang Yuan, Xiaokun Liu, Mingwu Zheng, Xiele Wu, Qiulin Wang, Menghan Xia, Xintao Wang, Xiaohong Liu, Fei Yang, Pengfei Wan, Di Zhang, Kun Gai, Yujiu Yang, and Wanli Ouyang. Improving video generation with human feedback. In *Advances in Neural Information Processing Systems*, pages 82155–82192, 2026.

[40] William Ljungbergh, Bernardo Taveira, Wenzhao Zheng, Adam Tonderski, Chensheng Peng, Fredrik Kahl, Christoffer Petersson, Michael Felsberg, Kurt Keutzer, Masayoshi Tomizuka, et al. R3d2: Realistic 3d asset insertion via diffusion for autonomous driving simulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 925–934, 2026.

[41] Jiasen Lu, Christopher Clark, Rowan Zellers, Roozbeh Mottaghi, and Aniruddha Kembhavi. UNIFIED-IO: A unified model for vision, language, and multi-modal tasks. In *The Eleventh International Conference on Learning Representations 2023, Kigali, Rwanda, May 1–5, 2023*. 2023.

[42] Shuo Lu, Yinuo Xu, Kecheng Yu, Siru Jiang, Yongcan Yu, Yubin Wang, Haitao Yang, Yuxiang Zhang, Bin Wang, Ran He, and Jian Liang. WorldCoder-Bench: Benchmarking physically grounded 3d world synthesis. *arXiv preprint arXiv:2606.01869*, 2026.

[43] Yiting Lu, Wei Luo, Peiyan Tu, Haoran Li, Hanxin Zhu, Zihao Yu, Xingrui Wang, Xinyi Chen, Xinge Peng, Xin Li, et al. 4dworldbench: A comprehensive evaluation framework for 3d/4d world generation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 34322–34332, 2026.

[44] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu based physics simulation for robot learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

[45] María Luisa Menéndez, Julio Angel Pardo, Leandro Pardo, and María del Carmen Pardo. The Jensen-Shannon divergence. *Journal of the Franklin Institute*, 334(2):307–318, 1997.

[46] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *ECCV 2020*, pages 405–421. 2020.

[47] Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-E: A system for generating 3d point clouds from complex prompts. *arXiv preprint arXiv:2212.08751*, 2022.

[48] Mingyu Ouyang, Siyuan Hu, Kevin Qinghong Lin, Hwee Tou Ng, and Mike Zheng Shou. GameWorld: Towards standardized and verifiable evaluation of multimodal game agents. *arXiv preprint arXiv:2604.07429*, 2026.

[49] Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. BALROG: Benchmarking agentic LLM and VLM reasoning on games. In *The Thirteenth International Conference on Learning Representations*, 2025.

[50] Jack Parker-Holder and Shlomi Fruchter. Genie 3: A new frontier for world models. https://deepmind.google/blog/genie-3-a-new-frontier-for-world-models/, 2025. Google DeepMind technical blog post, August 5, 2025.

[51] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. DreamFusion: Text-to-3d using 2d diffusion. In *The Eleventh International Conference on Learning Representations*. 2023.

[52] Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all. https://qwen.ai/blog?id=qwen3.6-35b-a3b, 2026. Model release note.

[53] 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. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8748–8763. 2021.

[54] Sebastian Risi and Julian Togelius. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence*, 2(8):428–436, 2020.

[55] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied AI research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9339–9347, 2019.

[56] Team Seedance, De Chen, Liyang Chen, Xin Chen, Ying Chen, Zhuo Chen, Zhuowei Chen, Feng Cheng, Tianheng Cheng, Yufeng Cheng, et al. Seedance 2.0: Advancing video generation for world complexity. *arXiv preprint arXiv:2604.14148*, 2026.

[57] Tyrone Serapio, Arjun Prakash, Haoyang Xu, Kevin Wang, and Amy Greenwald. Distilling game code world model generation into lightweight large language models. *arXiv preprint arXiv:2605.24375*, 2026.

[58] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[59] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[60] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems*, pages 9460–9471, 2022.

[61] Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. *Advances in Neural Information Processing Systems*, 38, 2026.

[62] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games*, 10(3):257–270, 2018.

[63] Richard S. Sutton. The bitter lesson. http://www.incompleteideas.net/IncIdeas/BitterLesson.html, 2019. Blog post.

[64] Hao Tang, Darren Key, and Kevin Ellis. WorldCoder, a model-based llm agent: Building world models by writing code and interacting with the environment. In *Advances in Neural Information Processing Systems*, pages 70148–70212, 2024.

[65] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. 2017.

[66] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012.

[67] Jingqi Tong, Jixin Tang, Hangcheng Li, Yurong Mou, Ming Zhang, Jun Zhao, Yanbo Wen, Fan Song, Jiahao Zhan, Yuyang Lu, et al. Game-rl: Synthesizing multimodal verifiable game data to boost vlms' general reasoning. *arXiv preprint arXiv:2505.13886*, 2025.

[68] Mark Towers, Ariel Kwiatkowski, John Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Kallinteris Andreas, Markus Krimmel, Arjun Kg, Rodrigo Perez-Vicente, et al. Gymnasium: A standard interface for reinforcement learning environments. *Advances in Neural Information Processing Systems*, 38, 2026.

[69] Unity Technologies. Unity documentation. https://docs.unity.com/en-us, 2026.

[70] Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphaël Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018.

[71] Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion Models Are Real-Time Game Engines. In *The Thirteenth International Conference on Learning Representations*. 2025.

[72] Vatsal Venkatkrishna, Indraneil Paul, and Iryna Gurevych. Aletheia: What makes RLVR for code verifiers tick? *arXiv preprint arXiv:2601.12186*, 2026.

[73] 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. *Transactions on Machine Learning Research*, 2024.

[74] Peng-Yuan Wang, Tian-Shuo Liu, Chenyang Wang, Ziniu Li, Yidi Wang, Shu Yan, Chengxing Jia, Xu-Hui Liu, Xinwei Chen, Jiacheng Xu, and Yang Yu. A survey on large language models for mathematical reasoning. *ACM Computing Surveys*, 58(8):209, 2026.

[75] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. POET: Open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 142–151. 2019.

[76] Thaddäus Wiedemer, Yuxuan Li, Paul Vicol, Shixiang Shane Gu, Nick Matarese, Kevin Swersky, Been Kim, Priyank Jaini, and Robert Geirhos. Video models are zero-shot learners and reasoners. *arXiv preprint arXiv:2509.20328*, 2025.

[77] Chengyue Wu, Xiaokang Chen, Zhiyu Wu, Yiyang Ma, Xingchao Liu, Zizheng Pan, Wen Liu, Zhenda Xie, Xingkai Yu, Chong Ruan, et al. Janus: Decoupling visual encoding for unified multimodal understanding and generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 12966–12977, 2025.

[78] Guanjun Wu, Jiemin Fang, Chen Yang, Sikuang Li, Taoran Yi, Jia Lu, Zanwei Zhou, Jiazhong Cen, Lingxi Xie, Xiaopeng Zhang, et al. Unilat3d: Geometry-appearance unified latents for single-stage 3d generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4366–4378, 2026.

[79] Jialong Wu, Shaofeng Yin, Ningya Feng, Xu He, Dong Li, Jianye Hao, and Mingsheng Long. iVideoGPT: Interactive VideoGPTs are scalable world models. In *Advances in Neural Information Processing Systems*, pages 68082–68119, 2024.

[80] Jialong Wu, Shaofeng Yin, Ningya Feng, and Mingsheng Long. RLVR-World: Training world models with reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 125312–125350, 2025.

[81] Hongchi Xia, Xuan Li, Zhaoshuo Li, Qianli Ma, Jiashu Xu, Ming-Yu Liu, Yin Cui, Tsung-Yi Lin, Wei-Chiu Ma, Shenlong Wang, et al. Sage: Scalable agentic 3d scene generation for embodied ai. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22358–22368, 2026.

[82] 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. In *Advances in Neural Information Processing Systems*, pages 50528–50652, 2024.

[83] Sherry Yang, Yilun Du, Seyed Kamyar Seyed Ghasemipour, Jonathan Tompson, Leslie Pack Kaelbling, Dale Schuurmans, and Pieter Abbeel. Learning interactive real-world simulators. In *The Twelfth International Conference on Learning Representations*. 2024.

[84] Yikang Yang, Zhanpeng Hu, Youtian Lin, Mengqi Zhou, Jingxi Xu, Feihu Zhang, Jiaheng Liu, and Yao Yao. P3d-bench: Benchmarking mllms for parametric 3d generation and structural reasoning. *arXiv preprint arXiv:2606.11152*, 2026.

[85] Yunhan Yang, Chunshi Wang, Junliang Ye, Yang Li, Zanxin Chen, Zehuan Huang, Yao Mu, Zhuo Chen, Chunchao Guo, and Xihui Liu. PhysForge: Generating physics-grounded 3d assets for interactive virtual world. *arXiv preprint arXiv:2605.05163*, 2026.

[86] Shunyu Yao, Tzu-Ming Harry Hsu, Jun-Yan Zhu, Jiajun Wu, Antonio Torralba, William T. Freeman, and Joshua B. Tenenbaum. 3D-aware scene manipulation via inverse graphics. In *Advances in Neural Information Processing Systems*, 2018.

[87] Dehao Ying, Fengchang Yu, Haihua Chen, Changjiang Jiang, Yurong Li, and Wei Lu. Beyond human annotation: Recent advances in data generation methods for document intelligence. *arXiv preprint arXiv:2601.12318*, 2026.

[88] Jiwen Yu, Yiran Qin, Xintao Wang, Pengfei Wan, Di Zhang, and Xihui Liu. GameFactory: Creating new games with generative interactive videos. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11590–11599, 2025.

[89] Yi Zhang, Yunshuang Wang, Zeyu Zhang, and Hao Tang. Code2Worlds: Empowering coding llms for 4d world generation. *arXiv preprint arXiv:2602.11757*, 2026.

[90] Yan Zheng and Florian Bordes. VoxelCodeBench: Benchmarking 3d world modeling through code generation. *arXiv preprint arXiv:2604.02580*, 2026.

[91] Chunting Zhou, Lili Yu, Arun Babu, Kushal Tirumala, Michihiro Yasunaga, Leonid Shamis, Jacob Kahn, Xuezhe Ma, Luke Zettlemoyer, and Omer Levy. Transfusion: Predict the next token and diffuse images with one multi-modal model. In *The Thirteenth International Conference on Learning Representations*. 2025.

[92] Gengze Zhou, Yicong Hong, Zun Wang, Chongyang Zhao, Mohit Bansal, and Qi Wu. SAME: Learning generic language-guided visual navigation with state-adaptive mixture of experts. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7794–7807, 2025.

[93] Pengfei Zhou, Xiaopeng Peng, Jiajun Song, Chuanhao Li, Zhaopan Xu, Yue Yang, Ziyao Guo, Hao Zhang, Yuqi Lin, Yefei He, et al. Opening: A comprehensive benchmark for judging open-ended interleaved image-text generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 56–66, 2025.

A. Supplementary Method Details

A.1. Agentic World Model (AWoMo)

Definition. AWoMo denotes our Agentic World Model system. It is not a standalone generator. It is a world model embedded in an agent workflow for building executable worlds. The system contains a model, an agent controller, a game-engine verifier, a reviewer, and a trace store. The model proposes world edits or assets; the agent executes them in the development environment; the engine checks structural constraints; the reviewer judges task fit and acceptance; the resulting assets are finalized game scenes, and the development trace can be recorded as training data.

System boundary. The general AWoMo framework is defined by four interfaces. The *intent interface* receives a multimodal task brief, references, and design constraints. The *action interface* emits scene programs, asset edits, tool calls, or repair actions. The *verification interface* records engine checks such as loading, collision, physics stability, navmesh reachability, and bounded playability probes. The *review interface* records reviewer acceptance, rejection, critique, and residual-risk notes. Thus, this boundary is important: AWoMo is the coupled workflow, not a single neural network.

Core model architecture and pretraining. The neural core inside the evaluated AWoMo instances is `UnifiedGameAssetModel`, a post-trained version of the latest Mixture-of-Transformers (MoT) model Cosmos 3 [1]. It is an omni-modal model built around modality-typed token streams, with game-asset heads for text specifications, rendered images, 3D Gaussian assets, meshes, and Unity, Godot, Unreal, and MuJoCo representations. The released checkpoint has 2.890B parameters, hidden width 3584, 8 transformer layers, 4 attention heads, and 4 gated experts; the gating routes each typed token stream through the expert mixture, so a single backbone serves both understanding-side reading and generation-side writing of scene programs. The model is initialized from Cosmos 3 [1] and obtained through on-policy distillation (OPD) followed by continued pretraining on the AWoMo manifest. The continued-pretraining run uses 8 A100 workers, global batch size 4096, gradient accumulation 2, and a production manifest with 79,130 training samples, 4,333 validation samples, and 4,282 test samples; the best checkpoint is selected at step 589 by validation loss. Baseline and evaluation details are given in Appendix B.1.

Pretraining data format and authorization. The AWoMo pretraining manifest contains 87,745 accepted samples and 504 rejected samples. Each entry is a UWDP-style multimodal record with a task brief, one or more typed inputs, typed outputs, optional engine tokens, and metadata for source, split, and sampling weight. Output modalities include text (87,745 instances), images (53,426), Unreal (9,678), Unity (7,423), 3D assets (3,194), meshes (3,194), MuJoCo (2,946), Godot (2,685), and 3D Gaussian assets (12). The developer-workflow data are collected through UWDP in an Agentic Game Development product. All developer-workflow data used in the experiments are developer-authorized; accepted manifest instances are kept only after the local allowlist and quality filters pass.

Execution loop. The basic loop is propose, render, verify, repair, and review. Given intent b, the agent produces an edit a_t and an executable state s_t. The engine returns check outputs g_t, rendered evidence

Table 1 | Example fields in one UWDP instance. The values illustrate how stable object IDs connect intent, scene state, engine checks, repairs, rendered evidence, and review.

Field	Example value	Functionality
b	Arrange a playable storage room with a clear path to the exit.	Conditions generation on design intent.
o_t	`crate_07`; relation `near(chair_03)`.	Localizes checks and repairs to objects or regions.
s_t	Transform, collider bounds, material tags, navmesh status, and visibility flags.	Grounds the model in executable scene state.
a_t	Move `crate_07` to $(2.4, 0.0, 2.6)$ and regenerate its collider.	Supervises next-edit and repair prediction.
g_t	Collision fail before repair; navmesh pass after repair.	Supplies verifier gates and dense rewards.
v_t	Rendered preview, object mask, and camera metadata.	Provides reviewer-visible evidence.
h_t	Reviewer marks *needs revision*, then *accept* after the repair.	Provides acceptance and preference signal.
ρ_t	Link from failed collision to repair action; edit cost; residual risk note.	Supports credit assignment and risk-aware selection.

v_t, and failure localization when checks fail. The agent can then issue a repair action, such as moving an object, replacing an asset, changing a collider, regenerating a layout region, or revising a script. The loop stops when the reviewer accepts, rejects, or requests further revision. In the intended full system, gameplay agents can also run bounded playtests before final review.

Trace design. AWoMo stores each loop as a UWDP trace rather than a final snapshot. A trace contains the design intent, stable object identifiers, intermediate scene states, actions, verifier outputs, rendered evidence, human decisions, repair links, costs, and residual risks. Stable identifiers let the model connect a failed check to the object or region that caused it and to the repair that fixed it. This makes the data closer to a code trajectory with tests and patches than to a static image-caption pair.

From traces to training data. Each stored trace is compiled into training instances in three forms. Accepted terminal states become supervised generation targets conditioned on the intent b; failed checks paired with their repair actions become next-edit and repair-prediction pairs; and the engine outputs g_t together with the reviewer decision h_t become the fused reward of RLHEV, with engine gates applied before the human-engine combination. Rejected traces are kept as well: they supply the negative half of the acceptance signal that final-artifact corpora usually lose.

Training signal. RLHEV is the training objective used for AWoMo when human and engine feedback are available. Engine checks supply dense local rewards or gates; human review supplies the acceptance target and veto. The same trace can also train supervised next-edit prediction, next-failure prediction, repair-success prediction, preference models, and candidate rankers.

RLHEV objective. Let $\mathbf{x}$ denote the multimodal state and task context, $\mathbf{a}$ is a generated edit or action sequence, $h(\mathbf{x}, \mathbf{a}) \in [0, 1]$ the human-review reward, and $e(\mathbf{x}, \mathbf{a}) \in [0, 1]$ the normalized engine reward. Let $\mathbf{g}(\mathbf{x}, \mathbf{a}) \in \{0, 1\}^m$ be binary engine gates and $\mathbf{1}$ the all-one vector. The gated scalar reward used by Full-RLHEV is

$$r_{\alpha,\beta}(\mathbf{x}, \mathbf{a}) = \mathbb{I}\{\mathbf{g}(\mathbf{x}, \mathbf{a}) = \mathbf{1}\} \left(\alpha h(\mathbf{x}, \mathbf{a}) + \beta e(\mathbf{x}, \mathbf{a})\right) - \lambda_c c(\mathbf{x}, \mathbf{a}), \tag{6}$$

where $c(\mathbf{x}, \mathbf{a}) \geq 0$ is an optional cost or risk penalty. In the UnitySceneBench main result, $\alpha = 0.65$, $\beta = 0.35$, and $\lambda_c = 0$ unless a candidate is rejected by a required gate. The Offline RLHF ablation sets $(\alpha, \beta) = (1, 0)$; Engine-based RLVR sets $(0, 1)$.

Let $\pi_\theta(\mathbf{a} \mid \mathbf{x})$ be the trained policy and π_0 the supervised reference policy. A standard regularized offline objective is

$$\max_\theta \ \mathbb{E}_{(\mathbf{x},\mathbf{a})\sim\mathcal{D}}\left[w(\mathbf{x},\mathbf{a})\left(r_{\alpha,\beta}(\mathbf{x},\mathbf{a}) - b(\mathbf{x})\right)\log \pi_\theta(\mathbf{a} \mid \mathbf{x})\right] - \lambda_{\mathrm{KL}}\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[D_{\mathrm{KL}}(\pi_\theta(\cdot \mid \mathbf{x}) \,\|\, \pi_0(\cdot \mid \mathbf{x}))\right], \quad (7)$$

where $w(\mathbf{x},\mathbf{a}) \geq 0$ is an offline importance or selection weight, $b(\mathbf{x})$ is a scalar baseline, and $\lambda_{\mathrm{KL}} \geq 0$ keeps the policy near the supervised model. The reported Unity experiment implements this principle through offline reward-weighted candidate training and held-out selection.

Remark on reward-estimation error. A standard uniform-error argument applies to the implemented reward: if the implemented reward deviates from the intended reward by at most ϵ everywhere, then the value gap of an η-optimal policy under the implemented reward is at most $2\epsilon + \eta$ under the intended reward.

A.2. Unifying Understanding and Generation of Scene Assets

As shown in Figure 3, our human-engine protocol is to place generation and understanding around the same scene-program representation. This perspective aligns with the broader trend toward unified models that perform multimodal understanding and generation within one framework [7, 11, 41, 91].

A scene is a program; the engine is its interpreter. Represent a 3D world not as pixels or an opaque latent but as a structured, executable specification: entities, transforms, materials, physical properties, and behavior scripts. The separated spatial understanding and generation tasks can therefore become inverse directions of one map:

- **Generation** is the forward map from design or synthesis intent to scene program or assets: produce a world that satisfies a goal.
- **Understanding** is the inverse map from observation of scene program, image, video, or scan to the reasoning results of the world scenes.

The two directions also share their supervision. The action that creates a generation target also creates much of its understanding supervision. The development protocol also adds the temporal and human dimensions: an observation can be paired not only with a final scene program, but with the edits, verifier failures, rendered previews, critiques, and accept/reject decisions that led to it. Passive real-world data rarely offers this density of causal labels; here it is a long-overlooked byproduct of running and reviewing the program.

Why this helps both directions. Understanding and generation are expected to complement each other when both operate on the same executable scene program [77, 93]. A model that generates stable, navigable scenes must learn the spatial structure needed to understand them; a model that recovers program intents from observations gains the prior needed to know how generated scenes behave correctly. This mirrors inverse-graphics systems, where an encoder recovers a structured scene representation and a decoder or renderer uses that representation for generation and manipulation [86].

Closing the world-model loop. The engine makes structural constraints cheap and repeatable to test, while human review decides whether the resulting world actually satisfies the intended purpose. The goal is therefore not to claim that game-engine success automatically transfers to all spatial domains. Rather, the claim is that executable feedback and human judgment together provide a scalable training loop for world models. Whether this loop generalizes beyond games still should be tested through further controlled studies.

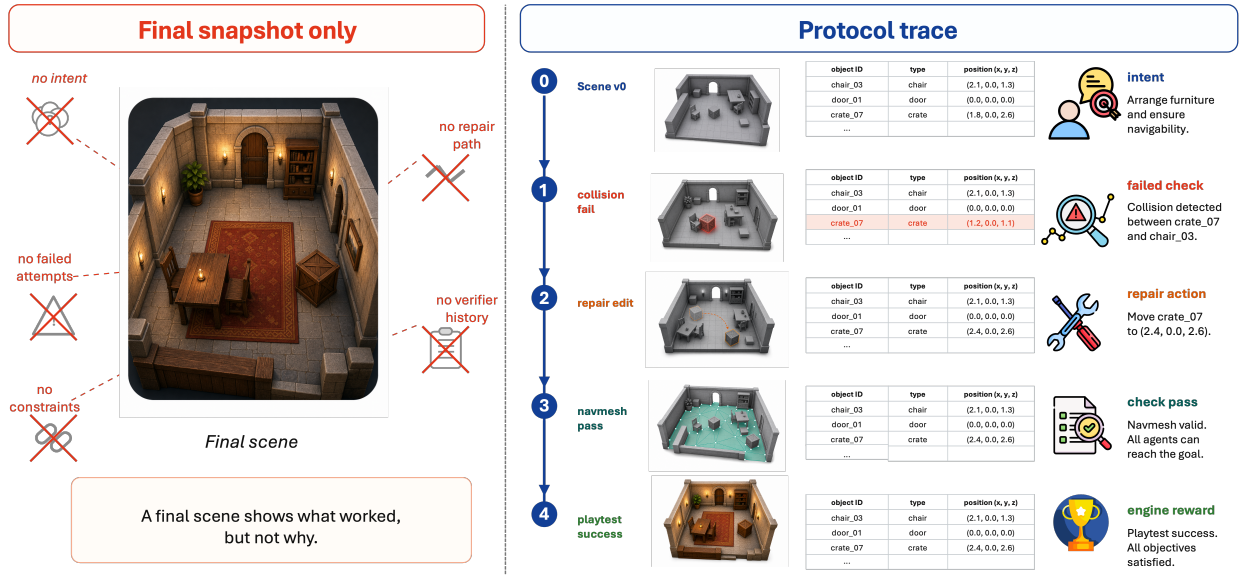

Figure 7 | Protocol traces preserve the process data that is absent from final snapshots. A final scene records only the terminal state, whereas a trace records intent, intermediate scene states, failed checks, repair edits, validation outcomes, and engine rewards. This object-linked sequence provides a training objective for asset diagnosis and repair.

A.3. Protocol Trace Analysis

UWDP Trace Illustration. Figure 7 illustrates why a final scene is not enough supervision. A snapshot shows what worked, but not intent, failed attempts, constraints, verifier history, or repair paths.

Source-Data Scaling. This analysis tests whether source-side workflow traces help target-engine ranking under a small target-label budget. The source split contains 720 Unity training examples, 80 validation, and 200 test examples [69]. The Godot and Unreal target splits [20, 25] each contain 720 training, 80 validation, and 200 test examples with target-engine labels. For each seed and target engine, the probe is trained under a deliberately scarce target-label budget: among the target-engine training examples, only 8 randomly sampled instances expose their target-engine pass/warn labels to the probe, and all remaining target instances are treated as unlabeled. This simulates the practical setting where a new engine has just been instrumented and almost no verified target-engine labels exist yet, so the probe must rely on transferred source-side signal. Under this budget, we vary the number of Unity source examples from 0 to 720.

Representations. The *snapshot-only* probe uses coarse final-output metadata such as format tags, prompt length, asset count, and aggregate output size. It does not use labels, deviation type, asset identity, or target-engine checks. The *protocol-trace* probe adds source-side workflow fields: Unity accept/reject label, deviation type, target-engine id, and used-asset identity. It does not use hidden target-engine check values or target-engine reason strings. Both probes are ridge-regression ranking models evaluated by Spearman correlation against held-out target-engine pass/warn labels.

Result. Figure 8 shows a clear representation gap. With development traces, snapshot-only features reach 0.159 ± 0.168 Spearman, while protocol traces reach 0.719 ± 0.094. Using 720 source instances, snapshot-only remains low (0.141 ± 0.084), while protocol traces reach 0.758 ± 0.044. The trace advantage stays around 0.56–0.62 across source sizes.

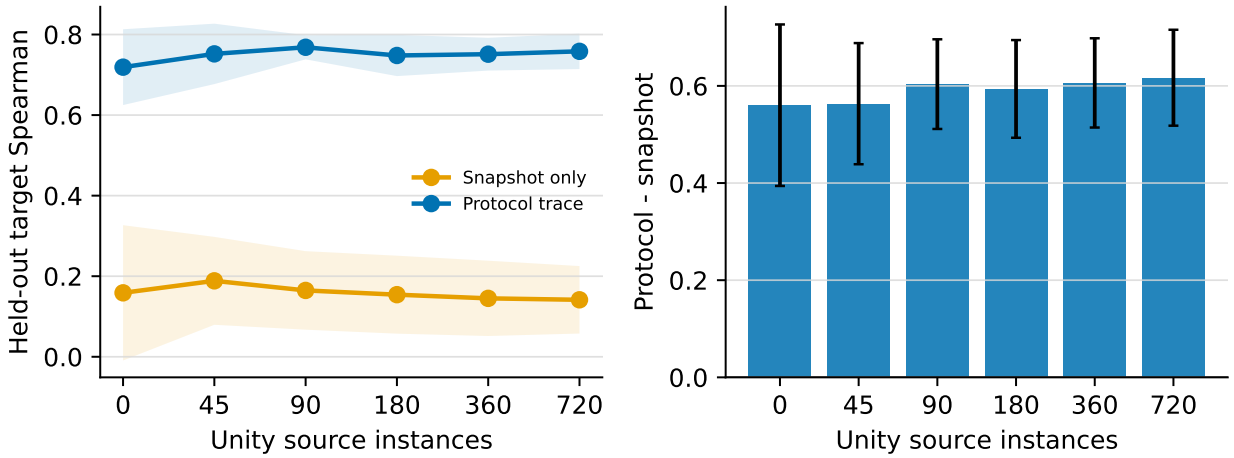

Figure 8 | Source-side trace transfer under an 8-label target budget. Values are held-out Spearman correlations over 8 seeds and two target engines. Protocol-trace features provide substantially stronger target-engine ranking signal than coarse final-snapshot features; increasing the number of Unity source instances gives a smaller additional gain.

B. Supplementary Experiment Details

B.1. Baselines, Benchmarks, and Evaluation Details

This appendix collects baseline, benchmark, and evaluation details for the reported experiments. It is intended to keep the main text concise while making the experimental claims auditable. Model architecture, pretraining, and RLHEV objective details are given in Appendix A.1.

Reviewer components and baselines. The zero-shot visual baseline uses CLIP [53]. The agentic MLLM-as-a-judge harness uses Qwen3.6-35B-A3B [52] under the same accept/reject rubric used by the human-review channel; its scores are used for evaluation and audit only, and all released judge scores were reviewed and verified by human inspection. Engine checks, held-out labels, and test examples remain separate from model inputs.

UnitySceneBench data. UnitySceneBench instances are constructed from the developer workflow (Appendix A.1) at a smaller evaluation scale. A task prompt and asset context are converted into a candidate edit, imported into Unity, rendered, checked by the engine harness, and assigned a reviewer accept/reject label.

Benchmarks and engines. Unity, Unreal, and Godot are cited through their engine documentation [20, 25, 69]. The embodied diagnostics use R2R [3], Gymnasium MuJoCo [66, 68], D4RL Gym-MuJoCo [22, 66], and SAME/DUET-style VLN trajectory evaluation [10, 92].

B.2. Full-RLHEV Validation Details

This appendix describes the full-RLHEV evaluation on *UnitySceneBench*.

Task and data. The task is binary asset classification for edited Unity asset candidates [69]: the model predicts whether a candidate edit is accepted or rejected. The input includes edit prompts, Unity asset/context features, reference-image features, candidate-layout features, and optional CLIP features [53]. The target label and engine verdict are excluded from the model input. The split has 720 training examples, 80 validation examples, and a 200-example test set with 100 accepted and

Table 2 | Method definitions for the *UnitySceneBench* comparison.

Name	Definition
Zero-shot CLIP	A visual-proxy baseline. It classifies the acceptance or rejection of candidates using a threshold on CLIP similarity, without a learned task model or RL.
Fuzzy Proxies Baseline	The baseline model trained with CLIP similarity rewards. It tests whether visual-proxy rewards help the supervised baseline.
SFT Baseline	The baseline model trained only with supervised GT accept/reject labels from the training split. The GT is the binary accepted/rejected target label. It is excluded from the model input, and no RL reward is used.
Offline RLHF	Offline RL using only the human-review reward, without engine reward.
Engine-based RLVR	Offline RL using only engine-verification reward, without the human-review reward.
Full RLHEV	The full model trained with fused human-review and engine rewards, using weights 0.65 and 0.35.

Table 3 | Unity assets generation quality for Figure 5. Values are mean quality over eight seeds per method for each training budget.

Method	40	80	160	320	640	720
Fuzzy Proxies Baseline	0.7478	0.7246	0.7154	0.7316	0.7127	0.7174
SFT Baseline	0.7651	0.7423	0.7270	0.7362	0.7486	0.7441
Offline RLHF	0.7735	0.7558	0.7449	0.7515	0.7589	0.7652
Engine-based RLVR	0.7430	0.7165	0.7137	0.7550	0.7904	0.7934
Full RLHEV	**0.8002**	**0.7833**	**0.7821**	**0.7936**	**0.8106**	**0.8197**

100 rejected candidates.

Rewards and baselines. Zero-shot CLIP [53] is used as a fuzzy proxy. Human feedback is represented by accept/reject labels from the human-review channel, while engine feedback comes from Unity-derived checks. The full-RLHEV setting combines only human and engine reward with weights 0.65 and 0.35, respectively. Appendix B.1 describes the reviewer-channel convention. Table 2 gives the definition of baseline methods.

Evaluation protocol. For Figure 4, the learned methods are trained with the 720 raw instances and evaluated on the same 200-instance test split under eight random seeds. The figure reports the best observed full-budget test performance for each method across these eight runs, so it should be read as a best-of-eight metric breakdown rather than a seed-averaged robustness estimate. Figure 5 reports the complementary mean±std view across training budgets and summarizes seed-level stability.

The generation-side protocol uses the same training budgets rather than a generated-candidate horizontal axis. The 720 full-RLHEV generation run produces 640 fresh Unity plans and labels with Unity exit code 0 and no missing render/snapshot/layout/engine-label artifacts. Table 3 breaks down the same-protocol generation quality for Figure 5.

Claim boundary. This experiment evaluates a bounded asset classification verifier for Unity asset edits. In the best-of-eight full-budget breakdown, Full RLHEV is the strongest method, with a +0.098 primary-score gain and +0.120 accuracy/balanced-accuracy gain over the strongest non-full baseline. The multi-seed scaling curve in Figure 5(a) should be used for seed-averaged robustness. The result does not imply final game quality or broader human preference alignment.

B.3. Generalization Experiment Details

This appendix describes the generalization evaluation.

Benchmark categories. The benchmark is intentionally split into two forms of generalization.

Table 4 | Generalization aggregate. Target means target-adapted transfer; scratch means target-only scratch training.

Category	Transfer	Condition	Assert	Proxy	Engine acc.	Loss
Distribution	Unity → held-out Unity	zero-shot	0.25	0.451	0.007	11.685
Distribution	Unity → held-out Unity	target	**0.75**	0.449	0.008	11.590
Distribution	Unity → held-out Unity	scratch	0.25	0.421	0.005	11.909
Cross-engine	Unity → Unreal	target	0.35	0.474	0.008	11.596
Cross-engine	Unity → Unreal	scratch	0.25	0.437	0.009	11.825
Cross-engine	Unity → Godot	target	0.35	0.478	0.007	11.575
Cross-engine	Unity → Godot	scratch	0.15	0.441	0.008	11.820

- *Distribution Transfer*: Unity source distribution to held-out Unity examples [69]. The split contains 5,895 source examples and 1,254 target examples.
- *Cross-Engine Transfer*: Unity source assets to Unreal and Godot targets [20, 25, 69]. Each source/target split contains 1,000 examples with 720/80/200 train/validation/test partitions.

Training conditions. Target-only scratch uses only the target-domain training split and does not initialize from a source-domain checkpoint. Target-adapted transfer first trains on the source-domain split, then adapts that checkpoint on the target-domain training split. All reported scores are measured on the target-domain test split. When reported, zero-shot evaluates the base checkpoint directly, and source-only evaluates the source-trained checkpoint without target adaptation. For the cross-engine boosted runs, target-adapted transfer starts from the Unity source checkpoint and runs 160 target-adaptation steps on the target-engine training split.

Metrics. The main metric is a normalized MLLM-as-a-judge score in $[0, 1]$. The MLLM-as-a-judge harness judges whether the generated asset is usable, engine-ready, and aligned with the task. There is currently no engine-native scalar metric that is directly comparable across Unity, Unreal, and Godot outputs, so this rubric-based judge is the only practical common scale for relative quality across engines. To keep the judge trustworthy, it follows the same accept/reject rubric as the human-review channel, and every reported judge score was reviewed and verified by human inspection before release; the judge is used for evaluation only and never as a training reward. We also report proxy scores and loss as independent checks alongside the MLLM-as-a-judge score. Table 4 reports the aggregate values. For cross-engine transfer, the target entries report target-adapted transfer.

Interpretation. Distribution transfer is the strongest positive case. Target adaptation raises the MLLM-as-a-judge score from 0.25 to 0.75 on the Unity distribution shift.

Cross-engine transfer also shows positive signal. Pretraining on the source and applying target-adapted transfer improves Unity-to-Unreal from 0.25 to 0.35 and Unity-to-Godot from 0.15 to 0.35 relative to scratch. Proxy scores and losses also improve. We use these results as generalization evidence in a transfer learning manner: source-engine traces provide useful initialization, and target-engine adaptation turns them into improved target-engine signal.

B.4. Cross-Engine Generalization

The cross-engine evaluation tests whether a Unity-trained asset workflow transfers to Unreal or Godot. Unity [69], Godot [25], and Unreal [20] are evaluated with their corresponding target-engine labels.

Setup. The source and target splits each contain 1,000 instances with a 720/80/200 train/validation/test split. Multimodal inputs include text, image, Unity, and mesh modalities. Outputs are text plus the target engine representation. The conditions follow Appendix B.3: scratch uses only the target-engine training split, while target-adapted transfer starts from the Unity source checkpoint and adapts on the

Table 5 | Cross-engine target-adapted transfer results. Target-adapted transfer gives measurable positive signal over scratch for both target engines, with higher proxy scores and lower losses.

Transfer	Condition	MLLM judge	Proxy	Loss
Unity → Unreal	zero-shot	0.25	0.396	12.377
Unity → Unreal	Unity-source only	0.15	0.005	16.693
Unity → Unreal	target-adapted transfer	0.35	0.474	11.596
Unity → Unreal	scratch	0.25	0.437	11.825
Unity → Godot	zero-shot	0.25	0.396	12.375
Unity → Godot	Unity-source only	0.15	0.005	16.695
Unity → Godot	target-adapted transfer	0.35	0.478	11.575
Unity → Godot	scratch	0.15	0.441	11.820

Table 6 | Final embodied generalization results. Values are mean±std where available. Δ is the direction-normalized relative improvement of AWoMo-augmented training over the original baseline.

Benchmark	Metric	Original	Naive aug.	AWoMo-aug.	Δ	Best
R2R	Success rate ↑	76.006±0.133	76.176±0.175	76.607±0.094	+0.79%	AWoMo
Gymnasium MuJoCo	Rollout return ↑	1568.44±1756.69	1648.03±1689.15	1724.73±1642.60	+9.96%	AWoMo
D4RL Gym-MuJoCo	Normalized score ↑	18.295±14.663	25.555±12.448	27.155±9.491	+48.43%	AWoMo

target-engine training split. Table 5 reports the target-adapted transfer result for each target engine.

Result. Target-adapted transfer raises Unity-to-Unreal from 0.25 to 0.35 and Unity-to-Godot from 0.15 to 0.35 relative to scratch. It also improves proxy score and loss for both target engines. These instances support the paper's engineering claim: an agentic world model should carry engine-specific protocol traces and calibrate to the target runtime before deployment.

B.5. Embodied Generalization Details

This appendix describes the embodied generalization evaluation. The reported scope covers R2R [3], Gymnasium MuJoCo [66, 68], and D4RL Gym-MuJoCo [22, 66]. Each benchmark compares three methods: the original baseline, a naive augmentation baseline, and AWoMo-augmented training.

Method definition. AWoMo denotes our proposed method of Agentic World Model. In the embodied experiment, AWoMo is not a standalone policy architecture; it is a profile-guided data-augmentation workflow driven by the released AWoMo checkpoint profile. For R2R, the workflow scores existing PREVALENT augmentation trajectories with a selector derived from the checkpoint profile, matches the resulting subset to the target path/instruction distribution with scan balancing, and fine-tunes the SAME policy on it. For Gymnasium MuJoCo and D4RL Gym-MuJoCo, the workflow synthesizes candidate state-action pairs by profile-guided augmentation and filters them with automatic acceptance checks before policy training.

Metrics. The three benchmarks use different metrics, so Figure 6(b) reports direction-normalized improvement over the original baseline. R2R uses success rate, Gymnasium MuJoCo uses rollout return, and D4RL Gym-MuJoCo uses the D4RL normalized score; higher is better for all three main metrics. Table 6 reports the raw values.

Result. AWoMo-augmented training improves over the original baseline on all three reported metrics. The largest relative gain is on D4RL Gym-MuJoCo, where the normalized score improves by +48.43%. Gymnasium MuJoCo rollout return improves by +9.96%, and R2R success rate improves by +0.79%. The R2R experiment uses SAME/DUET trajectories [10, 92] on the val-unseen split. The D4RL experiment uses Gymnasium MuJoCo rollouts with the official D4RL normalization constants.

C. Supplementary Discussion

C.1. Limitation Discussion and Future Analysis

We test the proposed approach against its strongest objections.

C1: Games are not reality (the sim-to-real gap). Correct; the gap is the central test, not an assumption. Simulation already transfers some policies to reality through domain randomization and large-scale training [44, 55, 65]; here the question is which executable structures survive OOD shifts. Our current experiments do not contain real scans, real robots, or a genuine real-to-sim-to-real loop, so they should not be read as evidence that game-engine rewards already transfer to the physical world. The next right test is explicit: use real-world environment, adapt with a little bit finetuning, and measure which executable checks remain useful. Until then, games are a scalable verifier-rich substrate, not a solved bridge to reality.

C2: Video generation is already advancing fast. It is, and the effect is impressive [5, 50]. But the progress is imitation- and compute-bound, judged by argument and fuzzy aggregates, with no self-improvement loop. Its typical failures are the physical and long-horizon inconsistencies. Progress on plausibility is not progress on correctness.

C3: The engine reward can be gamed too. Yes. A game engine is a partial verifier, not an oracle. A loose collision, navigation, or budget check can be optimized around if it is used alone. This is not a reason to abandon verifiers. It is a reason to use the combination of verifier ensembles, randomized probes, held-out checks, and human review. Engine rewards are still useful because their failures are localizable and testable rather than only perceptual.

C4: Real-world 3D data will get cheap. Cheaper data is not a cheaper verifier. A scan shows what one instance looked like without annotation. It also cannot tell whether a new synthesized output is correct. The engine supplies that missing automatic check; passive capture does not.

C5: Source-engine traces may overfit to one engine. Different engines encode different import formats, collision semantics, navigation systems, and runtime constraints, so target calibration matters. The cross-engine study (Appendix B.4) shows measurable positive signal for Unity-to-Unreal and Unity-to-Godot transfer after target adaptation, with stronger proxy scores and lower losses. The result supports the role of source-engine traces as useful initialization data and target runtime calibration as the step that turns them into target-engine signal.

Evidence and validation boundary. The experiments give bounded evidence for the roadmap. Human-engine feedback is strongest on the *UnitySceneBench* asset classification and generation tasks, and target adaptation plus agentic environment data improve the reported generalization and embodied diagnostics. These results do not yet prove complete game quality, human-subject validity, or a closed-loop embodied deployment. The next validation step is to implement playable artifacts and consented human studies.

Future paradigm and path. The long-term goal is a recursive game-building loop. An agentic world model builds an executable game; game agents play it, find broken rules, unreachable goals, physics failures, and weak pacing; those failures become training signal for the next world model. The main difficulty is making the loop trustworthy: the game specification must be executable, the playtest agents must be hard to fool, and human review can still judge design intent. A practical path is to start with bounded playable levels, add playtest agents and failure localization, then scale to multi-scene games with new engines, agents, and human reviewers.

C.2. Broader Impacts

This paper advocates a research direction rather than releasing a model, so its impacts are indirect. If this approach is effective, grounding spatial generation in verifiable engine reward would lower the cost of training spatial intelligence systems and embodied AI (with benefits for environment simulation and policy training) and would shift effort from scraping large real-world corpora toward computation against automatic verifiers, with attendant data-governance benefits. The same capabilities carry the usual dual-use risks of better world and video generation, including more convincing synthetic media. An engine-grounded pipeline is, if anything, easier to audit than an opaque generative model, since its outputs are produced against explicit, inspectable checks. At the same time, real development traces may contain creator intent, proprietary assets, debugging decisions, and project IP; deployed data engines should therefore use opt-in collection, project-level filtering, redaction of irrelevant content, access controls, and local or federated training paths for sensitive projects. Finally, raising capability through verifiable reward makes verifier design more important: a misspecified engine reward would be optimized just as literally as a fuzzy one.

The approach also reframes recursive self-improvement. Much recent discussion imagines capability growth arising primarily from AI systems training, evaluating, and improving other AI systems. Our thesis points instead to a hybrid loop in which AI systems participate in human productive processes and learn from the executable traces those processes generate. In game development, human spatial knowledge is continually externalized into model capabilities; agentic systems can contribute to this process while learning from it. Recursive improvement is therefore not only an AI-to-AI phenomenon, but can also be an ecosystem property emerging from the interaction between human practice, executable environments, and learning systems. Whether such socially grounded feedback loops scale beyond game development to broader domains of engineering, science, and physical-world interaction is an open research question.

Looking further ahead, the same loop suggests a demand-side flywheel. As scaling-law-driven models keep making software and content production dramatically cheaper, a growing share of human time and ambition is freed to move into digital worlds, where people can design, build, and inhabit virtual environments that express their own ideas and dreams. Each of these creators can work through an agentic world-building pipeline of the kind described in this paper, in which intent, edits, engine checks, and acceptance decisions are recorded as reusable interactive 3D construction traces. World-building activity then compounds: cheaper creation invites more creators, more creators produce a superlinearly growing corpus of verified construction traces, and those traces in turn accelerate the scaling of the next generation of world models. In this long-term view, the data engine is not a fixed corpus but a growing economy of human creativity in virtual worlds, whose natural byproduct is exactly the verified supervision that world models need.